Introdução

Há três anos, a palavra "java" evocava imagens de uma ilha da Indonésia ou de uma xícara de café quente. Atualmente, qualquer um que já tenha dado uma olhada em uma página da World Wide Web, revista de informática ou na seção de negócios de um jornal, sabe da existência de outra espécie de Java — a linguagem de programação introduzida pela Sun Microsystems.

Existem filmes bombásticos, atletas em orgias criminosas e políticos malcomportados que recebem menos propaganda e atenção da mídia do que a linguagem Java. Esta linguagem está se tornando mais conhecida como uma ferramenta de desenvolvimento de software do que como a fatídica ilha vulcânica.

Um tipo de Java mantém os programadores acordados à noite. O outro permite que eles descansem após desenvolver software.

A linguagem de programação Java se tornou proeminente no final de 1995, depois que a Netscape a licenciou para uso no navegador Navigator. Atualmente, existem várias opções para a inserção de programas interativos na Web, incluindo Macromedia Shokwave e Microsoft ActiveX. Quando a linguagem Java foi introduzida, ela revolucionou a natureza das páginas da Web. Os applets Java — pequenos programas que são executados dentro do navegador, como parte de uma página — são usados para vários objetivos: jogos, multimídia, animação e outros efeitos especiais. De acordo com o mecanismo de pesquisa Alta Vista, mais de um milhão de páginas na World Wide Web incluem applets Java.

Com o lançamento da versão 2, a linguagem Java ampliou seu alcance para além da World Wide Web. A linguagem, que é tão popular na Internet, também pode ser usada para o desenvolvimento de software de propósitos gerais.

Talvez você esteja familiarizado com as ferramentas de programação Java, como Symantec Visual Café e SunSoft Java Workshop. Esses programas tornam possível desenvolver programas Java funcionais, mas a melhor maneira de compreender a abrangência total da linguagem é trabalhando diretamente com ela no Java Development Kit, da Sun. O kit, que está disponível gratuitamente na Web, no endereço http://java.sun.com, é um conjunto de ferramentas de linha de comando para escrever, compilar e testar programas em Java.

E aí que se enquadra este livro. Você conhecerá todos os aspectos do desenvolvimento de software Java usando a versão mais atual da linguagem e as melhores técnicas disponíveis.

Quando terminar, você conhecerá muito bem as razões pelas quais a linguagem Java se tornou a linguagem de programação mais comentada da última década e por que ela pode ser a mais popular da próxima.

Como este livro está organizado

Este livro aborda a linguagem Java e suas bibliotecas de classe em 21 dias, organizados em três semanas separadas. Cada semana aborda uma área diferente do desenvolvimento de applets e aplicativos Java.

Na primeira semana, você aprenderá sobre a linguagem Java em si:

- O Dia 1 é a introdução básica: o que é Java, por que ela é legal e como obter o software necessário para se criar programas Java. Você também criará seu primeiro aplicativo Java.
- No Dia 2, você explorará os conceitos básicos da programação orientada a objetos, à medida que eles se aplicarem à linguagem Java.
- No Dia 3, você começará a entrar nos detalhes com os blocos básicos de construção da linguagem Java — tipos de dados, variáveis e expressões, como aritmética e comparações.
- O Dia 4 entrará em detalhes de como tratar objetos na linguagem Java — como criá-los, como acessar suas variáveis e chamar seus métodos, e como compará-los e copiá-los. Você também verá pela primeira vez as bibliotecas de classe Java.
- No Dia 5, você aprenderá mais sobre Java com arrays, instruções condicionais e loops.
- O Dia 6 explorará integralmente a criação de classes — os blocos de construção básicos de qualquer programa Java — e como criar um aplicativo Java (um programa que pode ser executado sozinho, sem um navegador da Web).
- O Dia 7 complementará o que você aprendeu no Dia 6. Você aprenderá mais a respeito de como criar e utilizar métodos, incluindo anulação e sobrecarga de métodos e a criação de construtores.

A semana 2 é dedicada principalmente aos applets e às técnicas de programação gráfica, como animação e tratamento de imagens:

- O Dia 8 fornecerá os fundamentos dos applets — como eles diferem dos aplicativos, como criá-los e sobre as partes mais importantes do ciclo de vida de um applet. Você também aprenderá a criar páginas HTML que contenham applets Java.
- No Dia 9, você aprenderá sobre as classes Java para desenhar figuras e caracteres na tela — incluindo a cobertura das novas classes Java2D, introduzidas na versão 2 da linguagem Java.
- No Dia 10, você criará programas multimídia que utilizam imagens, som e seqüências de animação. Você também terá sua primeira experiência com multithread — um modo de fazer seus programas tratarem de várias tarefas simultaneamente.

- O Dia 11 iniciará uma exploração de quatro dias da programação visual. Você aprenderá a criar um interface gráfica com o usuário para applets, usando o Abstract Windowing Toolkit, um conjunto de classes introduzido na linguagem Java 1.02.
- O Dia 12 tratará de como fazer uma interface com o usuário ter boa aparência, usando gerenciadores de layout, um conjunto de classes que determina como os componentes de uma interface serão organizados.
- O Dia 13 continua a abordagem do Abstract Windowing Toolkit, com classes de tratamento de eventos, que permitem a um programa responder a cliques de mouse e a outras interações do usuário.
- No Dia 14, você aprimorará seus conhecimentos sobre o AWT, aprendendo a desenvolver janelas instantâneas, menus e aplicativos independentes.

A semana 3 incluirá tópicos avançados e uma introdução ao Swing, a nova expansão de programação visual do Abstract Windowing Toolkit:

- No Dia 15, você aprenderá mais sobre interfaces e pacotes, que são úteis para o agrupamento de classes e para a organização de uma hierarquia de classes, assim como outros aspectos avançados do núcleo da linguagem em si.
- O Dia 16 tratará das exceções — erros, avisos e outras condições anormais, gerados pelo sistema ou por você, em seus programas. Você também aprenderá mais sobre a segurança no Java.
- O Dia 17 tratará da entrada e saída com o uso de fluxos, um conjunto de classes que permite o acesso a arquivos, acesso de rede e outros tratamentos de dados sofisticados.
- No Dia 18, você ampliará seu conhecimento sobre os fluxos para escrever programas que se comuniquem com a Internet, incluindo programação de soquete e tratamento de URL.
- O Dia 19 tratará de alguns dos recursos mais novos e sofisticados da versão 2 da linguagem Java, incluindo JavaBeans, Remote Method Invocation e outros assuntos.
- O Dia 20 iniciará uma discussão de dois dias sobre o Swing, as notáveis novas classes que fazem parte da linguagem Java 2. Você utilizará essas classes para criar a interface com o usuário de um aplicativo, fazendo uso de recursos como aparência e comportamento variáveis.
- No Dia 21, você concluirá seu conhecimento de Swing, transformando uma interface gráfica com o usuário em um aplicativo funcional que responde a eventos de usuário.

Sobre este livro

Este livro ensina tudo sobre a linguagem Java e como usá-la para criar applets para a World Wide Web e para aplicativos autônomos. Quando terminar este

livro, você saberá o suficiente sobre Java e sobre suas bibliotecas de classe para fazer praticamente qualquer coisa — dentro ou fora de um applet.

Quem deve ler este livro

Este livro ensina a linguagem Java para três grupos:

- Pessoas iniciantes, relativamente novatos na programação
- Pessoas que já conhecem Java 1.1 ou 1.02
- Projetistas experientes em outras linguagens, como Visual C++, Visual Basic ou Delphi

Você aprenderá a desenvolver applets; também aprenderá sobre os programas Java interativos que são executados como parte de uma página da World Wide Web e aplicativos, que são executados em qualquer parte. Quando você concluir este livro, poderá lidar com qualquer aspecto da linguagem e se sentirá suficientemente à vontade com Java para entrar em seus próprios projetos de programação ambiciosos — na Web ou fora dela.

Se você ainda é razoavelmente iniciante em programação ou se nunca escreveu um programa, pode estar se perguntando se este é o livro correto para ler. Como todos os conceitos deste livro são ilustrados com programas que funcionam, você poderá trabalhar no assunto independentemente de seu nível de experiência. Se você entende o que são variáveis, loops e funções, poderá tirar proveito deste livro. O tipo de pessoa que poderá querer ler este livro inclui você, se uma das características a seguir for verdadeira:

- Você é um ás em HTML, entende programação de CGI em Perl, Visual Basic ou alguma outra linguagem e quer passar para o próximo nível de design de página.
- Você teve noções de BASIC ou Pascal na escola, tem uma idéia do que é programação e já ouviu falar que Java é fácil de aprender, é poderosa e legal.
- Você já programou em C e C + + durante alguns anos, ouviu falar bem de Java e quer comprovar o fato.
- Você ouviu falar que Java é excelente para programação da Web e quer ver como é seu desempenho quando da sua utilização para outro desenvolvimento de software.

Se você nunca viu a programação orientada a objetos, o estilo de programação incorporado pela linguagem Java, não precisa se preocupar. Este livro não pressupõe nenhum conhecimento de design orientado a objetos e você tem a chance de aprender essa importante estratégia de desenvolvimento, enquanto estiver aprendendo Java.

Se você é completamente iniciante, o livro poderá ir um pouco mais rápido do que seu ritmo. Contudo, a linguagem Java é boa para se começar e se você

abordá-la lentamente e trabalhar nos exemplos, ainda poderá escolhê-la e começar a criar seus próprios applets Java.

Como este livro está oraanizado

Este livro se destina a ser lido e absorvido durante o decorrer de três semanas. Durante cada semana, você lê sete capítulos que apresentam conceitos relacionados à linguagem Java e à criação de applets e aplicativos.

Convenções

As notas apresentam informações técnicas interessantes.

As dicas mostram formas práticas de fazer algo.

Estes quadros advertem o leitor de problemas em potencial e ajudam-no a evitar problemas.

Os novos termos são acompanhados por caixas Novo termo, com o novo termo em itálico.

Um ícone Digite identifica algum código Java novo que você mesmo pode digitar. Você também pode obter o código a partir do site da Web referente ao livro.

Um ícone Entrada acompanha prompts de comando.

Um ícone Saída mostra a saída de um programa Java.

O texto que você digita e o texto que deve aparecer em sua tela são apresentados em tipo monoespaçado:

```
Ele ficará assim
```

O texto que você digitar também aparecerá em negrito.

Essa fonte imita o modo como o texto ficará em sua tela. Os locais reservados para variáveis e expressões aparecerão em tipo *itálico monoespaçado*.

O final de cada capítulo oferece perguntas normalmente feitas a respeito do assunto do dia, com respostas dos autores.

SEMANA 1

Iniciando a programação Java

1 Introdução ao Java

2 Uma amostra da programação orientada a objetos

3 Fundamentos do Java

4 Lições sobre objetos

5 Listas, lógica e loops

6 Criando classes

7 Usando métodos para realizar tarefas

SEMANA 1

Dia 1

Introdução ao Java

Isto representa o resultado final de quase 15 anos de tentativas de se alcançar uma linguagem e um ambiente de programação melhores para a criação de software mais simples e confiável.

— Bill Joy, co-fundador da Sun Microsystems

O *sineiro maluco estava certo. Dá para ganhar dinheiro em um lugar como este.*

— The Man with No Name, *A Fistful ofDollars*

Quando a linguagem de programação Java foi lançada publicamente pela primeira vez, em novembro de 1995, ela era muito semelhante a um pistoleiro de Clint Eastwood perambulando em uma inóspita cidade do oeste norte-americano.

Assim como Clint, Java era algo que as pessoas nunca tinham visto antes. Era uma linguagem de programação que podia ser executada em uma página da World Wide Web, destacando-se entre suas correlatas para imagens gráficas, texto, áudio e outros sinais onipresentes ainda "em criação". As pessoas vinham de longe — principalmente através de conexões na Internet, mas às vezes pessoalmente ao Silicon Valley — para dar uma espiada na linguagem.

Às vezes, a lenda da linguagem Java se distancia um pouco da realidade:

- "Os programas Java são executados sem problemas em diferentes plataformas de computador sem modificação!"

- "A linguagem Java acaba com a hegemonia dos sistemas operacionais da Microsoft!"
- "A linguagem Java transforma os autores de livros em celebridades queridas internacionalmente!"

Algo semelhante ocorre com a lenda dos pistoleiros cinematográficos de Clint:

- "Ele come balas de revólver no café da manhã!"
- "Ele bate tanto que seu avô teve uma contusão!"
- "Ele consegue matar um homem num piscar de olhos!"

Entretanto, nos últimos três anos a linguagem Java cumpriu grande parte do que se esperava dela inicialmente. A versão de Java 2 atesta o contínuo sucesso e crescimento da linguagem de programação.

A partir de hoje, você aprenderá sobre a linguagem Java e por que ela obteve sucesso rapidamente no setor da computação. Você usará Java 2 — a versão mais atual quando da redação deste livro — para criar aplicativos que podem ser executados em seu computador pessoal e através de uma rede como a Internet. Você também criará programas que são executados em páginas da Web usando Java 1.0.2, a versão que primeiro se tornou popular e ainda é suportada pelo maior número de navegadores.

Quando você chegar ao final deste livro, provavelmente perceberá o outro modo pelo qual a linguagem Java é como Clint Eastwood... Java é legal.

Não no sentido "acabei de sair do banho e onde, diabos, está minha toalha" ou "Jonathan Taylor Thomas é tão rico e nem mesmo sabe disso" ou ainda "sou uma estrela do rap e posso resmungar 75 adjetivos depreciativos sobre as mulheres em um minuto".

Java é legal porque se trata de uma linguagem extraordinária que torna mais fácil para muitos programadores realizarem coisas fantásticas. Java é legal porque torna conceitos complicados, como a programação orientada a objetos, muito mais fáceis de entender.

Assim como a palavra "salsa", Java é legal porque é divertido pronunciar em alto e bom som. A pronúncia é muito mais agradável do que exclamar "Visual Basic", "C plus plus", "Algol" ou "Mumps".

Se você estudar os 21 dias deste livro, irá se tornar um especialista na totalidade dos recursos da linguagem Java, incluindo imagens gráficas, entrada e saída de arquivos, design de interface com o usuário, tratamento de eventos e animação. Você vai escrever programas que podem ser executados em páginas da Web e outros que podem ser executados em seu PC.

Os objetivos de hoje são razoavelmente modestos. Você aprenderá sobre os seguintes tópicos:

- O que Java é atualmente e como chegou lá.

- Por que vale a pena aprender Java e por que é uma forte concorrente das outras linguagens de programação.
- O que você precisa para começar a escrever programas Java — o software, as habilidades e alguma terminologia básica.
- Como criar seu primeiro programa Java.

Com base na imensa propaganda sobre Java nos últimos anos e pelo número enorme de livros sobre esta linguagem (1.054, segundo os cálculos *da javaWorld)*, você poderia ter uma impressão deturpada do que a linguagem é capaz de fazer.

Java é uma linguagem de programação muito conveniente para o desenvolvimento de software que funcione em conjunto com a Internet. Ela também é uma linguagem de programação orientada a objetos que utiliza uma metodologia que está se tornando cada vez mais útil no mundo do design de software. Além disso, ela é uma linguagem multiplataforma, o que significa que seus programas podem ser criados para executar do mesmo modo no Microsoft Windows, Apple Macintosh e na maioria das versões de UNIX, incluindo a Solaris. A linguagem Java vai além da área de trabalho, sendo executada em dispositivos como televisões, relógios de pulso e telefones celulares. O JavaStation, o computador de rede da Sun, executa o sistema operacional JavaOS e é otimizado para a linguagem.

A linguagem Java está mais próxima das linguagens de programação populares, como C, C + +, Visual Basic e Delphi, do que de uma linguagem de descrição de página, como a HTML, ou de uma simples linguagem de scripts, como a JavaScript.

Nascida para execução... na Web

A linguagem Java é mais conhecida por sua capacidade de executar em páginas da World Wide Web. Os navegadores Netscape Navigator e Microsoft Internet Explorer podem carregar por download um programa Java a partir de uma página da Web e executá-lo localmente no sistema de um usuário.

Esses programas, que são chamados de *applets,* aparecem em uma página da Web de maneira semelhante às imagens. Ao contrário das imagens, os applets podem ser interativos — pegando a entrada do usuário, respondendo a ela e apresentando conteúdo mutante.

Os applets podem ser usados para criar animação, figuras, jogos, formulários que respondem imediatamente à entrada do leitor ou outros efeitos interativos nas mesmas páginas da Web, entre o texto e as imagens. A Figura 1.1 mostra um applet em execução no Netscape Navigator 4.04. A linguagem Java permite que uma pessoa jogue contra três oponentes do computador em um jogo de dominó.

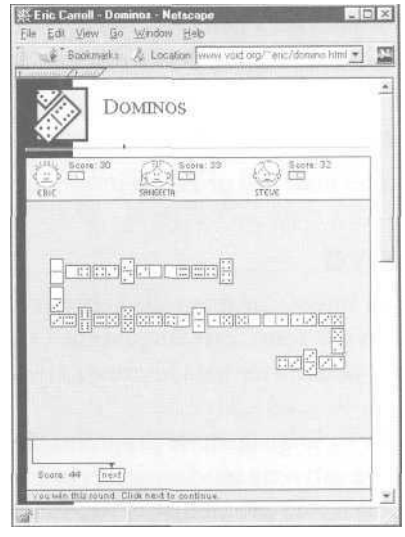


Figura 1.1
Um applet Java em execução no Netscape *Navigator 4.04.*

Nota

O applet Dominós foi escrito por Eric Carrol, um programador em Java, cartunista e oponente do computador que aparece como uma carinha na tela da Figura 1.1. Você pode encontrar o Dominós e a home page de Eric visitando o endereço http://www.void.org/-eric/domino.html.

Os applets são carregados por download na World Wide Web, exatamente como as páginas HTML, imagens gráficas ou qualquer outro elemento de site da Web. Em um navegador que esteja equipado para usar o Java, o applet começará a ser executado quando terminar o download.

Os applets são escritos com a linguagem Java, compilados de forma a poderem ser executados como um programa e colocados em um servidor da Web. A maioria dos servidores pode distribuir arquivos Java sem qualquer alteração em sua configuração.

Nota

Atualmente, quase todos os applets são escritos usando-se Java 1.0.2, a primeira versão popular da linguagem, pois as empresas líderes na fabricação de navegadores demoraram para oferecer suporte aos applets Java 1.1 e Java 2. No momento da redação deste livro, a Netscape havia lançado um patch de software no Navigator 4.04, que incluía suporte parcial à versão 1.1, e a Microsoft não anunciou planos para oferecer suporte integral a essa versão no Internet Explorer 4.

Os usuários da Web que possuam um navegador que não ofereça suporte a Java podem ver texto, uma imagem gráfica ou nada — isso depende de o projetista da página oferecer uma alternativa ao applet Java. Você aprenderá mais sobre como os applets, navegadores e a World Wide Web trabalham em conjunto por todo este livro.

Embora os applets provavelmente sejam o uso mais popular de Java, eles representam apenas uma maneira pela qual a linguagem pode ser usada. Assim como o Visual C + + , Visual Basic e Delphi, a linguagem Java é robusta, podendo ser usada para o desenvolvimento de uma ampla variedade de software, oferecendo suporte a interfaces gráficas com o usuário, interligação em rede, conectividade de banco de dados e outras funcionalidades sofisticadas. Para diferenciar dos applets, os programas em Java que não são executados dentro de um navegador da Web são denominados *aplicativos*.

A biografia não-autorizado

A linguagem Java foi desenvolvida na Sun Microsystems, em 1991, como parte do projeto Green, um grupo de pesquisas que estava trabalhando para desenvolver software a fim de controlar dispositivos eletrônicos para o consumidor. Os pesquisadores esperavam desenvolver a linguagem de programação que pudesse fazer funcionar os utensílios domésticos inteligentes do futuro — TVs interativas, torradeiras interativas, iluminação de caminho interativa (percebe um tema aqui?). Os pesquisadores da Sun também queriam que esses dispositivos se comunicassem entre si, de modo que o cortador de grama pudesse dizer ao liqüidificador para informar-lhe de que as vizinhas voltaram da escola e estavam tomando banho de sol de novo.

Para colocar sua pesquisa em ação, os pesquisadores da Green desenvolveram um protótipo de dispositivo chamado Star7, um aparelho parecido com um controle remoto que podia se comunicar com outros de mesma espécie. A idéia original era desenvolver o sistema operacional Star7 em C+ +, a linguagem de programação orientada a objetos extremamente popular, desenvolvida por Bjarne Stroustrup. Entretanto, o membro do projeto Green, James Gosling, aborreceu-se com a maneira como a linguagem C + + estava desempenhando a tarefa. Assim, ele se trancou em seu escritório e escreveu uma nova linguagem para manipular melhor o Star7. A linguagem foi denominada Oak, em homenagem a uma árvore que Gosling podia ver pela janela de seu escritório.

Posteriormente, a Sun descobriu que o nome Oak já estava sendo usado, mas não utilizou o método de "olhar pela janela" de Gosling, quando atribuiu outro nome à linguagem. Se tivesse feito isso, você poderia estar acrescentando um dos seguintes nomes de linguagem em seu currículo de 21 dias:

- Arbustos
- PrédioDeEscritóriosDaPortaAoLado
- PosteDeLuz
- LavadorDeJanelas
- SecretáriaSaindoParaAlmoçar
- SegurançaEstranho
- CaraGordoCortandoGrama

Como foi projetada tendo-se em vista utensílios domésticos e não os mais modernos PCs, a linguagem Java tinha de ser pequena, eficiente e facilmente portável para uma ampla gama de dispositivos de hardware. Ela também tinha de ser confiável. As pessoas aprenderam a conviver com falhas ou defeitos de sistema ocasionais em um aplicativo de software que ocupava até 5 MB de espaço em disco. Não é tão fácil aceitar uma torradeira malprogramada, com um desagradável hábito de queimar o pão.

Embora a linguagem Java não tenha funcionado originalmente como uma ferramenta de desenvolvimento de utensílios domésticos e TV interativa, as coisas que a tornaram boa para o Star7 se revelaram boas para a World Wide Web:

- A linguagem Java é pequena — faz os programas de uma página serem carregados por download mais rapidamente.
- A linguagem Java é segura — impede que hackers escrevam programas que danifiquem os sistemas dos usuários de navegador.
- A linguagem Java é portável — permite execução no Windows, Macintosh e outras plataformas sem modificação.

Além disso, a linguagem Java poderia ser usada como uma linguagem de programação de uso geral, para o desenvolvimento de software que pode ser executado em diferentes plataformas.

Para demonstrar o potencial da linguagem Java e evitar que seu projeto de pesquisa fosse engavetado, foi criado, em 1994, um navegador da Web que podia executar applets Java. O navegador demonstrou duas coisas a respeito da linguagem Java: o que ela oferecia para a World Wide Web e o tipo de programa que poderia criar. Os programadores Patrick Naughton e Jonathan Payne usaram Java para criar o navegador, originalmente denominado WebRunner, mas rebatizado como HotJava.

Embora a linguagem Java e o navegador Hotjava tenham recebido muita atenção na comunidade da Web, a linguagem realmente decolou depois que a Netscape se tornou a primeira empresa a licenciá-la, em agosto de 1995. O executivo e jovem milionário da Netscape Marc Andreesen foi uma das primeiras pessoas de fora da Sun a perceber o apelo da linguagem Java e deu-lhe o maior apoio na conferência JavaOne, em maio de 1996. "A linguagem Java representa uma tremenda oportunidade para todos nós", disse ele aos participantes. Logo após o primeiro lançamento público da linguagem, a Sun ampliou os esforços para o desenvolvimento da Java com uma nova subsidiária, chamada JavaSoft, e contratou centenas de funcionários para continuar a expandir a linguagem.

Versões da linguagem

A Sun lançou três versões mais importantes da linguagem Java:

- Java 1.0.2 — ainda a mais amplamente suportada pelos navegadores da Web.

- Java 1.1.5 — um lançamento realizado no primeiro semestre de 1997, com aprimoramentos na interface com o usuário, tratamento de eventos e mais padronização através da linguagem.
- Java 2 — a nova versão, lançada em dezembro de 1998.

Os números de versão da linguagem Java normalmente correspondem ao principal desenvolvimento de software Java da Sun — o Java Development Kit. Comumente referido como JDK, o kit está disponível nas versões 1.0.2, 1.1.5 e 1.2. Quando este livro estava no prelo, o JDK 1.2 ainda era a versão usada no que a Sun chamava de plataforma Java 2.

O JDK sempre esteve disponível gratuitamente no site da Web da Sun, no endereço http://java.sun.com e essa disponibilidade é um dos fatores que ampararam o rápido crescimento da linguagem. Essa é a primeira ferramenta de desenvolvimento a oferecer suporte às novas versões de Java, quando elas são lançadas, freqüentemente seis meses ou mais adiante de qualquer outro software de desenvolvimento para Java.

Além do JDK, existem muitas outras ferramentas de desenvolvimento disponíveis comercialmente para programadores de Java. Isso inclui as seguintes:

- Symantec Visual Café
- Borland JBuilder
- SuperCede
- Rogue Wave JFactory
- Natural Intelligence Roaster
- MetroWerks CodeWarrior
- SunSoft Java WorkShop

Se você vai usar algo diferente do JDK 1.2 da Sun para criar programas em Java, então precisa certificar-se de ter suporte para a versão 2 da linguagem. Quando este livro estava no prelo, o JDK ainda era a única ferramenta que tinha suporte integral para a versão 2 da linguagem.

Os programas deste livro foram testados com o JDK 1.2, a versão mais atualizada do Development Kit quando este trabalho ainda estava no prelo. Se você estiver usando algo diferente do JDK quando trabalhar no livro, primeiramente deve certificar-se de possuir suporte integral ao JDK 1.2/Java 2.

Revele-me, bola de cristal!

Qualquer um que possa adivinhar o futuro da linguagem Java deve se aventurar no mercado de capitais, em vez de escrever um livro. A empresa de tecnologia

KPCB (Kleiner, Perkins, Caufield and Byers) ofereceu 100 milhões de dólares para apoiar a abertura de empresas que realizassem trabalhos relacionados à linguagem Java.

Entretanto, os aprimoramentos incluídos na versão 2 da Java indicam seu futuro como uma sofisticada linguagem de propósito geral. As primeiras versões da linguagem Java eram mais adequadas a pequenos programas de software baseados na Web, em vez de a aplicativos completos, como software de colaboração em grupo, conjuntos de produtividade e jogos para vários participantes interligados em rede.

Isso não pode mais ser dito da linguagem Java versão 2. Alguns dos recursos avançados introduzidos na versão atual incluem os seguintes:

- Swing — Novos recursos para a criação de uma interface gráfica com o usuário, no estilo de um sistema operacional específico ou em novos "aparência e comportamento", denominado Metal.
- Arrastar e soltar — A capacidade de transferir informações interativamente entre diferentes aplicativos e de uma parte para outra da interface de um programa.
- Completa revisão dos recursos de áudio da linguagem Java, tornando-os mais de acordo com os recursos sonoros de outras linguagens.

Você terá a chance de trabalhar com esses e outros novos recursos nas próximas três semanas.

Por que você está aqui?

Era muito fácil descobrir por que as pessoas estavam escolhendo um livro deste tipo. A maioria dos leitores queria usar Java para criar applets.

Hoje em dia, isso não é tão claro. Cada nova versão de Java introduz recursos que a estendem para além de suas raízes como uma interessante tecnologia da Web.

Entretanto, as vantagens da linguagem Java continuam sendo sua natureza independente de plataforma e orientada a objetos, assim como sua facilidade de aprendizado.

A linauaaem Java é independente de plataforma

A *independência de plataforma* — a capacidade de o mesmo programa ser executado em diferentes plataformas e sistemas operacionais — é uma das vantagens mais significativas que a linguagem Java possui sobre as outras linguagens de programação.

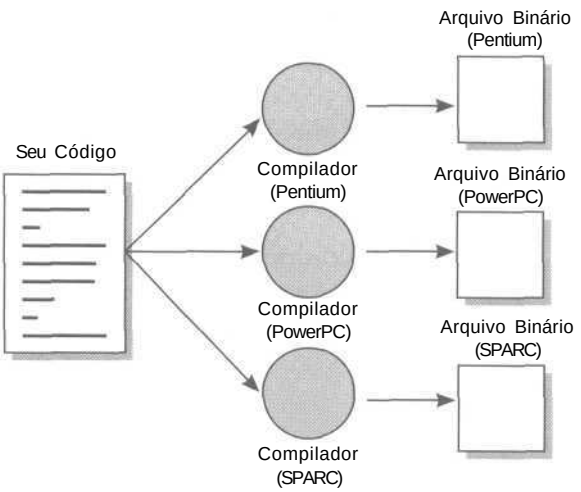
Quando você compila um programa escrito em C ou na maioria das outras linguagens, o compilador transforma seu arquivo-fonte em *código de máquina* — instruções específicas para o processador que seu computador está executando. Se você compilar seu código em um sistema baseado em Intel, o

programa resultante será executado em outros sistemas baseados em Intel, mas não funcionará em computadores Macintosh, Commodore VIC-20 ou outras máquinas. Se você quiser usar o mesmo programa em outras plataformas, deve transferir seu código-fonte para a nova plataforma e recompilá-lo para produzir o código de máquina específico para esse sistema. Em muitos casos, serão exigidas alterações no código-fonte antes que ele seja compilado na nova máquina, devido a diferenças em seus processadores e outros fatores.

A Figura 1.2 mostra o resultado de um sistema dependente de plataforma: vários programas executáveis devem ser produzidos para os vários sistemas.

Figura 1.2

Programas *tradicionais compilados*



Os programas Java atingem essa independência através da utilização de uma máquina virtual — uma espécie de computador dentro de outro. A *máquina virtual* pega os programas Java compilados e converte suas instruções em comandos que um sistema operacional possa manipular. O mesmo programa compilado, que existe em um formato chamado bytecode, pode ser executado em qualquer plataforma e sistema operacional que possua uma máquina virtual Java.

NOVO TERMO

Bytecode é a versão da máquina virtual Java do código de máquina, as instruções que ela entende diretamente.

A máquina virtual também é conhecida como *interpretador Java* ou *runtime Java.*

Se você estiver com dificuldade de entender o papel da máquina virtual, imagine o dispositivo que, na série *Jornada nas Estrelas* original, converte o idioma inglês (português, para nós) em qualquer idioma que os alienígenas estejam falando para a tripulação da nave espacial *Enterprise.* O capitão James T. Kirk não precisa aprender um novo idioma cada vez que desembarca em um planeta, pois o tradutor universal transforma suas palavras em algo que os

alienígenas compreendem. Da mesma forma, os programadores de Java não precisam criar diferentes versões de seus programas para cada plataforma em que desembarcam, pois a máquina virtual realiza a tradução necessária. (É claro que Kirk usava o tradutor para conhecer mulheres de outro mundo. Nós não garantimos, expressa ou implicitamente, que a linguagem Java arrume namoradas para você.)

A linguagem Java também é independente de plataforma no código-fonte. Os programas Java são salvos como arquivos de texto antes de serem compilados, os quais podem ser criados em qualquer plataforma que ofereça suporte à Java. Por exemplo, você poderia escrever um programa Java em um Macintosh e compilá-lo em bytecode em uma máquina Windows 95.

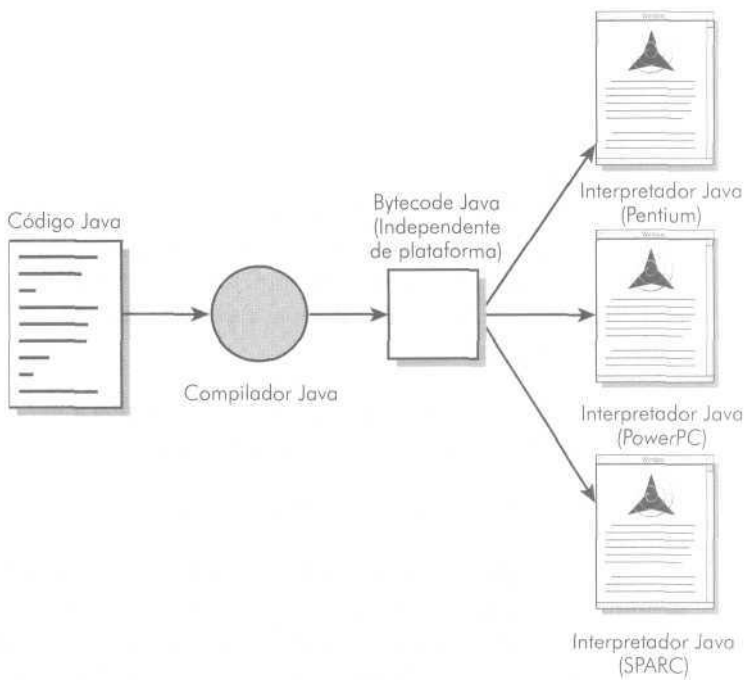
NOVO TERMO

Código-fonte é o conjunto de instruções de programação que um programador introduz em um editor de textos ao criar um programa. O código-fonte é compilado em bytecode para que possa ser executado por uma máquina virtual Java.

O bytecode é semelhante ao código de máquina produzido por outras linguagens, mas ele não é específico de nenhum processador. Ele acrescenta um nível entre o código-fonte e o código de máquina, conforme se vê na Figura 1.3.

Figura 1.3

Programas *multiplatoforma* da *linguagem Java.*



A máquina virtual Java pode ser encontrada em vários lugares. Para applets, a máquina virtual é incorporada em um navegador compatível com Java ou instalada separadamente para uso do navegador. Os programadores de applet não precisam se preocupar se ela existe no sistema de um usuário.

Entretanto, deve ser tomado um certo cuidado. Você precisa se preocupar com a máquina virtual Java suportada pelo navegador. Se você criar um applet que usa novos recursos da linguagem Java 2, seu bytecode não funcionará nos navegadores que contêm apenas a máquina virtual Java 1.0. O Java Plug-in, um aprimoramento de navegador que está sendo desenvolvido pela Sun, permite que os projetistas especifiquem uma máquina virtual diferente daquela incluída com o Netscape Navigatorou com o Microsoft Internet Explorer. Isso permite que os applets Java 1.1 e 2 funcionem, desde que sua correspondente máquina virtual seja especificada. Os detalhes sobre o Java Plug-in estão disponíveis na seguinte página da Web:

http://java.sun.com/products/plugin/

Por outro lado, os aplicativos Java só podem ser executados em um sistema onde a correspondente máquina virtual Java tenha sido instalada. Se você desejar executar aplicativos Java 2 em seu sistema de área de trabalho, primeiramente deverá instalar a máquina virtual.

Se estiver acostumado com o modo como as linguagens como Visual Basic e Delphi criam código específico de plataforma, você poderá pensar que o interpretador de bytecode acrescenta uma camada desnecessária entre seu código-fonte e o código de máquina compilado.

Isso pode causar alguns problemas de desempenho — os programas Java executam mais lentamente do que as linguagens compiladas dependentes de plataforma, como C, e a diferença de velocidade é a principal deficiência do Java. Algumas de suas ferramentas de desenvolvimento incluem compiladores "just-in-time" que podem executar o bytecode Java com maior velocidade.

A capacidade de um único arquivo de bytecode ser executado em várias plataformas é fundamental para o que faz a linguagem Java funcionar na World Wide Web, pois a própria Web é independente de plataforma. Assim como os arquivos HTML podem ser lidos em qualquer plataforma, os applets podem ser executados em qualquer plataforma com um navegador compatível com Java.

Para muitos programas básicos em Java, a velocidade pode não ser problema. Se você escrever programas que exijam maior velocidade de execução do que a que a máquina virtual pode fornecer, existem soluções disponíveis:

- Usar chamadas para código de máquina específico do sistema em seu programa Java, o que o torna dependente de plataforma.

- Usar compiladores just-in-time, que convertem os bytecodes Java em código específico do sistema.

Usando uma dessas soluções, você ganhará velocidade em detrimento da portabilidade da linguagem Java. Um aplicativo Java que use chamadas do Windows para seu acesso a disco não funcionaria em um computador Macintosh sem modificação.

A linguagem Java é orientada a objetos

Se você ainda não está familiarizado com a programação orientada a objetos, terá muitas chances de se tornar durante os próximos seis dias.

A *programação orientada a objetos* — também chamada *OOP* — é um modo de conceitualizar um programa de computador como um conjunto de objetos que se interagem. Para alguns, ela é apenas um modo de organizar programas e qualquer linguagem pode ser usada para criar programas orientados a objetos.

Entretanto, você obtém as maiores vantagens da programação orientada a objetos quando utiliza uma linguagem criada para ela. Java herda grande parte de seus conceitos de OOP do C + +, a linguagem em que está fortemente baseada. A linguagem Java também empresta conceitos de outras linguagens orientadas a objetos.

Você aprenderá mais sobre a programação orientada a objetos e Java no Dia 2.

A linguagem Java é fácil de aprender

Além de sua portabilidade e orientação a objetos, a linguagem Java é menor e mais simples do que as linguagens comparáveis. Isso deriva do objetivo original da linguagem Java ser algo que exija menos poder de computação para ser executado — ninguém vai gastar 3.000 dólares em um computador Pentium II com tecnologia MMX.

A linguagem Java pretendia ser mais fácil de escrever, compilar, depurar e aprender. A linguagem foi modelada fortemente de acordo com o C + + e grande parte da sintaxe e da estrutura orientada a objetos é proveniente diretamente dessa linguagem. Se você é programador de C + +, poderá aprender Java muito mais rapidamente e provavelmente examinar de forma superficial parte do material da primeira semana deste livro.

A despeito das semelhanças da linguagem Java com o C + +, os aspectos mais complexos e propensos a erros desta foram excluídos. Você não vai encontrar ponteiros ou aritmética de ponteiro na linguagem Java, pois esses recursos são facilmente confundidos em um programa e até mais difíceis de corrigir. Stnngs e arrays são objetos em Java, e o gerenciamento de memória é feito automaticamente, em vez de exigir que o programador controle isso. Os programadores experientes poderão sentir falta desses recursos, quando

começarem a usar Java, mas os outros aprenderão Java mais rapidamente devido à sua ausência.

Embora Java seja mais fácil de aprender do que muitas outras linguagens de programação, uma pessoa sem nenhuma experiência em programação terá dificuldades com a linguagem. Ela é mais complicada do que trabalhar em algo como HTML ou JavaScript, mas definitivamente é algo que um iniciante pode executar.

Meraulhando na programação em Java

Todas as lições de história serão deixadas para a segunda parte do trabalho de hoje. Você terá a chance de ver a linguagem Java em ação, criando seu primeiro programa em Java — um aplicativo.

Antes de começar, você precisará instalar uma ferramenta de desenvolvimento Java 2 em seu sistema.

Selecionando uma ferramenta de desenvolvimento Java

Escrever programas em Java requer algum tipo de software de programação Java. Se seu sistema já pode executar applets quando você surfa na Web, você pode achar que ele já está configurado para desenvolvimento. Não é verdade — você precisa de uma ferramenta de programação para criar e executar seus próprios programas em Java.

Para usar integralmente este livro, você precisar utilizar uma ferramenta de desenvolvimento que ofereça suporte total a Java 2. No momento da redação deste livro, a única escolha era o JDK. O JDK é sempre a primeira ferramenta a oferecer suporte a uma nova versão de Java.

O JDK é um conjunto de programas de linha de comando, utilitários baseados em textos que não fazem uso de uma interface gráfica com o usuário. Os programadores executam cada um dos utilitários do JDK digitando comandos em um prompt, como o seguinte:

```
java GetFunky.class
```

Esse comando diz ao programa java — o interpretador de bytecode — para que execute um arquivo de bytecode chamado GetFunky.class. (Conforme você verá posteriormente, todos os programas Java compilados possuem a extensão de arquivo .class.)

Os usuários de Windows 95 devem utilizar o comando Prompt do MS-DOS (Start | Programs | MS-DOS Prompt, a partir da barra de tarefas) para abrir uma janela em que os comandos podem ser digitados.

Isso está longe da maioria das modernas ferramentas de programação, que fazem uso de interfaces gráficas com o usuário, depuradores, editores de programação e muitos outros requintes. As pessoas que se sentem à vontade com o MS-DOS e com sistemas operacionais do tipo DOS não terão problemas para usar o JDK.

Se você possui outra ferramenta de desenvolvimento Java e tem certeza de que ela oferece suporte a Java 2, então, poderá usá-la para criar todos os programas deste livro. Caso contrário, o JDK 1.2 deve ser utilizado.

Instalando o Java Development Kit

A versão 1.2 do Java Development Kit está atualmente disponível para as seguintes plataformas:

- Windows 95
- Windows NT
- Solaris SPARC
- Solaris x86

Você pode carregar por download o JDK para essas plataformas no site na World Wide Web da Sun, que se encontra no seguinte endereço:

```
http://java.sun.com
```

A seção Products & APIs do site da Web oferece vínculos para as diferentes versões do JDK e outros produtos da Sun.

Para as versões oficiais do JDK para Windows e Solaris da Sun, o endereço direto atual é a seguinte página da Web:

```
http://www.javasoft.com/products/JDK/1.2/index.html
```

Essa página contém instruções sobre a instalação e um vínculo para se carregar por download o JDK para sua plataforma. Use esse vínculo para carregar o JDK por download e salvá-lo em uma pasta no seu sistema.

Escolha a versão mais atual do JDK, seja 1.2, 2.0 ou algo nessa linha. Periodicamente, a Sun disponibiliza versões para correção de erros que acrescentam um dígito extra ao número, como quando o Java 1.1 foi seguido por 1.1.1, 1.1.2 etc. Você deve carregar por download a versão mais recente do JDK disponível para sua plataforma.

Se você está usando outra plataforma, como o Apple Macintosh, então, pode verificar se ela possui um JDK compatível com a versão 1.2, visitando o site de Java oficial da Sun, no endereço:

```
http://java.sun.com
```

A página atual, que lista todas as versões conhecidas do JDK para diferentes plataformas, tem o seguinte endereço:

```
http://java.sun.com:80/cgi-bin/java-ports.cgi
```

Você deve certificar-se de que o arquivo inteiro foi recebido, após carregá-lo por download. As instruções de instalação da Sun, presentes na Web, listam o tamanho da versão corrente do JDK em sua plataforma.

Para verificar o tamanho no Windows 95, 98 ou no Windows NT, vá para a pasta que contém o arquivo repositório de instalação do JDK e dê um clique com o botão direito do mouse no arquivo. Um menu suspenso aparece e você pode selecionar o comando Properties para ver o tamanho do arquivo em bytes, junto com outras informações pertinentes.

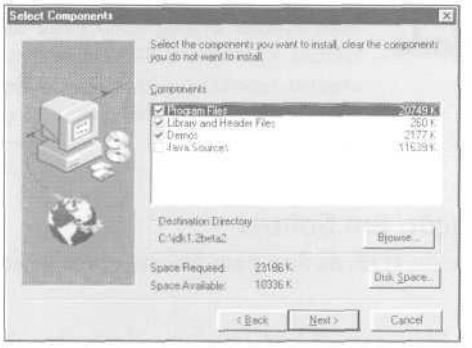
Instalação em Windows

Antes de instalar o JDK em seu sistema, você deve se certificar de que nenhuma outra ferramenta de desenvolvimento Java esteja instalada. Ter mais de uma ferramenta de programação Java em seu sistema provavelmente causará problemas de configuração, quando você tentar usar o JDK.

Para instalar o JDK no Windows, dê um clique duplo no arquivo relativo ao arquivo repositório de instalação ou use o comando Start | Run, na barra de tarefas do Windows, para encontrar e executar o arquivo.

Depois que você vir uma caixa de diálogo perguntando se deseja instalar o JDK, o JDK Setup Wizard será apresentado (ver Figura 1.4). Você pode usar essa janela para configurar o modo como o JDK é instalado em seu sistema.

Figura 1.4
D JDK Sefup *Wizard.*



As configurações padrão desse assistente devem ser satisfatórias para a maioria dos usuários. O JDK é instalado em uma nova pasta, com um nome baseado na versão que você está carregando por download (por exemplo, `\jdkl.2`), a não ser que você use o botão Browse para selecionar uma pasta diferente em seu sistema.

Dica

Todos os problemas de configuração que você possa ter com o JDK serão mais facilmente resolvidos se você se certificar de que ele esteja instalado na pasta `\jdkl.2`, que pode ser o padrão.

O assistente instalará três componentes do JDK:

- Arquivos de programa — Os programas executáveis necessários para criar, compilar e testar seus projetos em Java.
- Arquivos de biblioteca e cabeçalho — Arquivos usados apenas pelos programadores que estejam fazendo chamadas para código nativo a partir de seus programas Java. Você pode omitir isso nos exercícios dirigidos deste livro.
- Arquivos demo — Programas em Java, com versões que você pode executar e arquivos-f onte que pode examinar para aprender mais sobre a linguagem.

Se aceitar a instalação padrão, você precisará de 26 MB de espaço livre em disco rígido. Omitir tudo, menos os arquivos de programa, economiza 4 MB, no caso de você estar tentando espremer o JDK em uma unidade de disco abarrotada.

Você também pode optar por instalar arquivos-fonte — o código-fonte da linguagem Java em si está disponível publicamente. Entretanto, esses arquivos ocupam mais de 14 MB de espaço em disco e não são necessários para nenhum material deste livro.

Após instalar o JDK, você pode observar que um dos arquivos instalados na subpasta JDK\l ib se chama `classes.zip`. Embora seja um arquivo repositório Zip, ele não deverá ser descompactado. O JDK pode ler o arquivo `classes.zip` em seu formato de arquivamento nessa pasta.

Instalação em Solaris

A versão do JDK da Sun para Solaris pode ser instalada nas seguintes plataformas:

- Sistemas SPARC executando Solaris 2.4 ou posterior.
- Sistemas x86 executando Solaris 2.5 ou posterior.

O arquivo repositório de instalação do JDK deve ser descompactado em um diretório que ainda não possua um subdiretório chamado jdkl.2; se você fizer isso, poderá sobrescrever alguns arquivos já existentes em seu sistema.

Se você carregou por download o arquivo de instalação, então deve certificar-se de que pode acessá-lo corretamente usando o comando de shell chmod a+x no nome de arquivo.

Por exemplo, os usuários de SPARC usariam o seguinte comando:

```
% chmod a+x jdkl2-solaris2-sparc.bin
```

Para instalar o JDK após fazer a alteração chmod, use uma janela de shell para introduzir o comando ./, seguido do nome de arquivo do repositório. A seguir há um exemplo:

```
% ./jdkl2 -solaris2-sparc.bin
```

Testando a instalação

Em uma situação ideal, o JDK deve funcionar corretamente após a instalação. Além disso, o queijo deve ser light, os presidentes devem ser virtuosos e Jimmy Johnson ainda deve ser o técnico dos Dallas Cowboys.

Os problemas mais comuns encontrados ao se aprender a linguagem Java resultam de erros na configuração do JDK.

Os usuários de Windows podem testar sua instalação de JDK utilizando o comando MS-DOS Prompt (Start | Programs | MS-DOS Prompt, na maioria dos sistemas). Isso faz aparecer uma janela em que você pode introduzir comandos em MS-DOS, o sistema operacional que precedeu o Windows.

O prompt do MS-DOS também é chamado de *prompt de comando*, pois você pode usá-lo para digitar comandos que o sistema operacional executa.

O MS-DOS pode ser complexo para as pessoas que estão acostumadas ao estilo gráfico do "apontar e dar um clique" do Windows 95. Entretanto, você não pode usar o JDK sem aprender um pouco de MS-DOS. Este livro oferece algumas dicas para permitir que você aprenda o mínimo humanamente possível do MS-DOS.

Digite o seguinte em um prompt de comando, para testar se seu sistema consegue encontrar a versão correta do JDK:

```
ENTRADA> java -version
```

Se você estiver usando o JDK 1.2, deverá ver a seguinte mensagem em resposta:

```
SAÍDA> Java version "1.2"
JAVA W
```

O número de versão apresentado é específico; portanto, se você tiver carregado por download uma versão subseqüente com correção de erros, com o número 1.2.1, por exemplo, isso deve ser indicado pelo comando java -version. Se você vir o número de versão errado ou o erro "Bad command or file name", seu sistema não conseguiu encontrar a versão correta de java.exe, o arquivo que executa os programas Java. Isso deve ser corrigido antes que você possa começar a escrever programas Java. Consulte o Apêndice D para obter mais informações.

Seu primeiro aplicativo Java

Agora, vamos trabalhar realmente. Comece criando um aplicativo Java simples: um programa que apresenta uma das coisas mais estranhas jamais pronunciada aos gritos para uma celebridade, por um membro do público — "What's the frequency, Kenneth?"

Em 4 de outubro de 1986, um homem gritou "What´s the frequency, Kenneth?" para o âncora da televisão Dan Rather, pouco antes de golpeá-lo em uma rua de Nova York. Por vários anos, ninguém entendeu o motivo do ataque, e o grupo R.E.M. imortalizou o bordão em uma música. Posteriormente, o homem foi preso, após atirar em um técnico da NBC em 1 994 e disse a um psiquiatra que as redes de televisão não conseguiriam apanhá-lo.

Os aplicativos Java são programas independentes que não exigem um navegador da Web para serem executados. Eles são parecidos com os programas que você utiliza mais freqüentemente em seu sistema—você os executa localmente com seu mouse ou digitando o nome do programa em uma linha de comando.

Embora um programa Java possa ser criado como um applet e como um aplicativo, quase todos os programas que você encontrar serão de um tipo ou de outro.

Nesta primeira semana, você escreverá a maior parte dos aplicativos enquanto aprenderá a linguagem Java. Esse conhecimento será estendido à programação de applets na Semana 2. Se você é um desses leitores que estão interessados principalmente no desenvolvimento de applets, não pule direto para a segunda semana. Enquanto estiver criando aplicativos Java simples, tudo o que você aprender se aplicará à criação de applets e começar com o básico é melhor. Você criará muitos applets do Dia 8 ao Dia 14.

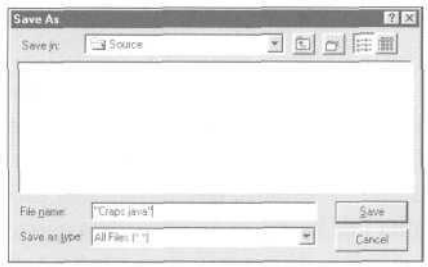
Criando o arquivo-fonte

Assim como acontece com a maioria das linguagens de programação, seus arquivos-fonte Java são salvos como arquivos de texto puro. Você pode criá-los com qualquer editor ou processador de textos que salve texto puro, um formato que também é chamado de texto ASCII ou texto DOS. Os usuários de Windows 95 podem escrever programas Java com o Bloco de Notas, DOS Edit e Write, assim como o Microsoft Word, se você tiver o cuidado de salvar o arquivo como texto, em vez de gravá-lo no formato proprietário do Word. Os usuários de UNIX podem escrever programas com emacs, pico e vi, e os usuários de Macintosh possuem o SimpleText para a criação de arquivo-fonte Java.

O Java Development Kit não inclui um editor de textos, mas quase todas as outras ferramentas de desenvolvimento Java possuem seu próprio editor para a criação de arquivos de código-fonte.

Se você estiver usando Windows 95, 98 ou Windows NT, um editor de textos como o Bloco de Notas pode inserir uma extensão .txt extra no nome de todos os arquivos-fonte Java que forem gravados (o que transforma um nome como GetFunky.java em GetFunky.java.txt. O compilador Java manipula apenas os arquivos-fonte com a extensão .java. Para evitar esse problema, coloque o nome do arquivo entre aspas, quando estiver salvando um arquivo-fonte. A Figura 1.5 mostra essa técnica sendo usada para salvar o arquivo-fonte Craps.java a partir do Bloco de Notas do Windows.

Figura 1.5
Salvando um *arquivo-fonte.*



Dica

Uma solução melhor é usar o Windows Explorer para associar permanentemente os arquivos .java ao editor que você irá usar. Isso permite abrir um arquivo-fonte fazendo-se a edição, dando-se um clique duplo no arquivo em uma pasta do Windows. Para aprender a configurar isso, consulte o Apêndice E.

Escrevendo o programa Ative seu editor predileto e digite o programa Java apresentado na listagem 1.1. Certifique-se de que todos os parênteses, chaves e aspas da listagem sejam introduzidos corretamente e use as letras maiúsculas e minúsculas lá apresentadas. Se seu editor exige um nome de arquivo antes que você comece a introduzir qualquer coisa, use HelloDan.java.

DIGITE

Listagem 1.1 O código-fonte do aplicativo HelloDan.

```
1: class HelloDan {
2:   public static void main (String[ ] arguments) {
3:       System.out.println("What's the frequency, Kenneth?");
4:   }
5: }
```

Os números de linha e os dois-pontos presentes no lado esquerdo da listagem 1.1 não fazem parte do programa — eles foram incluídos para que você possa fazer referência a linhas específicas pelo número em um programa. Se você estiver em dúvida a respeito do código-fonte de um programa deste livro, poderá compará-lo com uma cópia dele, que se encontra no site da World Wide Web oficial do livro, no seguinte endereço:

http://www/prefect.com/j ava21

Depois de terminar de digitar o programa, salve o arquivo em algum lugar de sua unidade de disco, com o nome **HelloDan.java.**

Se você é usuário do Windows JDK, que está tentando aprender o mínimo possível do MS-DOS, abra a pasta-raiz de seu sistema e crie uma nova subpasta chamada j21 work. Salve o arquivo Hel loDan.java e todos os outros arquivos-fonte Java deste livro nessa pasta. Em breve você verá por quê.

Os arquivos-fonte Java devem ser salvos com a extensão .java. Os arquivos-fonte Java são compilados em bytecode com a extensão .class. Na linguagem Java, o termo *classe* muitas vezes é sinônimo *de programa.* (Você aprenderá mais a respeito das classes nos próximos três dias.)

Alinha 1 da listagem 1.1 identifica que o programa Java é a classe Hel loDan; portanto, o nome de arquivo deve ser Hel loDan.java. Se você der outro nome ao arquivo-fonte (mesmo que seja algo como hel lodan. javaou Hel lodan. java), não poderá compilá-lo.

Compilando e executando o programa no Windows

Agora, você está pronto para compilar o arquivo. Se estiver usando uma outra ferramenta de desenvolvimento que não seja o JDK, você deve consultar a documentação desse software para ver os detalhes sobre como compilar programas Java. Provavelmente, essa é uma operação razoavelmente simples, como dar um clique em um botão ou apresentar um comando de menu.

Com o JDK, você precisa usar a ferramenta de linha de comando javac, o compilador Java. O compilador lê um arquivo-fonte .java e cria um ou mais arquivos . cl ass que podem ser executados pela máquina virtual Java.

Os usuários de Windows devem carregar uma janela de prompt do MS-DOS, usando Start | Programs | MS-DOS Prompt e mudar as pastas para a que contém HelloDan. java.

Se você salvou o arquivo em uma pasta J21work recentemente criada, fora da pasta-raiz de seu sistema, o comando MS-DOS seria o seguinte:

```
cd \J21work
```

cd é a abreviação de "change directory" (mudar diretório); os termos "pasta" e "diretório" são sinônimos.

Quando você estiver na pasta correta, pode compilar Hel loDan.java introduzindo o seguinte em um prompt de comando:

ENTRADA

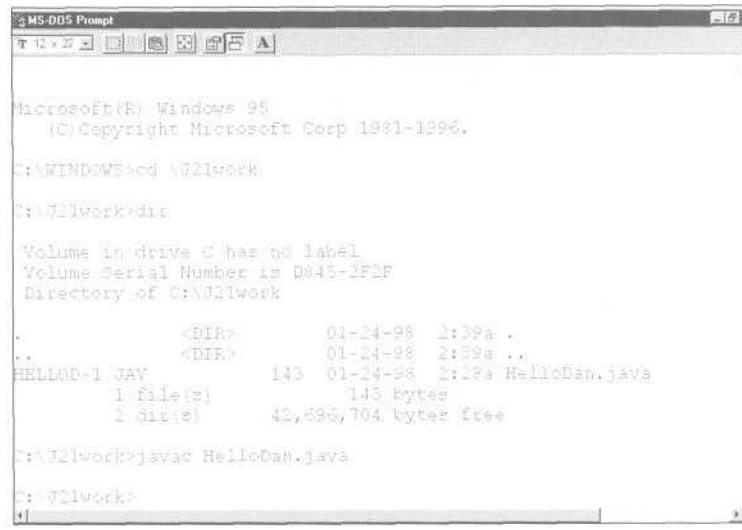
```
javac HelloDan.java
```

Nota

Se você usaro comando di r para listartodos os arquivos de uma pasta no MS-DOS, poderá observar que um arquivo possui dois nomes — aquele que você deu, como Hel loDan. java, e uma versão abreviada, como HELLOD-1. JAV. A forma abreviada é uma ramificação de como o Windows gerencia nomes de arquivo com mais de oito caracteres para o nome e três para a extensão. Quando você usar os utilitários do JDK em um prompt de comando, utilize sempre o nome que deu ao arquivo e não a abreviação.

A Figura 1.6 mostra os comandos do MS-DOS utilizados para trocar para a pasta \J21work, listar os arquivos da pasta e compilar Hel loDan. java.

Figura 1.6
Compilando programas Java em uma janela do MS-DOS.



O compilador do JDK não apresentará nenhuma mensagem, se você tiver sucesso ao compilar o programa. Se o programa foi compilado sem nenhum erro, será encontrado, na mesma pasta, um arquivo chamado Hel loDan.class, contendo Hel loDan. java. Esse arquivo class é o bytecode Java que pode ser executado pela máquina virtual. Se você obtiver erros, volte ao seu arquivo-fonte original e certifique-se de tê-lo digitado exatamente como aparece na listagem 1.1.

Quando você tiver um arquivo de classe, poderá executá-lo usando o interpretador de bytecode. A versão do JDK do interpretador é chamada java e ela também é executada a partir do prompt do MS-DOS, no Windows. Execute o HelloDan, alternando para a pasta que contém Hel loDan.class e introduzindo o seguinte:

ENTRADA
```
java HelloDan
```

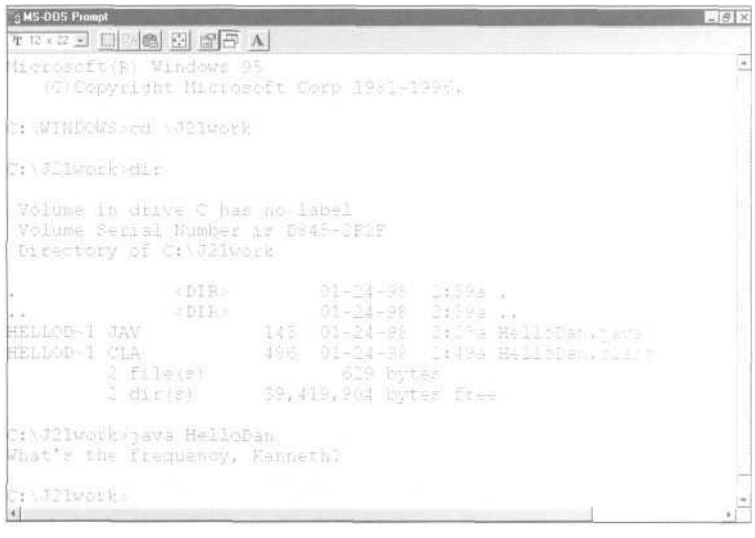
Se seu programa foi digitado e compilado corretamente, você deverá ver a frase What' s the frequency, Kenneth? exibida na tela, na janela Prompt do MS-DOS.

Nota

Certifique-se de retirar a extensão .class, quando executar um programa Java com a ferramenta java — digitar java Hel loDan.class resultará em um erro. Além disso, se você vir a mensagem de erro "Class **Not** Found" — mesmo estando na mesma pasta onde está **Hel l**oDan.class —, talvez seja preciso mudar outra configuração em seu arquivo autoexec.bat. Consulte o Apêndice D.

A Figura 1.7 mostra a saída do aplicativo HelloDan, junto com os comandos do MS-DOS utilizados para se chegar a esse ponto.

Figura 1.7 *Execução* de *aplicativos Java em uma janela do* MS-DOS.



Compilando e executando o programa no Solaris

Para compilar o arquivo-fonte Java em um sistema Solaris, use o compilador Java de linha de comando que acompanha o JDK. Em uma linha de comando do UNIX, use o comando cd a fim de mudar para o diretório que contém o arquivo-fonte Java Hel loDan. java. Se você utilizou o diretório J21work recomendado para usuários de Windows, o seguinte comando seria usado:

```
cd ~/J21work
```

Depois que você estiver no diretório correto, use o comando javac com o nome do arquivo, como segue:

```
javac HelloDan.java
```

Salvo quaisquer erros, você acabará com um arquivo chamado Hel l o-Dan. class. Esse é o arquivo de bytecode Java que pode ser executado pela máquina virtual. Se você obtiver qualquer erro, volte para o arquivo-fonte original e certifique-se de tê-lo digitado exatamente como aparece na listagem 1.1.

Quando você tiver um arquivo de classe, poderá executá-lo usando o interpretador de bytecode.

A versão do JDK do interpretador Java é chamada java e também é executada a partir da linha de comando. Execute o Hel loDan, trocando para o diretório que contém HelloDan.class e introduzindo o seguinte:

```
java HelloDan
```

Se seu programa foi digitado e compilado corretamente, você deverá ver a frase What´s the frequency, Kenneth? apresentada na tela.

Nota

Se você vir a mensagem de erro "Cl ass Not Found", mesmo estando na mesma pasta onde está Hel loDan. class, talvez seja preciso mudar a configuração de seu sistema para encontrar o JDK. Consulte o Apêndice D.

Resumo

Agora que você configurou uma ferramenta de desenvolvimento Java e utilizou-a para escrever seu primeiro programa Java, pode acrescentar o título "programador de Java" em seu currículo.

Isso, afinal, não é uma inverdade. Você não apenas criou um aplicativo Java que funciona, como também obteve informações sobre a história, as vantagens, desvantagens e o futuro da linguagem.

Java é uma linguagem de programação orientada a objetos, inspirada no C + +. Ela foi criada para ser mais simples, menos propensa a erros e mais fácil de aprender do que a linguagem C + +. Ela é independente de plataforma e pequena, duas características que a tornam ideal para ser executada em páginas da World Wide Web.

Os applets são programas Java que são executados na Web e os aplicativos são outro tipo de software que pode ser escrito com Java.

Ainda há muito pela frente, mas agora você já deve ter a base para criar aplicativos mais complexos e seus primeiros applets. Inclua uma linha em branco em seu currículo. Amanhã, você poderá escrever a lápis "programador orientado a objetos".

Perguntas e respostas

P **Qual é o relacionamento entre JavaScript e Java?**

R Suas primeiras quatro letras são iguais.

Uma interpretação errônea no mundo da Web é a de que Java e JavaScript têm mais em comum do que realmente acontece. Java é a linguagem de programação de propósito geral, a qual você aprenderá neste livro; você a utiliza para criar applets. JavaScript é uma linguagem de criação de scripts inventada pela Netscape, que se parece um pouco com Java; com ela, você pode fazer várias coisas interessantes em páginas da Web. Elas são linguagens independentes, usadas para objetivos diferentes. Se você estiver interessado em programação com JavaScript, vai querer outro livro, como o *Teach YourselfJavaScriptinaWeek* ou o *Web Workshop: Java Script* de *Laura Lemay*, ambos disponíveis pela Sams Publishing.

P **Onde posso aprender mais a respeito de Java e encontrar applets e aplicativos para trabalhar?**

R Você pode ler o restante deste livro! Aqui estão alguns outros lugares para procurar informações sobre Java e applets Java:

- A homepagedejava, no endereço http://www.java.sun.com/, é a fonte oficial de informações sobre Java, incluindo dados sobre o JDK, sobre o próximo lançamento da versão 1.2 e sobre ferramentas para o projetista, como o Java Workshop, assim como uma ampla documentação.
- Gamelan, no endereço http://www.gamelan.com/, é um repositório de applets e informações sobre Java, organizadas em categorias. Se você quiser mexer com applets ou aplicativos, deve olhar aqui.
- Para discussões sobre Java, verifique os newsgroups de comp.l ang. java, incluindo comp. 1 ang. java.programmer, comp. lang. java.tech. comp. 1 ang. java. advocacy, etc. (Você vai precisar de um leitor de notícias da Usenet para acessar esses newsgroups.)

SEMANA 1

Dia 2

Uma amostra da programação orientada a objetos

A programação orientada a objetos é como a cerveja.

A maioria das pessoas que toma um gole da bebida pela primeira vez, não gosta dela e pode questionar a sanidade mental daqueles que a elogiam. "O que foi que eu lhe fiz", perguntam eles, "para que você me desse isso para beber?"

Entretanto, com o passar do tempo, o gosto pela cerveja pode ser cultivado naqueles que continuarem a tomá-la. (Para muitas pessoas, esse período de tempo é chamado de escola.)

Assim como a cerveja, a programação orientada a objetos é um gosto adquirido. É uma das idéias de programação mais interessantes surgidas nos últimos anos e também é a fonte de muita consternação para os programadores que não estão familiarizados com ela.

De certa forma, a reputação é merecida. A programação orientada a objetos, também chamada de *OOP*, é um assunto que pode ser estudado e praticado por anos. Entretanto, a idéia central é simples: organizar seus programas de forma que eles espelhem o modo como os objetos são organizados no mundo real.

Hoje, você terá o primeiro contato com a maneira como a linguagem Java incorpora os princípios da programação orientada a objetos. Os seguintes tópicos são abordados:

- A organização de programas em elementos chamados classes e como essas classes são utilizadas para se criar objetos
- A definição de uma classe através de dois aspectos de sua estrutura: como ela deve se comportar e quais são seus atributos
- A interconexão das classes de modo que uma herde a funcionalidade da outra
- O vínculo entre classes por meio de pacotes e interfaces

Se você já está familiarizado com a programação orientada a objetos, grande parte da lição de hoje será apenas uma revisão. Mesmo que examine o material introdutório superficialmente, você deve criar o exemplo de programa para obter alguma experiência no desenvolvimento de seu primeiro applet Java.

Pensando em termos de objetos

A programação orientada a objetos é, em sua raiz, um modo de conceitualizar um programa de computador. Você pode imaginar um programa como uma lista de instruções que dizem ao computador o que fazer, ou como um grupo de pequenos programas que respondem a eventos específicos, iniciados pela entrada do usuário.

O modo OOP de ver um programa é como um conjunto de objetos que trabalham juntos de maneiras predefinidas para realizar tarefas. Considere o exemplo dos blocos de construção do LEGO.

Os blocos do LEGO, para aqueles que não têm crianças ou um filho com que se manter ocupado, são pequenos blocos de plástico, vendidos em várias cores e tamanhos. Esses blocos possuem pequenos pinos arredondados de um lado e eles se encaixam perfeitamente em receptáculos de outros blocos. Essas combinações de blocos criam formas maiores, e existem muitas peças diferentes de LEGO, como rodas, mecanismos, dobradiças e roldanas, para se usar.

Usando os blocos de LEGO, você pode fazer todos os tipos de coisas: castelos, automóveis, trailers enormes, ganchos, roupas de esporte — praticamente tudo o que imaginar. Cada peça de LEGO é um objeto que se encaixa em outros objetos de maneiras específicas para criar um objeto maior.

Considere outro exemplo. Com um pouco de experiência e alguma ajuda, você pode ir até uma loja de computadores e montar um sistema de computador pessoal inteiro a partir de vários componentes: uma placa-mãe, chip de CPU, placa de vídeo, disco rígido, teclado etc. Quando terminar de montar as várias unidades independentes, você terá um sistema em que todas elas trabalham juntas para criar um sistema maior, o qual você poderá usar para resolver os problemas para os quais comprou o computador.

Internamente, cada um desses componentes poderia ser extremamente complicado e construído por diferentes empresas usando diferentes métodos de design. Entretanto, você não precisa saber como cada componente funciona,

o que cada chip da placa faz ou como uma letra "A" é enviada para seu computador quando você pressiona a respectiva tecla em seu teclado. Cada componente utilizado é uma unidade independente e, como montador do sistema global, você está interessado apenas em como as unidades interagem entre si:

- Essa placa de vídeo vai encaixar no slot da placa-mãe?
- Esse monitor funcionará com essa placa de vídeo?
- Cada componente informará os comandos corretos para os outros componentes com que ele interage, de modo que cada parte do computador seja entendida por todas as outras?

Quando você conhece as interações entre os componentes e consegue combinar essas interações, confeccionar o sistema global é fácil.

A programação orientada a objetos é muito parecida com as estruturas de construção dos blocos do LEGO ou com a montagem de um PC. Usando a OOP, seu programa global é constituído de diferentes componentes, chamados objetos.

NOVO TERMO Um *objeto é* um elemento independente de um programa de computador, que representa um grupo de recursos relacionados e é projetado para realizar tarefas especificas. Os objetos também são denominados *instâncias.*

Cada objeto tem um papel específico em um programa, e todos os objetos podem trabalhar com outros objetos de maneiras definidas.

Objetos e classes

A programação orientada a objetos é modelada segundo a observação de que, no mundo real, os objetos são constituídos de muitos tipos de objetos menores. Entretanto, a capacidade de combinar objetos é apenas um aspecto geral da programação orientada a objetos. Ela também inclui conceitos e recursos que tornam a criação e o uso de objetos mais fáceis e mais flexíveis. O mais importante desses recursos é a classe.

NOVO TERMO Uma *classe* é um modelo usado para criar vários objetos com características semelhantes.

As classes incorporam todos os recursos de um conjunto de objetos em particular. Quando você escreve um programa em uma linguagem orientada a objetos, não define objetos individuais. Em vez disso, você define classes de objetos.

Por exemplo, você poderia ter uma classe Tree que descrevesse as características de todas as árvores:

- Possui folhas e raízes.
- Cresce.
- Gera clorofila.

A classe Tree serve como um modelo abstrato para o conceito de árvore. Para ter um objeto que possa realmente ser tratado em um programa, você deve ter uma instância concreta da classe Tree.

As classes são usadas para se criar objetos e você trabalha com os objetos diretamente em um programa. Uma classe Tree pode ser usada para criar muitos objetos Tree diferentes e cada um deles poderia ter características diferentes:

- Baixa ou alta.
- Cerrada ou rala.
- Frutífera ou não.

Embora esses objetos sejam diferentes entre si, eles ainda têm o suficiente em comum para serem imediatamente reconhecidos como objetos relacionados. A Figura 2.1 mostra uma classe Tree e vários objetos criados a partir desse modelo.

Um exemplo de desian de classe

Em um exemplo mais próximo ao que poderia ser feito usando Java, você poderia criar uma classe para um botão de comando, um item para uso em janelas, caixas de diálogo e outros programas interativos. Quando a classe CommandButton fosse desenvolvida, ela poderia definir as seguintes características de um botão:

- O texto que identifica o objetivo do botão.
- O tamanho do botão.
- Aspectos de sua aparência, como se ele tem um sombreado tridimensional.

A classe CommandButton também poderia definir como um botão se comporta:

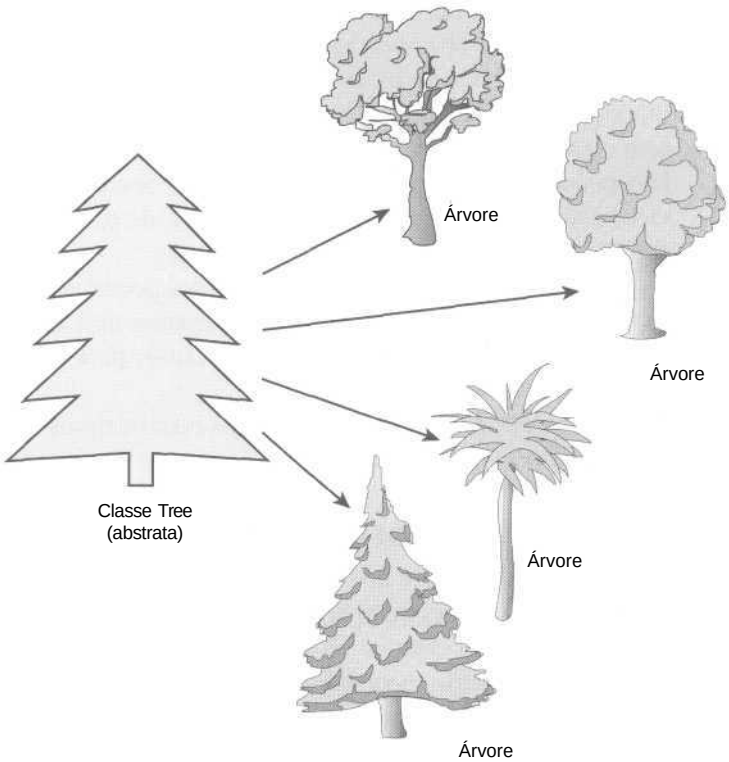
- Se o botão precisa de um clique de mouse apenas ou de um clique duplo para ser usado.
- Se ele deve ignorar totalmente os cliques de mouse.
- O que ele faz quando recebe um clique de mouse.

Uma vez definida a classe CommandButton, você pode criar instâncias desse botão — em outras palavras, objetos CommandButton. Todos os objetos assumem as características básicas de um botão de comando, conforme o que estiver definido pela classe, mas cada um poderia ter aparências diferentes e comportamento de acordo com o que o botão específico pode fazer. Criando uma

classe CommandButton, você não precisa ficar rescrevendo o código de cada botão de comando que queira usar em seus programas. Além disso, você pode reutilizar a classe CommandButton para criar diferentes tipos de botões, à medida que precisar deles, tanto nesse programa como em outros.

Figura 2.1

A classe Tree e vários objetos Tree.



Nota

Uma das classes padrão da linguagem Java, java.awt.Button, abrange toda a funcionalidade desse exemplo CommandButton hipotético e muito mais. Você terá uma chance de trabalhar com ela no Dia 1 1.

Quando escreve um programa em Java, você projeta e constrói um conjunto de classes. Quando o seu programa for executado, os objetos serão criados a partir dessas classes e usados quando necessário. Sua tarefa como programador Java é criar o conjunto de classes correto para realizar o que seu programa precisa fazer.

Felizmente, você não precisa começar do zero. Cada versão da linguagem Java inclui um grupo de classes que implementa a maioria da funcionalidade básica de que você precisará. Esses agrupamentos são chamados de bibliotecas.

NOVO TERMO Uma *biblioteca de classe é* um grupo de classes criado para ser usado com outros programas. A biblioteca de classe Java padrão contém dezenas de classes.

Quando você está falando sobre a utilização da linguagem Java, na verdade está falando sobre o uso da biblioteca de classe Java e de algumas palavras-chave e operadores que são reconhecidos por um compilador Java.

A biblioteca padrão da linguagem Java lida com inúmeras tarefas, como funções matemáticas, tratamento de texto, imagens gráficas, som, interação com o usuário e interligação em rede. Em muitos casos, as bibliotecas de classe Java serão suficientes para suas necessidades. Seu trabalho em tal circunstância seria criar uma classe simples que seja utilizada para criar objetos a partir das classes Java padrão e manipular sua interação.

Para programas Java complicados, você poderia criar um conjunto inteiro de novas classes, com interações definidas entre elas. Elas poderiam ser usadas para formar a sua própria biblioteca de classe, para uso posterior em outros programas.

A reutilização é uma das vantagens fundamentais da programação orientada a obietos.

Atributos e comportamento

Geralmente, toda classe que você escreve em Java é constituída de dois componentes: atributos e comportamento. Nesta seção, você aprenderá sobre cada componente, de acordo como ele se aplica à classe teórica Jabberwock. Para completar esta seção, você cria uma classe Java que implementa uma representação de um *jabberwock* — uma espécie de dragão, do poema *Jabberwocky*, de Lewis Carroll.

Atributos de uma classe de objetos

Atributos são as coisas específicas que diferenciam uma classe de objetos de outra e determinam a aparência, o estado e outras qualidades dessa classe. Imagine como uma classe teórica chamada Jabberwock poderia ser criada. Os atributos de um jabberwock poderiam incluir os seguintes:

- Cor orange (laranja), raw umber (castanho natural), lemon yellow (verde-limão), maize (amarelo).
- Sexo male (masculino), female (feminino), mind-your-own-business (não é da sua conta).
- Apetite **ful** 1 (satisfeito), hungry (com fome).

Os atributos de uma classe de objetos também podem incluir informações sobre o estado de um objeto. Por exemplo, você poderia ter as características da disposição do jabberwock (enfurecido ou calmo), o estado

de saúde (vivo ou morto) e tendências políticas (conservador, independente, café-com-leite)

Em uma classe, os atributos são definidos por variáveis. Você pode considerá-los análogos às variáveis globais de cada objeto dessa classe. Cada objeto possui diferentes valores para suas variáveis, e elas são chamadas de variáveis de instância.

NOVO TERMO Uma *variável de instância* é um item de informação que define um atributo de um objeto em particular. A classe do objeto define o tipo de atributo de que se trata e cada instância armazena seu próprio valor para esse atributo. As variáveis de instância também são chamadas de *variáveis de objeto*.

Cada atributo de classe possui uma única variável correspondente; você muda esse atributo em um objeto mudando o valor da variável. No programa que você criar posteriormente no dia de hoje, a seguinte instrução será usada para indicar que um objeto Jabberwock, representado por j, não está mais com fome (hungry):

```
j.hungry = false
```

As variáveis de instância podem receber um valor quando um objeto é criado e permanecerem constantes durante toda a existência do objeto, ou elas podem receber diferentes valores, à medida que o objeto for utilizado em um programa que esteja em execução.

Outro tipo de atributo é usado para descrever uma classe inteira de objetos, em vez de objetos específicos da classe. Eles são chamados de variáveis de classe.

NOVO TERMO Uma *variável de classe* é um item de informação que define um atributo de uma classe inteira. A variável se aplica à classe em si e a todas as suas instâncias; portanto, apenas um valor é armazenado, independentemente de quantos objetos da classe tenham sido criados.

Para um bom exemplo de variável de classe, considere uma variável que controle o número exato de objetos Jabberwock vivendo em uma comunidade. Se uma variável de instância fosse criada para a contagem da classe Jabberwock, cada um dos objetos poderia ter uma contagem diferente, o que não seria preciso. Uma variável de classe seria usada para que apenas um valor fosse armazenado e cada objeto Jabberwock teria acesso a essa variável.

Comportamento de uma classe de objetos

Comportamento é o modo pelo qual uma classe de objetos pode fazer algo para os próprios objetos ou para outros. O comportamento de uma classe determina o que os objetos dessa classe fazem para mudar seus atributos e também o que eles fazem quando outros objetos pedem para que façam algo. O comportamento de um objeto Jabberwock poderia incluir o seguinte:

- Ficar com fome.
- Ficar calmo.
- Comer um camponês.
- Pular o jantar.
- Restabelecer-se.

O comportamento de uma classe de objetos é feito usando-se métodos.

NOVO TERMO

Métodos são grupos de instruções relacionadas em uma classe de objetos, que atuam sobre eles mesmos e em outras classes e objetos. São usados para realizar taretas especificas, da mesma maneira que as funçoes são usadas em outras linguagens de programação.

Os objetos se comunicam entre si usando métodos. Uma classe ou um objeto pode chamar métodos em outra classe ou objeto por muitas razões, incluindo as seguintes:

- Para relatar uma alteração para outro objeto.
- Para dizer a outro objeto para que altere algo em si mesmo.
- Para dizer a outro objeto para que faça algo.

Por exemplo, considere o espadachim que existe no *poema Jabberwock*. Ele ataca o monstro com sua espada; aqui está, no trecho do poeta Lewis Carroll, o que aconteceu, passo a passo:

"One, two! One, two! And through and through

The vorpal blade went snicker-snack!

He left it dead, and with its head

He went galumphing back."

Em Java, o espadachim poderia ser criado como um objeto Knight (cavaleiro), usando a classe Kni ght como modelo de como seria o objeto Kni ght. Quando o espadachim corta a cabeça do monstro, isso definitivamente causa uma mudança no estado interno do jabberwock. Para denotar essa mudança, o objeto Knight usaria um método para dizer ao objeto Jabberwock: "Ei, eu cortei sua cabeça. Você está morto!".

Assim como existem variáveis de instância e de classe, também existem métodos de instância e classe. Os *métodos de instância*, que são tão comuns a ponto de serem chamados apenas de *métodos*, aplicam-se a um objeto da classe. Se o método faz uma mudança em um objeto específico, ele deve ser um método de instância. Os *métodos de classe* se aplicam a uma classe em si.

Criando uma classe

Agora que a terminologia básica da programação orientada a objetos foi apresentada, as coisas começam a se tornar mais claras com um exemplo mais concreto. Você criará um exemplo funcional da classe Jabberwock, para ver como as variáveis e métodos de instância são definidos em uma classe. Você também criará um applet Java que gera um novo objeto usando a classe Jabberwock, modificará suas variáveis de instância e realizará uma ação baseada em seus valores.

Nota

A sintaxe real desse exemplo não é abordada em muitos detalhes aqui. Utilize-a como uma introdução à programação orientada a objetos, em vez de usar como uma lição sobre a sintaxe de programação Java, o que será estudado mais a fundo no Dia 3.

Abra o editor de textos que você está usando para criar programas Java, para que possa começar a criar um arquivo-fonte Java. Em vez de digitar um programa inteiro, você introduz algumas instruções, enquanto aprende sobre sua utilização. Você tem a chance de conferir seu trabalho no final, para certificar-se de que ele está correto.

O lugar certo para começar é em uma definição de classe básica. Digite o seguinte:

```
class Jabberwock {
}
```

Você criou uma classe. No momento, ela não faz muita coisa, mas as duas linhas representam um exemplo de classe Java em seu estado mais simples.

Para tornar a classe Jabberwock mais sofisticada, crie três variáveis de instância para ela. Imediatamente abaixo da linha class Jabberwock {, insira as três linhas a seguir:

```
String color;
String sex;
boolean hungry;
```

Essas linhas criam três variáveis de instância. Duas delas, color e sex, podem conter objetos String. Um *string é* um termo genérico que significa um grupo de caracteres, mas um objeto String é criado em java usando-seumadas classes padrão da biblioteca de classe Java. A classe String é usada para armazenamento de texto e muitas funções de tratamento de texto.

O terceiro objeto, hungry, é uma variável boolean, ou seja, do tipo booleana, que pode armazenar apenas um de dois valores: true ou false. Esse objeto é usado para controlar se o jabberwock está com fome (hungry) ou satisfeito (full), true ou false, respectivamente.

As variáveis booleanas representam um tipo especial de variável que pode conter apenas os valores `true` ou `false`. Ao contrário de outras linguagens, em Java as variáveis booleanos não possuem valores numéricos, onde 1 é equivalente a `true` e 0 é equivalente a `false`. O termo "booleano" homenageia George Boole, um matemático irlandês que viveu de 1 815 a 1 864. Seu outro homônimo é a álgebra booleana, que é fundamental para a programação de computadores, eletrônica digital e lógica.

Você pode acrescentar algum comportamento à classe Jabberwock pela inclusão de métodos. Existem muitas coisas que um jabberwock pode fazer (agarrar aquele bocado, morder aquele pedaço etc), mas para abreviar as coisas, duas são inseridas — uma para alimentar o jabberwock e outra para verificar seus atributos.

Para começar, inclua as seguintes linhas imediatamente abaixo das três variáveis de instância em sua definição de classe:

```
void feedJabberwock( ) {
   if (hungry == true) {
        System.out.println("Yum - a peasant!");
        hungry = false;
   } else
        System.out.println("No, thanks - already ate.");
}
// tem mais
```

A última linha, `// tem mais`, é uma linha de comentário. Os comentários são usados para proveito de alguém que esteja examinando o código-fonte, para que possa saber o que ele está fazendo. Os computadores não têm nenhum interesse neles — tudo o que estiver após as barras `//` iniciais até o final da linha será ignorado pelo compilador Java. Na classe Jabberwock, o comentário está sendo usado como um lugar reservado. Você o substituirá em breve.

O método feedJabberwock( ) faz um teste para ver se um objeto Jabberwock está com fome ("hungry", na linha `if (hungry == true)`). Se estiver, o objeto é alimentado (até se fartar) e o estado de hungry é mudado para fal se. Se o objeto não estiver com fome, uma mensagem é apresentada dizendo que o jabberwock já comeu. O programa deverá estar assim até o momento:

Listagem 2.1 O texto atual de Jabberwock. java

```
1: class Jabberwock {
2:  String color;
3:  String sex;
4:  boolean hungry;
```

Listagem 2.1 Continuação

```
5:
6:  void feedJabberwock( ) {
7:    if (hungry == true) {
8:      System.out.println("Yum - a peasant!");
9:      hungry = false;
10:   } else
11:     System.out.println("No, thanks - already ate.");
12: }
13:
14: // tem mais
15: }
```

A endentação e as linhas em branco usadas para criar espaços no arquivo-fonte são desprezadas pelo compilador Java. Assim como os comentários, elas são introduzidas para benefício dos programadores, para que a lógica do programa seja mais fácil de seguir. A endentação e o espaçamento utilizados aqui, com linhas em branco entre métodos e variáveis e métodos endentados, serão usados em todo este livro. A biblioteca de classe Java utiliza uma endentação semelhante. Você pode escolher qualquer estilo de endentação que preferir.

Antes de compilar essa classe, você precisa incluir mais um método. O método showAttributes( ) apresenta os atuais valores das variáveis de instância em uma instância de sua classe Jabberwock.

No programa, exclua a linha de comentário // tem mais e substitua-a pelo seguinte:

```
void showAttributes( ) {
    System.out.println("This is a " + sex + " " + color + " jabberwock.");
    if (hungry •= true)
        System.out.println("The jabberwock is hungry.");
    else
        System.out.println("The jabberwock is full.");
 }
```

O método showAttributes( ) apresenta duas linhas na tela: os atributos sex e color (sexo e cor) do objeto Jabberwock e se ele está com fome (hungry). Salve o arquivo-fonte em seu editor de textos, certificando-se de atribuir-lhe o nome Jabberwock. java, para que o nome de arquivo combine com o nome da classe.

Nesse ponto, você tem uma classe Jabberwock com variáveis de instância e métodos de instância que podem ser usados para apresentar e modificar essas variáveis.

Utilize um dos seguintes procedimentos para compilar o programa, dependendo do sistema que você estiver usando.

WINDOWS Mude para a pasta que contém seu arquivo-fonte Java, usando o comando cd no prompt do MS-DOS e use o comando javac para compilá-lo:

ENTRADA
```
javac Jabberwock.java
```

SOLARIS A partir da linha de comando, mude para o diretório que contém seu arquivo-fonte Java, usando o comando cd, e use o comando javac para compila-lo:

```
javac Jabberwock.java
```

Se você encontrar quaisquer problemas na compilação do programa, verifique a existência de erros tipográficos, confrontando-o com a listagem 2.2.

DIGITE **Listagem 2.2** O texto corrente de Jabberwock. java.

```
1: class Jabberwock {
2:       String color;
3:       String sex;
4:       boolean hungry;
5:
6:   void feedJabberwock( ) {
7:      if (hungry »• true) {
8:        System.out.println("Yum - a peasant!");
9:        hungry = false;
10:     } el se
11:         System.out.println("No, thanks - already ate.");
12:   }
13:
14:   void showAttributes( ) {
15:     System.out.println("This is a " + sex + " " + color
16: + " jabberwock.");
17:      if (hungry -= true)
18:        System.out.println("The jabberwock is hungry.");
19:     else
20:        System.out.println("The jabberwock is full.");
21:   }
22: }
```

Executando o programa

Se você executar o arquivo Jabberwock. class com uma ferramenta de linha de comando como o interpretador Java, obterá um erro como o seguinte:

```
In classe Jabberwock: void main(String argv[ ]) is not defined
```

Esse erro ocorre porque o interpretador Java assume que o programa é um aplicativo, quando você tenta executá-lo na linha de comando. Quando um

aplicativo é executado, o ponto de partida do programa é seu método main( ). Como a classe Jabberwock não possui um método main( ), o interpretador não sabe o que fazer com ela.

Existem duas maneiras de usar a classe Jabberwock:

- Criar um applet ou aplicativo Java separado que utilize essa classe.
- Incluir um método main( ) na classe Jabberwock, para que ela possa ser executada diretamente.

A última maneira é usada para este exercício. Carregue o arquivo Jabberwock.java em um editor de textos e insira uma linha em branco imediatamente abaixo da última linha do programa (linha 21, na listagem 2.2).

Nessa linha, insira o seguinte:

```
public static void main (String arguments[ ]) {
    Jabberwock j = new Jabberwock( );
    j.color = "orange";
    j.sex = "male";
    j.hungry = true;
    System.out.println("Calling showAttributes ...");
    j.showAttributes( );
    System.out.println("———");
    System.out.println("Feeding the jabberwock ...");
    j.feedJabberwock( );
    System.out.println("———");
    System.out.println("Calling showAttributes ...");
    j.showAttributes( );
    System.out.println("———");
    System.out.println("Feeding the jabberwock ...");
    j.feedJabberwock( );
}
```

Com o método main( ), a classe Jabberwock pode agora ser usada como um aplicativo. Salve e compile o arquivo.

A listagem2.3 mostra o arquivo-fonte Jabberwock.java final, para o caso de você ter problemas para compilá-lo.

Se você tiver problemas com qualquer programa deste livro, pode encontrar uma cópia do arquivo-fonte e de outros arquivos relacionados no site da Web oficial do livro, no endereço http://www.prefect.com/java21.

Listagem 2.3 A versão final de Jabberwock.java.

```
1: class Jabberwock {
2:     String color;
3:     String sex;
4:     boolean hungry;
```

Listagem 2.3 Continuação

```
5:
6:      void feedJabberwock( ) {
7:           if (hungry == true) {
8:                System.out.println("Yum --a peasant!");
9:               hungry = false;
10:          } else
11:               System.out.println("No, thanks -- already ate.");
12:     }
13:
14:     void showAttributes( ) {
15:          System.out.println("This is a " + sex + " " + color + "jabberwock.");
16:          if (hungry == true)
17:               System.out.println("The jabberwock is hungry.");
18:          else
19:               System.out.println("The jabberwock is full.");
20:     }
21:
22:     public static void main (String argumentsf ]) {
23:          Jabberwock j = new Jabberwock( );
24:       j.color = "orange";
25:          j.sex = "male";
26:          j.hungry - true;
27:           System.out.printlnCCalling showAttributes ...");
28:          j.showAttributes( );
29:          System.out.println("———");
30:           System.out.println("Feeding the jabberwock ...");
31:          j.feedJabberwock( );
32:          System.out.println("———");
33:           System.out.println("Calling showAttributes ...");
34:          j.showAttributes( );
35:          System.out.println("———");
36:           System.out.println("Feeding the jabberwock ...");
37:          j.feedJabberwock( );
38:     }
39: }
```

O aplicativo Jabberwock pode ser executado usando-se um dos procedi-

WINDOWS A partir do prompt do MS-DOS, vá para a pasta que contém o arquivo Jabberwock.class, usando o comando cd. Use o comando java para executa-lo com o interpretador, como se segue:

```
java Jabberwock
```

SOLARIS A partir de uma linha de comando, vá para o diretório que contém o arquivo Jabberwock.class, usando o comando cd. Use o comando java para executa-lo com o interpretador, como se segue:

```
java Jabberwock
```

Quando você executar a classe Jabberwock, a saída deverá ser a seguinte:

SAÍDA
```
Calling showAttributes ...
This is a male orange jabberwock.
The jabberwock is hungry.

Feeding the jabberwock ...
Yum -- a peasant!

Calling showAttributes ...
This is a male orange jabberwock.
The jabberwock is full.

Feeding the jabberwock ...
No, thanks -- already ate.
```

Usando a listagem 2.3 como guia, acontecerá o seguinte no método main( ):

- Linha 22 — O método main( ) é declarado. A primeira linha do método main( ) sempre é semelhante a esta e você aprenderá sobre cada elemento dessa instrução posteriormente, nesta semana.
- Linha 23, Jabberwock j = new Jabberwock ( );—Cria um novo objeto da classe Jabberwock e armazena uma referência a ela em uma nova variável, chamada j. Conforme você já aprendeu, normalmente não se opera diretamente nas classes em seus programas Java. Em vez disso, você cria objetos a partir dessas classes e chama métodos desses objetos para operar nelas.
- Linhas 24 a 26 — As variáveis de instância color, sex e hungry são definidas para o objeto Jabberwock criado na linha 2. O atributo col or é orange, sex é male e a variável hungry recebe o valor booleano true. Isso indica que o objeto Jabberwock está com fome.
- Linha 27 — Nessa linha e em várias outras que seguem, uma instrução System. out. printl n ( ) é utilizada para apresentar informações na tela. Tudo o que estiver entre parênteses será apresentado.
- Linha28 — O método showAttributes( ), definido no objeto Jabberwock, é chamado. Isso faz o objeto Jabberwock indicar os valores de suas variáveis de instância: color, sex e hungry.

- Linha 31 — O método feedJabberwock( ) do objeto Jabberwock é chamado, o que muda o valor da variável hungry de true para false e apresenta um comentário apreciativo do objeto Jabberwock: "Yum -- a peasant!".
- Linha 33 — O método showAttributes( ) é chamado novamente para apresentar os valores das variáveis de instância do objeto Jabberwock. Desta vez, a saída deve refletir que esse jabberwock está satisfeito (pois hungry tem o valor fal se).
- Linha 36 — O método feedJabberwock( ) é chamado novamente, em uma tentativa de alimentar o jabberwock. Como ele já está satisfeito, recusa-se a comer com um educado "No, thanks -- already ate" (em português: "Não, obrigado, já comi.").

Nesse ponto, presume-se que você saiba como compilar e executar corretamente um aplicativo Java. Se necessário, revise o Dia 1 e a documentação de sua ferramenta de desenvolvimento Java para obter mais informações.

Organizando classes e comportamento de classe

Um exemplo introdutório da programação orientada a objetos em Java não está completo sem um primeiro exame em três conceitos: herança, interfaces e pacotes.

Todos esses três itens são mecanismos para se organizar as classes e comportamento de classe. A biblioteca de classe Java utiliza esses conceitos e as classes que você cria para os seus próprios programas também necessitam deles.

Herança

A herança representa um dos conceitos mais importantes na programação orientada a objetos e tem efeito direto em como você projeta e escreve suas próprias classes Java.

Herança é um mecanismo que permite a uma classe herdar todo o comportamento e os atributos de outra classe.

Através da herança, uma classe possui imediatamente toda a funcionalidade de uma classe já existente. Por causa disso, a nova classe pode ser criada indicando-se apenas sua diferença em relação à outra já existente.

Com a herança, todas as classes são organizadas em uma hierarquia rigorosa — aquelas que você cria e aquelas da biblioteca de classe Java e de outras bibliotecas.

NOVO TERMO Uma classe que herda de outra é chamada *subclasse* e a classe que fornece a herança é chamada *superdasse*.

Uma classe pode ter apenas uma superclasse, mas cada classe pode ter um número ilimitado de subclasses. As subclasses herdam todos os atributos e o comportamento de suas superclasses.

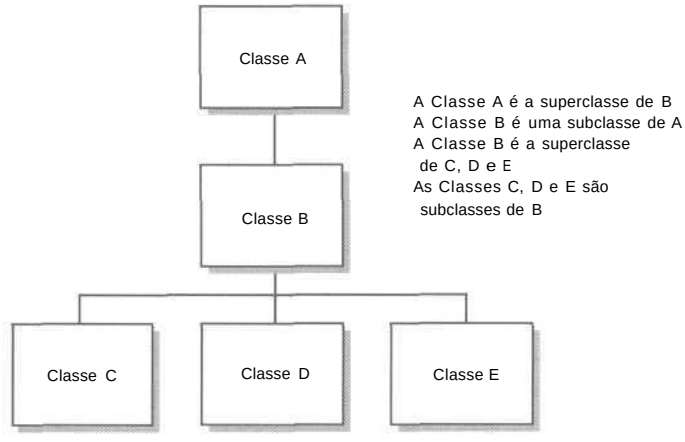
Em termos práticos, isso significa que, se a superclasse tiver comportamento e atributos de que sua classe precisa, você não precisa redefini-la ou copiar esse código para ter o mesmo comportamento e atributos. Sua classe recebe automaticamente essas coisas de sua superclasse, esta os recebe de sua superclasse e assim por diante, até o início da hierarquia. Sua classe se torna uma combinação de todos os recursos das classes que estão acima dela na hierarquia, assim como de seus próprios recursos.

A situação é bastante comparável ao modo como você herda todos os tipos de coisas de seus pais, como altura, cor de cabelos, gosto por *ska music* e uma relutância em pedir orientação. Eles herdaram algumas dessas coisas de seus pais, que herdaram do avós, até chegar ao Jardim do Éden, ao Big Bang ou *insira sua crença cosmológíca pessoal aqui.*

A Figura 2.2 mostra a organização de uma hierarquia de classes.

Figura 2.2

Uma *hierarquia de classe.*



No topo da hierarquia de classe Java está a classe Obj ect — todas as classes herdam dessa superclasse. Object é a classe mais geral da hierarquia e define o comportamento e os atributos herdados por todas as classes da biblioteca de classe Java. Cada classe que está mais abaixo na hierarquia se torna mais personalizada para um propósito específico. Uma hierarquia de classe define conceitos abstratos no topo da hierarquia. Esses conceitos se tornam mais concretos mais abaixo na linha de subclasses.

Freqüentemente, quando criar uma nova classe em Java, você desejará toda a funcionalidade de uma classe já existente, com algumas modificações de sua própria criação. Por exemplo, você poderia querer uma versão de um CommandButton que produzisse um som explosivo e ensurdecedor ao receber um clique de mouse. (Nem os autores deste livro nem a editora sugerem que isso seja uma boa idéia, e não terão responsabilidades legais no caso de tal som não ser apreciado por aqueles que ficarem inesperadamente surdos por causa dele.)

Para receber toda a funcionalidade de CommandButton sem fazer nada para recriá-lo, você pode definir sua classe como uma subclasse de CommandButton. Sua classe herdaria automaticamente o comportamento e os atributos definidos em CommandButton e o comportamento e os atributos definidos na superclasse de CommandButton. Você só precisa se preocupar com o que torna sua classe diferente da própria CommandButton. O uso de subclasses é um mecanismo para a definição de novas classes por meio das diferenças entre essas classes e sua superclasse.

NOVO TERMO *Uso de subclasse* é a criação de uma nova classe que herda de outra já existente. A única tarefa na subclasse é indicar as diferenças no comportamento e nos atributos, entre ela e a superclasse.

Se sua classe define um comportamento inteiramente novo e não é uma subclasse de outra, você pode herdar diretamente da classe Object. Isso permite que ela se encaixe perfeitamente na hierarquia de classe Java. Na verdade, se você criar uma definição de classe que não indique uma superclasse, a linguagem Java irá pressupor que a nova classe está herdando diretamente de Object. A classe Jabberwock que você criou herda da classe Object.

Criando uma hierarauia de classe

Se você estiver criando um grande conjunto de classes, faz sentido que suas classes herdem da hierarquia de classe existente e que elas mesmas constituam uma hierarquia. Organizar suas classes dessa forma requer um planejamento significativo, mas as vantagens incluem o seguinte:

- A funcionalidade comum a várias classes pode ser colocada em superclasses, o que permite que ela seja usada repetidamente em todas as classes que estejam abaixo na hierarquia.
- As alterações em uma superclasse são refletidas automaticamente em todas as suas subclasses, nas subclasses destas etc. Não há necessidade de mudar ou recompilar nenhuma das classes inferiores; elas recebem as novas informações por herança.

Por exemplo, imagine que você tenha criado uma classe Java para implementar todas as características de um jabberwock. (Isso não deverá exigir muita imaginação, se você não pulou direto para este ponto do dia.)

A classe Jabberwock está concluída, funciona corretamente e tudo está OK. Agora, você quer criar uma classe Java chamada Dragon.

Os jabberwocks e dragões possuem muitas características semelhantes — ambos são grandes e comem camponeses. Ambos possuem presas afiadas, dentes poderosos e personalidades tipo A. Seu primeiro impulso poderia ser abrir o arquivo-fonte Jabberwock. java e copiar uma grande parte dele em um novo arquivo-fonte, chamado Dragon. java.

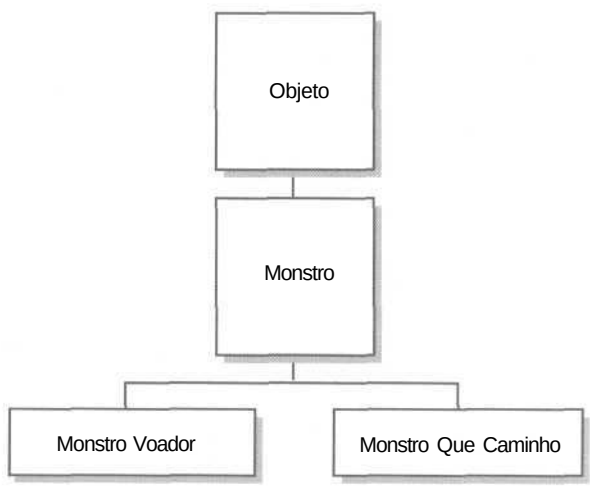
Um plano melhor é descobrir a funcionalidade comum de Dragon e Jabberwock, organizando-a em uma hierarquia de classe mais geral. Isso poderia ser trabalhoso, apenas para as classes Jabberwock e Dragon, mas e se você também quiser incluir Medusa, Yeti, Sasquatch, Grue e DustBunny? Colocar o comportamento comum em uma ou mais superclasses reutilizáveis reduz significativamente o trabalho global a ser realizado.

Para projetar uma hierarquia de classe que possa atender a esse propósito, comece no topo, com a classe Object, o pináculo de todas as classes Java. A classe mais geral a que um jabberwock e um dragão pertencem poderia ser chamada de Monster. Um monstro, de maneira geral, poderia ser definido como uma criatura feroz de algum tipo, que aterroriza as pessoas e diminui os valores das propriedades. Na classe Monster, você define apenas o comportamento que qualifica algo que é feroz, terrível e ruim para a vizinhança.

Poderiam existir duas classes abaixo de Monster: FlyingMonster (monstro voador) e Wal kingMonster (monstro que caminha). A evidência que diferencia essas classes é que um pode voar e outro, não. O comportamento dos monstros voadores poderia incluir mergulho sobre a presa, carregar camponeses pelo céu, deixá-los nos picos das montanhas etc. Os monstros que caminham se comportariam de forma diferente e seriam muito mais suscetíveis a quedas. A Figura 2.3 mostra o que você tem até aqui.

Figura 2.3

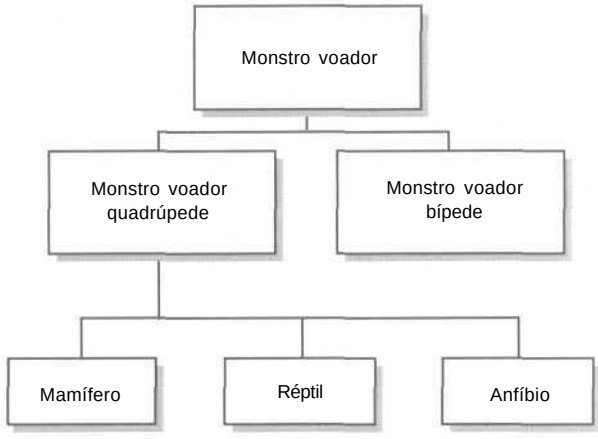
A hierarquia Monster básica.



Agora, a hierarquia pode se tornar ainda mais específica. Com FlyingMonster, você poderia ter várias classes: Mammal (mamífero), Repti 1 e (réptil), Amphibian (anfíbio) etc. Como alternativa, você poderia decompor ainda mais a funcionalidade e ter classes intermediárias para monstros TwoLegged (bípedes) e FourLegged (quadrúpedes), com diferentes comportamentos para cada um (ver Figura 2.4).

Figura 2.4

Monstros voadores bípedes e quadrúpedes.



Finalmente, a hierarquia está terminada e você tem um lugar para Jabberwock. Ela pode ser uma subclasse de monstros voadores que são répteis, voadores e quadrúpedes. (Na verdade, indo-se até o início da hierarquia de classe, Jabberwock seria uma subclasse de objetos monstros répteis voadores e quadrúpedes — pois FlyingMonster é uma subclasse de Monster e esta é uma subclasse de Object.

E onde entram qualidades como sexo, cor ou apetite? Elas entram no lugar em que se enquadram mais naturalmente na hierarquia de classe. Você pode definir sex e color como variáveis de instância em Monster, e todas as subclasses também possuirão essas variáveis. Lembre-se de que você precisa definir um comportamento ou atributo apenas uma vez na hierarquia e ele será herdado automaticamente por cada subclasse.

Nota

Projetar uma hierarquia de classe eficiente envolve muito planejamento e revisão. A medida que tentar colocar atributos e comportamento em uma hierarquia, provavelmente você encontrará razões para mover algumas classes para locais diferentes nessa hierarquia. O objetivo é reduzir o número de características repetitivas que são necessárias. Se você fosse projetar uma hierarquia de monstros, talvez quisesse colocar Mammal, Repti 1 e e Amphi bi an imediatamente abaixo de Monster, caso isso descreva melhor a funcionalidade para a qual está usando classes para incorporar.

A herança em ação

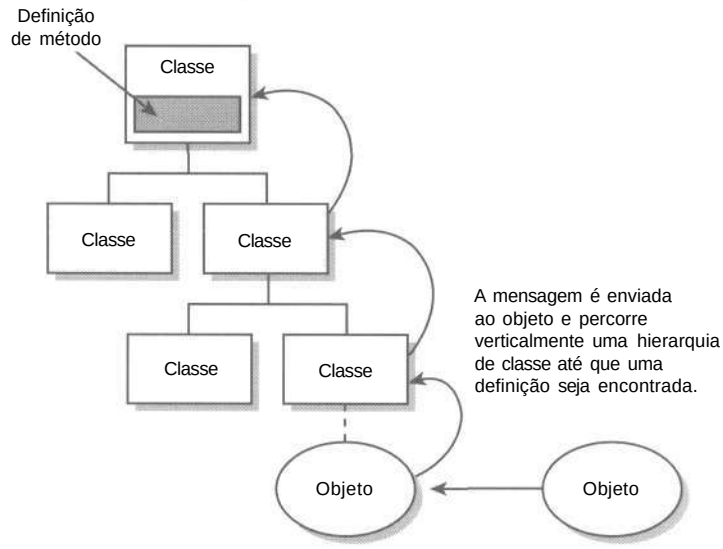
Na linguagem Java, a herança funciona de modo muito mais simples do que no mundo real. Em Java, não existem testamenteiros, juizes ou tribunais de qualquer espécie.

Quando você cria um novo objeto, a linguagem Java controla cada variável definida para esse objeto, assim como cada variável definida para cada superclasse do objeto. Desse modo, todas as classes se combinam para formar um modelo do objeto atual e cada objeto preenche as informações apropriadas para sua situação.

Os métodos funcionam de modo semelhante: os novos objetos têm acesso a todos os nomes de método de sua classe e de sua superclasse. Isso é determinado dinamicamente, quando um método é utilizado em um programa que está em execução. Se você chamar um método de um objeto em particular, o interpretador Java procurará primeiro a classe do objeto para esse método. Se o método não for encontrado, o interpretador procurará na superclasse dessa classe e assim por diante, até que a definição do método seja encontrada. Isso está ilustrado na Figura 2.5.

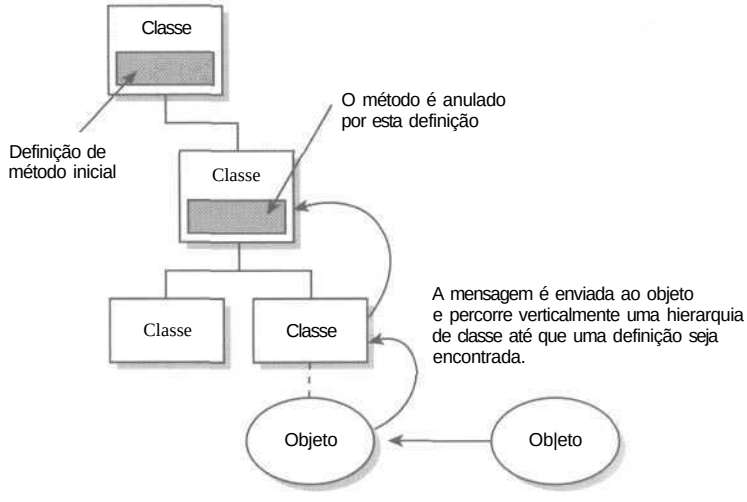
Figura 2.5

Como os *métodos são localizados* em uma *hierarquia de classe.*



As coisas se complicam quando uma subclasse define um método que possui o mesmo nome, tipo de retorno e argumentos que um método definido em uma superclasse. Nesse caso, a definição do método que for encontrada primeiro (do final para o início da hierarquia) é a que será usada. Por isso, você pode criar um método em uma subclasse que impeça a utilização de um outro de uma superclasse. Para fazer isso, você atribui ao método o nome, o tipo de retorno e os argumentos iguais ao da superclasse. Esse procedimento é chamado de *anulação* (ver Figura 2.6).

Figura 2.6
Anulando métodos.



Herança simples e múltipla

A forma de herança da linguagem Java é chamada de *herança simples*, pois cada classe Java pode ter apenas uma superclasse (embora uma dada superclasse possa ter várias subclasses).

Em outras linguagens de programação orientada a objetos, como C + +, as classes possuem mais de uma superclasse, e elas herdam variáveis e métodos combinados de todas essas superclasses. Isso se chama *herança múltipla* e proporciona os meios para se criar classes que abrangem praticamente todo comportamento que se possa imaginar. Entretanto, isso complica significativamente as definições de classe e o código necessário para produzi-las. A linguagem Java torna a herança mais fácil permitindo apenas herança simples.

Interfaces

A herança simples torna o relacionamento entre classes e a funcionalidade que elas implementam mais fáceis de entender e projetar. Entretanto, isso também pode ser restritivo — especialmente quando você tem comportamento semelhante que precisa ser duplicado em diferentes ramos de uma hierarquia de classe. A linguagem Java resolve o problema do comportamento compartilhado usando interfaces.

NOVO TERMO Uma *interface* é um conjunto de métodos que indica que uma classe possui algum comportamento além daquele herdado de suas superclasses.

As interfaces serão um assunto melhor explorado depois que você tiver algum tempo para trabalhar com a programação orientada a objetos e com a linguagem Java. Você irá explorá-las integralmente no Dia 15.

Pacotes

Na linguagem Java, os *pacotes* representam um modo de agrupar classes e interfaces relacionadas. Os pacotes permitem que grupos de classes estejam disponíveis somente se forem necessários, além de eliminarem os conflitos em potencial entre nomes de classe em diferentes grupos de classes.

Por enquanto, existem apenas algumas coisas que você precisa saber:

- *As bibliotecas de classe na linguagem Java estão contidas em um pacote chamado* java. Há garantia de que as classes do pacote java estejam disponíveis em qualquer implementação Java e são as únicas que com certeza estão disponíveis em diferentes implementações. O pacote java contém pacotes menores que definem subconjuntos específicos da funcionalidade da linguagem Java, como recursos padrão, tratamento de arquivos, multimídia e muitas outras coisas. As classes de outros pacotes, como sun e netscape, freqüentemente estão disponíveis apenas em implementações específicas.

- *Por definição, suas classes Java têm acesso apenas às classes de* java. 1 ang *(recursos básicos da linguagem).* Para usar classes de qualquer outro pacote, você pode fazer referência a elas explicitamente pelo nome do pacote ou importá-las em seu arquivo-fonte.

- *Para fazer referência a uma classe dentro de um pacote, você deve listar todos os pacotes em que a classe está contida, seguidos do nome da classe, com cada elemento separado por pontos finais (.).* Por exemplo, considere a classe Color. Ela está contida no pacote awt que, por sua vez, está contido no pacote java. Para fazer referência à classe Color em seus programas, a notação java.awt.Color pode ser usada.

Criando uma subclasse

Como um projeto final para hoje, você cria uma subclasse de outra classe e anula alguns métodos. Você também tem um melhor entendimento de como os pacotes funcionam.

Quando você começa a programar em Java, o uso mais comum de subclasses se dá quando são criados applets. A criação de applets é diferente da criação de aplicativos. Os applets Java são executados como parte de uma página da Web e, portanto, possuem regras especiais para seu comportamento. Devido a essas regras especiais dos applets, a criação de um applet simples é mais complicada do que a criação de um aplicativo simples.

Todos os applets são subclasses da classe Applet (que faz parte do pacote java.applet). Criando uma subclasse de Appl et, você recebe automaticamente todo o comportamento e os atributos que permitem a um programa Java a ser executado como parte de uma página da Web.

Nesse exemplo, você cria um applet semelhante ao aplicativo Hel 1 oDan de ontem. Para iniciar o exemplo, construa primeiro a definição de classe em si. Carregue seu editor de textos e digite as seguintes instruções:

```
public class Palindrome extends java.applet.Applet {
// tem mais
}
```

Isso define uma classe chamada Pal indrome. As instruções são semelhantes ao modo como você criou a classe HelloDan durante o Dia 1. Uma novidade é o texto extends java. appl et. Appl et.

A cláusula extends é o modo de declarar que uma classe é subclasse de outra. A classe Palindrome é uma subclasse da classe Applet, que faz parte do pacote java.applet. Para indicar isso em um programa, a cláusula extends java. appl et. Appl et é usada para definir o relacionamento entre as duas classes.

Como a classe Applet está contida no pacote java.applet, você não tem acesso automático a essa classe; portanto, tem de fazer referência a ela explicitamente, pelo nome do pacote e da classe. As únicas classes a que você pode fazer referência sem considerar o nome do pacote são aquelas do pacote java.lang.

Outro elemento novo na instrução class é a palavra-chave public. Essa palavra-chave indica que sua classe estará acessível para outras classes que podem precisar usá-la. Normalmente, você só precisa tornar uma classe pública (compubl ic), se quiser que ela seja utilizada por outras classes em seu programa Java. Entretanto, todos os applets devem ser public.

Uma definição de classe que não contém nada a não ser a instrução de comentário // tem mais, não faz muita coisa — ela não acrescenta nada de novo nem anula métodos ou variáveis de sua superclasse. Para tornar a classe Palindrome diferente de sua superclasse, exclua a linha de comentário // tem mais e comece a inserir novas instruções no programa, partindo da seguinte:

```
Font f = new Font("TimesRoman", Font.BOLD, 36);
```

Essa instrução realiza duas coisas:

- Um objeto Font, chamado f, é criado. Font, parte do pacote java.awt, é usado para representar uma fonte de tela. Ele é utilizado para apresentar uma fonte e um estilo de fonte diferentes do padrão normalmente usado em um applet.

- O objeto Font recebe o valor de uma fonte Times Roman, em negrito, de 36 pontos. A instrução new cria um novo objeto Font, com os valores especificados entre parênteses, e esse novo objeto é atribuído a f.

Criando uma variável de instância para conter esse objeto Font, você o torna disponível para todos os métodos de sua classe. A próxima etapa no projeto Pal indrome é criar um método que o utilize.

Quando você escreve applets, existem vários métodos definidos na superclasse Applet que normalmente são anulados em seu applet. Isso inclui os métodos para configurar o applet antes que ele seja executado, para iniciá-lo, para responder à entrada de mouse e para fazer a limpeza quando ele encerrar sua execução.

Um desses métodos é o paint( ), que trata de tudo o que deve ocorrer quando o applet é apresentado em uma página da Web. O método paint( ) herdado por Pal indrome não faz nada — trata-se de um método vazio. Anulando paint( ), você indica o que deve ser desenhado na janela do applet, quando ela precisar ser apresentada no momento em que o programa for executado. Inclua uma linha em branco abaixo da instrução Font e insira o seguinte para anular o método paint( ):

```
public void paint(Graphics screen) {
    screen.setFont(f);
    screen.setColor(Color.red);
    screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
}
```

O método pai nt ( ) é declarado publ i c, assim como o próprio applet, mas por uma razão diferente. Neste caso, paint( ) deve ser público porque o método que ele está anulando também é público. Um método de superclasse que é publico só pode ser anulado por um método público, senão o programa Java não será compilado com sucesso.

O método paint( ) recebe um argumento: uma instância da classe Graphics, chamada screen. A classe Graphics fornece o comportamento para renderizar fontes e cores e desenhar linhas e outras figuras. Você aprenderá mais sobre a classe Graphi cs na Semana 2, quando criará mais applets.

Você fez três coisas dentro de seu método pai nt ( ):

- Informou ao objeto Graphics que a fonte a ser usada quando exibir texto é aquela contida na variável de instância f.
- Informou ao objeto Graphi cs que a cor a ser usada em texto e em outras operações de desenho é uma instância da classe Color para a cor red (vermelha).
- Finalmente, você desenhou o texto "Go hang a salami, I'm a lasagna hog!" na tela, na coordenada x,y referente a 5, 40. O string será renderizado na fonte e na cor indicadas.

Aqui está como é o applet até o momento:

```
public class Palindrome extends java.applet.Applet {
    Font f = new Font("TimesRoman", Font.BOLD, 36);

    public void paint(Graphics screen) {
        screen.setFont(f);
        screen.setColor(Color.red);
        screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
    }
}
```

Talvez você tenha notado que está faltando algo no exemplo até este ponto. Se você salvasse o arquivo e tentasse compilá-lo, veria muitos erros como o seguinte:

```
Palindrome.java:2: Class Font not found in type declaration.
```

Esses erros estão ocorrendo porque as classes Graphics, Font e Color fazem parte do pacote java.awt e não estão disponíveis por padrão. Você fez referência à classe Applet na primeira linha da definição de classe, através de seu nome de pacote completo (java. appl et. Applet). No restante do programa, você fez referência a outras classes sem usar seus nomes de pacote.

Existem duas maneiras de resolver esse problema:

- Fazer referência a todas as classes externas usando seu nome de pacote completo, como em java. awt. Graphi cs, j ava. awt. Font e j ava. awt. Col or.
- Usar uma instrução import no início do programa para tornar um ou mais pacotes e classes disponíveis no programa.

O que você escolhe é principalmente uma questão de preferência pessoal, mas se você fizer referência a uma classe de outro pacote várias vezes, talvez queira usar a instrução import para diminuir a quantidade de digitação.

Neste exemplo, usaremos a última opção.

Para importar essas classes, inclua as três instruções a seguir, acima da instrução public class Palindrome:

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
```

Dica Você também pode importar um pacote inteiro usando um asterisco (*) no lugar de um nome de classe específico. Por exemplo, você pode usar a seguinte instrução para importar todas as classes do pacote java.awt:

```
import java.awt.*;
```

Agora, com as classes corretas importadas para seu programa, Palindrome. java deve compilar sem problemas em um arquivo de classe. A listagem 2.4 mostra a versão final para conferência.

DIGITE

Listagem 2.4 A versão final de Palindrome.java.

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class Palindrome extends java.applet.Applet {
6:       Font f = new Font("TimesRoman", Font.BOLD, 36);
7:
8:       public void paint(Graphics screen) {
9:            screen.setFont(f);
10:           screen.setColor(Color.red);
11:          screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
12:      }
13: }
```

Salve esse arquivo como Palindrome.java. Como esse arquivo-fonte contém uma classe pública, Pal indrome, o nome de arquivo deve combinar com o nome da classe pública para que ele seja compilado com sucesso. As letras maiúsculas e minúsculas são levadas em consideração aqui; portanto, elas devem ser idênticas.

Esse arquivo-fonte pode ser compilado da mesma maneira que os aplicativos Java que você criou até agora. Entretanto, para executá-lo, você precisa criar uma página da Web para, como o Claris, colocá-lo lá.

Muitos programas de desenvolvimento de página da Web Home Page e o Macromedia Dreamweaver, permitem que um applet Java seja colocado em uma página da Web.

Se você não possui uma dessas ferramentas, pode criar uma página da Web simples usando os recursos relacionados a Java da linguagem de descrição de página HTML.

Dica

Embora alguns recursos da HTML sejam descritos neste livro, à medida que eles se relacionam com a linguagem Java, está fora dos objetivos deste livro ensinar HTML e desenvolvimento de páginas da Web. Os co-autores deste livro escreveram outros sobre o assunto, incluindo *Aprenda em 1 semana HTML* (publicado no Brasil pela Editora Campus), de Laura Lemay, e *Aprenda em 24 Horas a Criar uma Home Page,* de Rogers Cadenhead.

Para criar uma nova página HTML que possa conter o applet Pal i ndrome, carregue o mesmo editor de textos que você está usando para criar programas Java e inicie um novo documento.

Insira a listagem 2.5 e salve o arquivo como Pa1indrome.html, na mesma pasta que contém Palindrome.java e Pal indrome. class. Se você estiver usando

o Windows 95, coloque o nome do arquivo entre aspas para garantir que a extensão .txt não seja incluída.

DIGITE > **Listagem 2.5** A pagina da Web Palindrome.html.

```
1: <APPLET CODE="Palindrome.class" WIDTH=600 HEIGHT=100>
2: </APPLET>
```

Você aprenderá mais sobre a tag <APPLET> da HTML posteriormente neste livro, mas aqui existem duas coisas a serem observadas:

- O atributo CODE indica o nome da classe que contém o applet — Pal indrome.class, neste exemplo.
- Os atributos WIDTH e HEIGHT determinam qual será o tamanho da janela do applet em uma página da Web, em pixels. Neste exemplo, a janela terá 600 pixels de largura e 100 pixels de altura.

Para ver esse applet, você precisa de um navegador da Web que possa executar applets Java ou da ferramenta appl etvi ewer, incluída no JDK.

Todos os applets deste livro utilizam apenas os recursos Java 1.0, a menos que se diga o contrário, para que os applets possam ser vistos com qualquer navegador que ofereça suporte a Java. Os aplicativos usam Java 2, pois eles podem ser executados diretamente com um interpretador Java 2.

Para abrir a página da Web Palindrome.html usando um navegador da Web, utilize um comando de menu suspenso do navegador para abrir arquivos locais, em vez de páginas da Web. No Netscape Navigator 4.04, o comando é File | Open Page | Choose File.

Para abrir a página com a ferramenta appletvi ewer do JDK, vá para o prompt de linha de comando, na mesma pasta que contém Pal indrome.html, e digite o seguinte comando:

```
appletviewer Palindrome.html
```

Ao contrário de um navegador da Web, a ferramenta appletvi ewer apresenta apenas o applet (ou applets) que está incluído na página da Web. Ela não manipula nada mais que esteja contido na página.

A Figura 2.7 mostra o applet visto com a ferramenta appl etvi ewer.

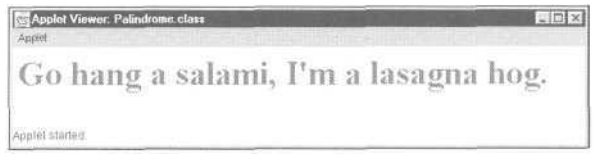


Figura 2.7

O app/ef *Palindrome* em execução na ferramenta *appletviewer*.

Dica

Se você ainda não estiver familiarizado com o que é palíndromo (palindrome, em inglês), dê uma olhada na Figura 2.7 e leia a frase "Go hang a salami, I'm a lasagna hog", ao contrário. Palíndromos são palavras ou frases que, se lidas da esquerda para a direita, ou da direita para a esquerda, têm o mesmo sentido, desconsiderando-se todos os espaços e a pontuação, como "As Satan sees Natasha" e "To Idi Amin: I'm an idiot". Esses últimos palíndromos foram extraídos da Gigantic List of Palindromes de Neil/Fred, no seguinte endereço da Web:

http://www.tsoft.net/~derf/pal indrome.html

Resumo

Se este foi o seu primeiro encontro com a programação orientada a objetos, talvez tenha descoberto outra maneira pela qual ela é semelhante à cerveja.

O programa orientado a objetos também é capaz de deixá-lo tonto, desorientado e, talvez, um pouco enjoado.

Se o material de hoje parece teórico e pesado até este ponto, não se preocupe. Você usará as técnicas orientadas a objetos pelo resto do livro e vai se familiarizar à medida que ganhar experiência em sua utilização.

Uma das grandes barreiras da programação orientada a objetos não são necessariamente os conceitos, mas sim, os nomes. A OOP tem mais jargão e linguagem técnica vagamente ameaçadora do que um episódio *de Arquivo X*.

Para resumir o material de hoje, há a seguir um glossário de termos e conceitos que foram abordados:

Classe Um modelo para um objeto que contém variáveis para descrevê-lo e métodos para descrever como ele se comporta. As classes podem herdar variáveis e métodos de outras classes.

Objeto Uma instância de uma classe. Vários objetos que sejam instâncias da mesma classe têm acesso aos mesmos métodos, mas freqüentemente possuem valores diferentes para suas variáveis de instância.

Instância O mesmo que objeto. Cada objeto é uma instância de alguma classe.

Método Um grupo de instruções em uma classe que define como os objetos da classe se comportarão. Os métodos são análogos às funções em outras linguagens, mas sempre devem estar localizados dentro de uma classe.

Método de classe Um método que opera em uma classe em si, em vez de operar em instâncias específicas de uma classe.

Método de instância Um método que opera em instâncias dessa classe, em vez de operar na classe em si. Como os métodos de instância são muito mais comuns do que os métodos de classe, eles freqüentemente são chamados apenas de métodos.

Variável de classe Uma variável que descreve um atributo de uma classe, em vez de instâncias específicas da classe.

Variável de instância Uma variável que descreve um atributo de uma instância de uma classe, em vez da classe em si.

Interface Uma especificação de comportamento abstrato que classes específicas podem então implementar.

Pacote Um conjunto de classes e interfaces. As classes de pacotes que não sejam o java.lang devem ser importadas explicitamente ou deve ser feita uma referência através de seus nomes de pacote e classe completos.

Subclasse Uma classe mais abaixo do que outra na hierarquia de classe, sua superclasse. A criação de uma nova classe a partir de outra já existente freqüentemente é chamada de uso de subclasse. Uma classe pode ter quantas subclasses forem necessárias.

Superclasse Uma classe acima de outra na hierarquia de classe, sua subclasse. Uma classe pode ter apenas uma superclasse imediatamente acima dela, mas essa classe também pode ter uma superclasse e assim por diante.

Perguntas e respostas

P **Na verdade, os métodos são funções definidas dentro de classes. Se eles se parecem com funções e agem como funções, por** que **não são chamados de funções?**

R Algumas linguagens de programação orientada a objetos os chamam de funções (o C++ os chama de *funções membro).* Outras fazem diferenciação entre funções dentro e fora do corpo de uma classe ou objeto, pois, nessas linguagens, o uso dos termos separados é importante para se entender como cada função trabalha. Como a diferença é relevante **em** outras linguagens e como o

termo *método* agora é usado Comumente na terminologia orientada a objetos, a linguagem Java utiliza o termo também.

P **Qual é a diferença entre variáveis e métodos de instância e seus correlates, variáveis e métodos de classe?**

R Quase tudo o que você fizer em um programa Java envolverá instâncias (também chamadas de objetos), em vez de classes. Entretanto, alguns comportamentos e atributos fazem mais sentido se forem armazenados na própria classe e não no objeto. Por exemplo, para criar uma nova instância de uma classe, você precisa de um método que esteja definido e disponível para a própria classe. Caso contrário, você entrará no dilema do ovo e da galinha — você não pode criar um novo objeto-filho sem um objeto-mãe já existente, que possua um método para fazer filhos, e nenhum objeto-mãe pode existir sem primeiro ter tido um filho.

SEMANA 1

Dia 3

Fundamentos do Java

Conforme você aprendeu, um programa em Java é composto de classes e objetos que, por sua vez, são compostos de métodos e variáveis. Os métodos são compostos de instruções e expressões, que são constituídos de operadores.

Nesse ponto, você poderia estar achando que Java é como as bonecas russas Matryoshka. Cada uma dessas bonecas parece ter uma boneca menor dentro dela, que é tão intrincada e detalhada como sua companheira maior.

Relaxe, babushka — este capítulo deixa para lá as bonecas grandes para revelar os menores elementos da programação Java. Você deixará de lado as classes, objetos e métodos por um dia e examinará o básico do que pode fazer em uma única linha de código Java.

São abordados os seguintes assuntos:

- Instruções e expressões Java
- Variáveis e tipos de dados
- Comentários
- Literais
- Aritmética
- Comparações
- Operadores lógicos

Devido à ligação da linguagem Java com C e C++, boa parte do material deste capítulo será familiar para os programadores que já conhecem essas linguagens. Quando necessário, notas técnicas como esta descreverão as diferenças específicas entre Java e as outras linguagens.

Instruções e expressões

Todas as tarefas que você quer realizar em um programa Java podem ser divididas em uma série de instruções.

Uma *instrução é* um comando simples escrito em uma linguagem de programação, que faz algo acontecer.

As instruções representam uma ação simples, que é realizada em um programa em Java. Todos os exemplos a seguir são instruções simples da linguagem Java:

```
int age = 30;

import java.awt.dnd;

System.out.println("You're not the boss of me!");

player.score = 41367;
```

Algumas instruções produzem um valor, como quando você soma dois números em um programa. Essas instruções são chamadas de expressões.

Uma *expressão é* uma instrução que resulta na produção de um valor. O valor pode ser armazenado para uso posterior no programa, utilizado imediatamente em outra instrução ou desprezado. O valor produzido por uma instrução é chamado de *valor de retorno.*

Algumas expressões produzem um valor de retorno numérico, como no exemplo da soma de dois números. Outras produzem um valor booleano — true ou false — ou podem até produzir um objeto Java. Elas serão discutidas posteriormente, ainda hoje.

Embora muitos programas Java listem uma instrução por linha, essa é uma decisão de formatação que não determina onde uma instrução termina e outra começa. Cada instrução em Java é terminada com um caractere de ponto-e-vírgula (;). Um programador pode colocar mais de uma instrução em uma linha e ela será compilada sem problemas. Por exemplo:

```
j.color = "lemon yellow"; j.hungry = false;
```

Na linguagem Java, as instruções são agrupadas usando-se chaves de abertura ({) e fechamento (}). Um grupo de instruções organizado entre esses caracteres é chamado de *bloco* ou *instrução de bloco,* e você aprenderá mais sobre elas no Dia 5.

Variáveis e tipos de dados

No aplicativo Jabberwock, que você criou no Dia 2, foram utilizadas variáveis para controlar as informações.

NOVO TERMO *Variáveis* são um lugar no qual as informações podem ser armazenadas enquanto um programa está em execução. O valor pode ser alterado em qualquer ponto do programa — dai o nome.

Para criar uma variável, você precisa dar-lhe um nome e identificar o tipo de informação que ela vai armazenar. Você também lhe dará um valor inicial, ao mesmo tempo em que a cria.

Existem três tipos de variáveis em Java: variáveis de instância, variáveis de classe e variáveis locais.

As *variáveis de instância*, conforme você aprendeu ontem, são usadas para definir os atributos de um objeto. As *variáveis de classe* definem os atributos de uma classe de objetos inteira e aplicam-se a todas as instâncias dela.

As *variáveis locais* são usadas dentro de definições de método ou mesmo em blocos de instruções menores dentro de um método. Elas só podem ser usadas enquanto o método ou bloco estiver sendo executado pelo interpretador Java e, depois disso, elas deixam de existir.

Embora os três tipos de variáveis sejam criados da mesma forma, as variáveis de classe e instância são usadas de uma maneira diferente das variáveis locais. Você aprenderá sobre as variáveis locais hoje e abordará as variáveis de instância e classe no Dia 4.

Nota

Ao contrário de outras linguagens, Java não possui *variáveis globais* (variáveis que podem ser usadas em todas as partes de um programa). As variáveis de instância e de classe são usadas para comunicar informações de um objeto para outro e isso substitui a necessidade de variáveis globais.

Criando variáveis

Antes de poder usar uma variável em um programa Java, você precisa criá-la declarando seu nome e o tipo de informação que ela irá armazenar. O tipo de informação é listado primeiro, seguido do nome da variável. Todos os exemplos a seguir são declarações de variáveis:

```
int highScore;
String username;
boolean gameOver;
```

Você aprenderá mais sobre os tipos de variável posteriormente ainda hoje, mas já poderia estar familiarizado com os tipos utilizados nesse exemplo. O tipo i int representa inteiros, boolean é usado para valores true/f alse e String é um tipo de variável especial, usada para armazenar texto.

As variáveis locais podem ser declaradas em qualquer lugar dentro de um método, exatamente como qualquer outra instrução Java, mas devem ser declaradas antes de poderem ser usadas. O lugar em que normalmente ficam as declarações de variável é imediatamente após a instrução que atribui o nome e identifica o método.

No exemplo a seguir, três variáveis são declaradas no início do método main( ) do programa:

```
public static void main (String arguments[ ] ) {
    int total;
    String reportTitle;
    boolean active;
}
```

Se você estiver criando diversas variáveis do mesmo tipo, pode declará-las todas na mesma instrução, separando seus nomes com vírgulas. A instrução a seguir cria três variáveis String, chamadas street, city e state:

```
String street, city, state;
```

As variáveis podem receber um valor ao serem criadas, usando-se um sinal de igualdade (=) seguido do valor. As instruções a seguir criam novas variáveis e fornecem-lhes valores iniciais:

```
int zipcode = 90210;

String name = "Brandon";

boolean cheatedOnKelly = true;

int age = 28, height = 70, weight = 140;
```

Conforme a última instrução indica, você pode atribuir valores a diversas variáveis do mesmo tipo usando vírgulas para separá-las.

As variáveis locais devem receber valores antes de serem usadas em um programa, senão o programa não será compilado. Por esse motivo, é considerada uma boa prática atribuir valores iniciais a todas as variáveis locais.

As definições de variável de instância e classe recebem um valor inicial dependendo do tipo de informação que contêm. Por exemplo:

- Variáveis numéricas 0
- Caracteres '\0'
- Booleanas false
- Objetos nul 1

Nomeando variáveis

Nota

Além disso, a linguagem Java usa o conjunto de caracteres Unicode, que inclui o conjunto de caracteres padrão, mais milhares de outros, para representar alfabetos internacionais. Os caracteres acentuados e outros símbolos podem ser usados em nomes de variáveis, desde que possuam um número de caractere Unicode.

Quando se atribui um nome a uma variável e ao usá-la em um programa, é importante lembrar que a linguagem Java faz diferenciação entre letras maiúsculas e minúsculas — a utilização dessas letras deve ser coerente. Por isso, um programa pode ter uma variável chamada X e outra chamada x — e uma rosa não é uma Rosa, não é uma ROSA.

Nos programas deste livro e de qualquer outra parte, as variáveis Java recebem nomes significativos que incluem várias palavras unidas. Para tornar mais fácil reconhecer as palavras, a seguinte regra geral é utilizada:

- A primeira letra do nome da variável é uma letra minúscula.
- Cada palavra sucessiva do nome da variável começa com uma letra maiúscula.
- Todas as outras letras são minúsculas.

As declarações de variável a seguir obedecem a essa regra geral de atribuição de nomes:

```
Button loadFile;
int areaCode;
boolean playerSetNewHighScore;
```

Tipos de variável

Além do nome, uma declaração de variável deve incluir o tipo de informação que está sendo armazenado. O tipo pode ser qualquer um dos seguintes:

- Um dos tipos de dados básicos
- O nome de uma classe ou interface
- Um array

Você aprenderá a declarar e utilizar variáveis de array no Dia 5. Esta lição tratará dos outros tipos de variável.

Tipos de dados

Existem oito tipos básicos de variável para o armazenamento de inteiros, números em ponto flutuante, caracteres e valores booleanos. Freqüentemente, esses são chamados de *tipos primitivos*, pois são partes integrantes da linguagem Java e não objetos, o que os torna mais eficientes de usar. Esses tipos de dados possuem o mesmo tamanho e características, independentemente do sistema operacional e da plataforma em que você esteja, ao contrário de alguns tipos de dados de outras linguagens de programação.

Existem quatro tipos de dados que podem ser usados para armazenar inteiros. O que vai ser usado depende do tamanho do inteiro, conforme indicado na Tabela 3.1.

Tabela 3.1 Tipos inteiros.

Tipo	*Tamanho*	*Valores que podem ser armazenados*
byte	8 bits	-128 a 127
short	16 bits	-32.768 a 32.767
int	32 bits	-2.147.483.648 a 2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Todos esses tipos têm sinal, o que significa que eles podem conter tanto números positivos como negativos. O tipo usado para uma variável depende do intervalo de valores que ela poderia precisar conter. Nenhuma dessas variáveis inteiras pode armazenar seguramente um valor que seja grande ou pequeno demais para seu tipo designado; portanto, você deve tomar cuidado ao designar o tipo.

Outro tipo de valor que pode ser armazenado é o número em ponto flutuante, que tem o tipo float ou double. Os *números em ponto flutuante* representam os números com uma parte decimal. O tipo float deve ser suficiente para a maioria dos casos, pois ele pode manipular qualquer número de 1.4E-45 a 3,.4E+38. Caso contrário, o tipo double pode ser usado para números mais precisos, variando de 4,9E-324 a 1,7E+308.

O tipo char é usado para caracteres individuais, como letras, números, pontuação e outros símbolos.

O último dos oito tipos de dados básicos é o boolean. Conforme você aprendeu, os valores booleanos contêm true ou false em Java.

Todos esses tipos de variável são listados em letras minúsculas e você deve utilizá-los dessa forma nos programas. Existem classes com o mesmo nome de alguns desses tipos de dados, mas com utilização diferente de letras maiúsculas e minúsculas — por exemplo, Boolean e Char. Elas têm funcionalidade diferente

em um programa Java; portanto, você não pode usá-las indistintamente. Amanhã você verá como essas classes especiais são utilizadas.

Tipos de classe

Além dos oito tipos de dados básicos, uma variável pode ter uma classe como tipo, como nos exemplos a seguir:

```
String lastName = "Walsh";

Color hair;

Jabberwock firstMonster;
```

Quando uma variável tem uma classe como tipo, ela se refere a um objeto dessa classe ou de uma de suas subclasses.

O último dos exemplos da lista precedente, Jabberwock f i rstMonster;, cria uma variável chamada f i rstMonster que faz referência a um objeto Jabberwock.

Fazer referência a uma superclasse como um tipo de variável é útil quando a variável pode ser uma das diversas subclasses diferentes. Por exemplo, considere uma hierarquia de classe com uma superclasse Fruit e três subclasses: Apple, Pear e Strawberry. Se você criasse uma variável Fruit chamada favorite-Fruit, ela poderia ser usada para fazer referência a um objeto Apple, Pear ou Strawberry.

Declarar uma variável de tipo Object significa que ela pode conter qualquer objeto.

A linguagem Java não tem nada comparável à instrução typedef de C e C++. Para se declarar novos tipos em Java, uma nova classe é declarada e as variáveis podem usar essa classe como tipo.

Atribuindo valores às variáveis

Uma vez declarada uma variável, um valor pode ser atribuído a ela com o operador de atribuição — um sinal de igualdade (=). A seguir, estão exemplos de instruções de atribuição:

```
idCode = 8675309;

snappyDresser = false;
```

Comentários

Uma das maneiras mais importantes de melhorar a legibilidade de seu programa é usar comentários.

Comentários são informações incluídas em um programa estritamente para proveito dos seres humanos que estejam tentando descobrir o

que está acontecendo ali. O compilador Java ignora totalmente os comentários ao preparar uma versão executável de um arquivo-fonte Java.

Existem três tipos diferentes de comentários que podem ser usados em programas Java e você pode usar cada um deles como quiser.

O primeiro modo de incluir um comentário em um programa é precedendo-o com dois caracteres de barra normal (//). Tudo o que estiver das barras até o final da linha é considerado um comentário, como na instrução a seguir:

```
int creditHours = 3; // define as horas de crédito do curso
```

Nesse exemplo, tudo o que está depois de // até o final da linha é um comentário e será ignorado por um compilador Java.

Se você precisar de um comentário que ocupe mais de uma linha, pode iniciá-lo com o texto /* e terminá-lo com */. Tudo o que estiver dentro desses dois delimitadores será considerado um comentário, como no seguinte:

```
/* Este programa foi escrito tarde da noite, sob a
influência de medicamento anti-histamínico com validade expirada
e soda comum de supermercado. Eu não dou garantias
expressas ou implícitas de que ele funcione para
nenhum propósito útil, qualquer que seja ele. */
```

O último tipo de comentário é destinado a ser lido pelo computador e também por seres humanos. Se você iniciar um comentário com o texto /** (em vez de /*) e finalizá-lo com */, ele será interpretado como documentação oficial a respeito de como a classe e seus métodos públicos funcionam.

Esse tipo de comentário pode, então, ser lido por utilitários, como a ferramenta javadoc incluída no JDK. O programa javadoc utiliza comentários oficiais para criar um conjunto de páginas da Web que documentam o programa, sua hierarquia de classe e métodos.

Toda documentação oficial da biblioteca de classe Java é proveniente de comentários no estilo do j avadoc. Você pode ver a documentação da linguagem Java 2 na Web, na seguinte página:

```
http://java.sun.com/products/jdk/1.2/docs
```

Literais

Além das variáveis, você também usará uma literal em uma instrução Java.

NOVO TERMO

Literais são quaisquer números, texto ou outras informações que representem diretamente um valor.

Literal é um termo de programação que significa basicamente que o que você digita é o que obtém. A instrução de atribuição a seguir utiliza uma literal:

```
int year = 1998;
```

A literal é 1998, pois ela representa diretamente o valor inteiro 1998. Números, caracteres e strings, todos são exemplos de literais.

Embora o significado e a utilização de literais pareçam intuitivos na maioria das vezes, a linguagem Java possui alguns tipos especiais de literais que representam tipos diferentes de números, caracteres, strings e valores booleanos.

Literais numéricas

A linguagem Java possui várias literais inteiras. O número 4, por exemplo, é uma literal inteira do tipo de variável int. Ela também pode ser atribuída às variáveis byte e short, pois o número é suficientemente pequeno para caber nesses tipos de inteiro. Uma literal inteira maior do que um int pode conter, é considerada automaticamente do tipo long. Você também pode indicar que uma literal deve ser um inteiro 1ong incluindo a letra L (L ou l) no número. Por exemplo, a instrução a seguir armazena o valor 4 em um inteiro 1 ong:

```
long pennyTotal = 4L;
```

Para representar um número negativo como uma literal, coloque um sinal de subtração (-) antes dela, como em -45.

Nota

Os sistemas de numeração octal e hexadecimal são convenientes para muitas utilizações de programação avançada, mas de pouca utilidade para os iniciantes. Os *números* octais são baseados no sistema de numeração de base 8, o que significa que eles só podem representar os valores de 0 a 7 como um único dígito. O oitavo número em octal é 10 (ou 010, como uma literal Java).

Hexadecimal é um sistema de numeração de base 16 e pode representar 1 6 números como um único dígito. As letras de A a F representam os seis últimos dígitos; portanto, os 16 primeiros números são 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Para certas tarefas de programação, os sistemas octal e hexadecimal são mais convenientes do que o sistema decimal normal. Se você já usou HTML para definir a cor de fundo de uma página da Web, então já utilizou números hexadecimais.

Se você precisar usar um inteiro literal com numeração octal, coloque um 0 antes do número. Por exemplo, o número octal 777 seria a literal 0777. Os inteiros hexadecimais são usados como literais, colocando-se Ox antes do número, como em 0x12 ou OxFF.

As literais em ponto flutuante usam um caractere de ponto-final (.) para o ponto decimal, conforme seria de se esperar. A instrução a seguir usa uma literal para definir uma variável double: •

```
double myGPA = 2.25;
```

Todas as literais em ponto flutuante são consideradas do tipo de variável double, em vez de float. Para especificar uma literal float, inclua a letra F (F ou f) na literal, como no exemplo a seguir:

```
float piValue = 3.1415927F;
```

Você pode usar expoentes em literais em ponto flutuante, usando a letra e ou E, seguida do expoente, que pode ser um número negativo. As instruções a seguir utilizam notação exponencial:

```
double x = 12e22;
double y = 19E-95;
```

Literais booleanas

Os valores booleanos true e f alse também são literais. Esses são os únicos dois valores que você pode usar ao atribuir um valor a um tipo de variável bool ean ou ao usar um booleano em uma instrução de outras maneiras.

Se você já usou outras linguagens, como C, poderia esperar que o valor 1 fosse equivalente a true e 0 equivalente a false. Esse não é o caso em Java — você precisa usar os valores true ou false para representar valores booleanos. A instrução a seguir define uma variável booleana:

```
boolean toThineOwnSelf = true;
```

Observe que a literal true não está entre aspas. Se ela estivesse, o compilador Java assumiria que se trata de um string de caracteres.

Literais de caractere

As literais de caractere são expressas por um único caractere entre apóstrofos, como em 'a', '#' e '3[1]. Você pode estar familiarizado com o conjunto de caracteres ASCII, que possui 128 caracteres, incluindo letras, numerais, sinais de pontuação e outros caracteres úteis na computação. A linguagem Java oferece suporte a milhares de caracteres adicionais, através do padrão Unicode de 16 bits.

Algumas literais de caractere representam caracteres que não são prontamente imprimíveis ou acessíveis através de um teclado. A Tabela 3.2 lista os códigos especiais que podem representar esses caracteres especiais, bem como os elementos do conjunto de caracteres Unicode. A letra *d* nos códigos de escape octal, hexadecimal e Unicode representa um número ou um dígito hexadecimal (a-f ou A-F).

Tabela 3.2 Códigos de escape de caractere.

Escape	*Significado*
\n	Nova linha
\t	Tabulação
\b	Retrocesso
\r	Retorno de carro
\f	Avanço de formulário
\\	Barra invertida
\'	Apóstrofo
\"	Aspas
d	Octal
\xd	Hexadecimal
ud	Caractere Unicode

Os programadores de C e C++ devem observar que a linguagem Java não inclui códigos de caractere para \a (campainha) ou \v (tabulação vertical).

Literais de strina

O último tipo de literal que você pode usar em um programa Java representa strings de caracteres. Em Java, um string é um objeto e não um tipo de dado básico, e os strings não são armazenados em arrays, como acontece em linguagens como o C.

Como os objetos de string são objetos reais na linguagem Java, métodos estão disponíveis para combinar strings, modificá-los e determinar se dois strings possuem o mesmo valor.

As literais de string são compostas de uma série de caracteres entre aspas, como nas instruções a seguir:

```
String coAuthor = "Laura Lemay, killer of trees";
String password = "swordfish";
```

Os strings podem incluir os códigos de escape de caractere listados na Tabela 3.2, apresentada anteriormente, como se vê aqui:

```
String example = "Sócrates asked, VHemlock is poison?\"";
System.out.println("Bob Kemp\nOne on One Sports\n2 a.m. to 6 a.m.");
String title = "Sams Teach Yourself Java in a 3-Day Weekend\u2122"
```

No último exemplo, a seqüência de código Unicode \u2122 produz um símbolo ™ em sistemas que foram configurados para oferecer suporte a Unicode.

Cuidado

A maioria dos usuários de países de língua inglesa provavelmente não verá caracteres Unicode ao executarem programas Java. Embora a linguagem Java ofereça suporte à transmissão de caracteres Unicode, o sistema do usuário também deverá oferecê-lo para que os caracteres sejam apresentados. O suporte a Unicode proporciona um modo de codificar seus caracteres para sistemas que aceitam o padrão. Embora a linguagem Java 1.02 aceite apenas o subconjunto Latin do Unicode, as versões 1.1 e 2 oferecem suporte à exibição de qualquer caractere Unicode que possa ser representado por uma fonte hospedeira.
Para obter maiores informações sobre Unicode, visite o site da Web do Unicode Consortium, no endereço http://www.unicode.org/.

Embora as literais de string sejam usadas de maneira semelhante às outras literais de um programa, nos bastidores elas são tratadas de forma diferente.

Quando uma literal de string é usada, a linguagem Java armazena esse valor como um objeto Stri ng. Você não precisa criar explicitamente um novo objeto, como acontece ao se trabalhar com outros objetos; portanto, é tão fácil trabalhar com elas como com os tipos de dados básicos. Os strings são incomuns a esse respeito — nenhum dos tipos básicos é armazenado como um objeto quando utilizado. Você aprenderá mais sobre strings e a classe String hoje e amanhã.

Expressões e operadores

Uma *expressão* é uma instrução que produz um valor. Algumas das expressões mais comuns são as matemáticas, como no exemplo de código-fonte a seguir:

```
int x = 3;
int y = 4;
int z = X * y;
```

A última instrução desse exemplo é uma expressão. O operador de multiplicação * é usado para multiplicar os inteiros x e y, e a expressão produz o resultado da multiplicação. Esse resultado é armazenado no inteiro z.

O valor produzido por uma expressão é chamado de "valor de retorno", conforme você já aprendeu. Esse valor pode ser atribuído a uma variável e utilizado de muitas outras maneiras em seus programas Java.

Em Java, a maioria das expressões utiliza operadores como *.

NOVO TERMO

Operadores são símbolos especiais utilizados para funções matemáticas, alguns tipos de instruções de atribuição e comparações lógicas.

Aritmética

Existem cinco operadores usados para operações aritméticas básicas na linguagem Java. Eles estão apresentados na Tabela 3.3.

Tabela 3.3 Operadores aritméticos.

Operador	*Significado*	*Exemplo*
+	Adição	3 + 4
	Subtração	5 - 7
*	Multiplicação	5 * 5
/	Divisão	1 4 / 7
	Módulo	20 % 7

Cada operador exige dois operandos, um em cada lado do operador. O operador de subtração também pode ser usado para negar um único operando — o que é equivalente a multiplicar esse operando por - 1 .

É preciso estar ciente, ao se usar a divisão, dos tipos de números com que se está tratando. Se você armazenar uma operação de divisão em um inteiro, o resultado será arredondado para um número inteiro, pois o tipo de dado int não pode manipular números em ponto flutuante. Como exemplo, a expressão 3 1 / 9 resulta em 3, se for armazenada como um inteiro.

A divisão em módulo, que usa o operador %, produz o resto de uma operação de divisão. Usar 31 % 9 resulta 4, pois 31 dividido por 9 deixa um resto igual a 4.

Observe que a maioria das operações aritméticas envolvendo inteiros produz um int, independentemente do tipo original dos operandos. Se você estiver trabalhando com outros números, como números em ponto flutuante ou inteiros long, deve certificar-se de que os operandos tenham o mesmo tipo que está tentando obter.

A listagem 3.1 é um exemplo de aritmética simples na linguagem Java.

DIGITE **Listagem 3.1** O arquivo-fonte Amoebamath.java.

```
1: class AmoebaMath {
2:   public static void main (String arguments[ ]) {
3:    int x = 6;
4:     short y = 4;
5:     float a = .12f;
6:
7:     System.out.pn'ntln("You start with " + x + " pet amoebas.");
8:     System.out.println("\tTwo get married and their spouses move in.");
9:    x = x + 2;
10:    System.out.println("You now have " + x);
11:
12:    System.out.println("\tMitosis occurs, doubling the number of amoebas.");
13:   x = x * 2;
14:    System.out.println("You now have " + x);
15:
16:    System.out.println("\tThere's a fight. " + y + " amoebas move out.");
```

Listagem 3.1 Continuação

```
17:     x = x - y;
18:     System.out.println("You now have " + x);
19:
20:     System.out.println("\tParamecia attack! You lose one-third of the colony.");
21:     x = x - (x / 3);
22:     System.out.println("You end up with " + x + " pet amoebas.");
23:     System.out.println("Daily upkeep cost per amoeba: $" + a);
24:     System.out.println("Total daily cost: $" + (a * x));
25:   }
26: }
```

Se você executar esse aplicativo Java, ele produzirá a seguinte saída:

SAÍDA

```
You start with 6 pet amoebas.
        Two get married and their spouses move in.
You now have 8
       Mitosis occurs, doubling the number of amoebas.
You now have 16
       There's a fight. 4 amoebas move out.
You now have 12
        Paramecia attack! You lose one-third of the colony.
You end up with 8 pet amoebas.
Daily upkeep cost per amoeba: $0.12
Total daily cost: $0.96
```

Nesse simples aplicativo Java, três variáveis são criadas com valores iniciais nas linhas 3 a 5: o x inteiro, o y inteiro short e o número em ponto flutuante a. Como o tipo padrão para números em ponto flutuante é double, um f é anexado à literal . 12 para indicar que isso é do tipo float.

O restante do programa usa operadores aritméticos para controlar a população de uma colônia de amebas. (Nenhuma ameba foi ferida durante a produção deste capítulo.)

Esse programa também faz uso de System.out.println( ) em várias instruções. O método System.out.println( ) é usado em um aplicativo para apresentar strings e outras informações no dispositivo de saída padrão, que normalmente é a tela.

System.out.println( ) recebe um argumento dentro de seus parênteses: um string. Para apresentar mais de uma variável ou literal como argumento de pri ntl n ( ), você pode utilizar o operador + para combinar esses elementos em um único string.

Você aprenderá mais sobre essa utilização do operador + posteriormente, ainda hoje.

Mais informações sobre a atribuição

A atribuição de um valor a uma variável é uma expressão, pois isso produz um valor. Devido a essa característica, você pode encadear instruções de atribuição da seguinte maneira:

```
x = y = z = 7;
```

Nessa instrução, todas as três variáveis acabam com o valor 7.

O lado direito de uma expressão de atribuição é sempre calculado antes que a atribuição ocorra. Isso torna possível usar uma instrução de atribuição como no exemplo de código a seguir:

```
int x = 5;
x = x + 2;
```

Na expressão x = x + 2, o que primeiro acontece é o cálculo de x + 2. O resultado desse cálculo, 7, é então atribuído a x.

O uso de uma expressão para mudar o valor de uma variável é uma tarefa extremamente comum em programação. Existem vários operadores utilizados estritamente nesses casos.

A Tabela 3.4 mostra esses operadores de atribuição e as expressões funcionalmente equivalentes.

Tabela 3.4 Operadores de atribuição.

Expressão	*Significado*
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`

Esses operadores de atribuição abreviados são funcionalmente equivalentes às instruções de atribuição mais longas, as quais substituem. Entretanto, se um dos lados de sua instrução de atribuição fizer parte de uma expressão complexa, haverá casos em que os operadores não serão equivalentes. Por exemplo, se x é igual a 20 e y é igual a 5, as duas instruções a seguir não produzirão o mesmo valor:

```
x = x / y + 5;
x /= y + 5;
```

Quando estiver em dúvida, simplifique uma expressão usando várias instruções de atribuição e não utilize os operadores abreviados.

Incrementando e decrementando

Outra tarefa comum é somar ou subtrair 1 de uma variável inteira. Existem operadores especiais para essas expressões, que são chamadas operações de incremento e decremento.

NOVO TERMO *Incrementar* uma variável significa somar 1 ao seu valor. E *decrementá-la* significa subtrair 1 de seu valor.

O operador de incremento é ++ e o operador de decremento é --. Esses operadores são colocados imediatamente antes ou depois de um nome de variável, como no exemplo de código a seguir:

```
int x = 7;
x = x++;
```

Nesse exemplo, a instrução x = x++ incrementa a variável x de 7 para 8.

Esses operadores de incremento e decremento podem ser colocados antes ou depois de um nome de variável e isso afeta o valor das expressões que os envolvem.

NOVO TERMO Os operadores de incremento e decremento são chamados operadores *prefixados* se estiverem listados antes de um nome de variável, e operadores *posfixados*, se estiverem listados após um nome.

Em uma expressão simples, como standards--;, usar um operador prefixado ou posfixado não altera o resultado. Contudo, quando as operações de incremento e decremento fazem parte de uma expressão maior, a escolha entre operadores prefixados e posfixados é importante.

Considere as duas expressões a seguir:

```
int x, y, z;
x = 42;
y = x++;
z = ++x;
```

Essas duas expressões produzem resultados muito distintos, por causa da diferença entre as operações prefixadas e posfixadas. Quando você usa operadores posfixados, como em y = x++, y recebe o valor de x antes que ele seja incrementado em 1. Ao se usar operadores prefixados, como em z = ++x, x é incrementado em 1 antes que o valor seja atribuído a z. O resultado final desse exemplo é que y é igual a 42, z é igual a 44 e x é igual a 44.

Se você ainda estiver com problemas para perceber isso, eis o exemplo novamente, com comentários descrevendo cada etapa:

```
int x, y, z;   // x, y e z são declaradas
x = 42;        // x recebe o valor 42
y = x++;       // y recebe o valor de x (42), antes que ele seja incrementado
               // e x é então incrementado para 43
z = ++x;       // x é incrementado para 44 e z recebe o valor de x
```

Assim como os operadores abreviados, os operadores de incremento e decremento podem produzir resultados inesperados, quando usados em expressões extremamente complexas. O conceito da "atribuição de x para y, antes que x seja incrementado" não é precisamente correto, pois a linguagem Java avalia tudo o que estiver no lado direito de uma expressão, antes de atribuir seu valor no lado esquerdo. A linguagem Java armazena alguns valores antes de tratar de uma expressão, para fazer a notação posfixa funcionar do modo como foi descrito nesta seção. Quando você não estiver obtendo os resultados esperados de uma expressão complexa que inclui operadores prefixados e posfixados, tente dividir a expressão em várias instruções, para simplificá-la.

Comparações

A linguagem Java possui vários operadores que são usados ao se fazer comparações entre variáveis, variáveis e literais ou outros tipos de informações em um programa.

Esses operadores são usados em expressões que retornam os valores booleanos true ou fal se, dependendo da comparação que está sendo feita ser verdadeira ou não. A Tabela 3.5 mostra os operadores de comparação.

Tabela 3.5 Operadores de comparação.

Operador	*Significado*	*Exemplo*
	Igual	x == 3
!=	Diferente	x != 3
<	Menor que	x < 3
>	Maior que	x > 3
<=	Menor ou igual a	x <= 3
>=	Maior ou igual a	x >= 3

O exemplo a seguir mostra um operador de comparação em uso:

```
boolean hip;
int age = 31;
hip = age < 25;
```

A expressão age < 25 produz um resultado true ou false, dependendo do valor do inteiro age. Como age é 31 nesse exemplo (que não é menor do que 25), hi p recebe o valor booleano false.

Operadores lógicos

As expressões que resultam em valores booleanos, como as operações de comparação, podem ser combinadas para formar expressões mais complexas. Isso é realizado por operadores lógicos. Esses operadores são usados para as combinações lógicas AND, OR, XOR e NOT lógico.

No caso de combinações de AND, são usados os operadores lógicos & ou &&. Quando duas expressões booleanas são vinculadas pelos operadores & ou &&, a expressão combinada retornará o valor true apenas se as duas expressões booleanas forem verdadeiras.

Considere o exemplo a seguir, extraído diretamente do filme *Ensina-me a viver:*

```
boolean unusual = (age < 21) & (girlfriendAge > 78);
```

Essa expressão combina duas expressões de comparação: age < 21 e girlfriendAge > 78. Se essas duas expressões forem verdadeiras, o valor true será atribuído à variável unusual. Em qualquer outra circunstância, o valor false será atribuído a unusual.

A diferença entre & e && reside no trabalho realizado pela linguagem Java na expressão combinada. Se for usado &, as expressões dos dois lados desse símbolo serão avaliadas, independentemente do que sejam. Se for usado && e o lado esquerdo desse símbolo for false, a expressão do lado direito nunca será avaliada.

No caso de combinações de OR, os operadores lógicos ! ou !! são usados. Essas expressões combinadas retornam o valor true se uma das expressões booleanas for verdadeira.

Considere o exemplo inspirado em *Ensina-me a viver:*

```
boolean unusual = (grimThoughts > 10) !! (girlfriendAge > 78);
```

Essa expressão combina duas expressões de comparação: grimThoughts > 10 e girlfriendAge > 78. Se uma das expressões for verdadeira, o valor true será atribuído à variável unusual. Somente se essas duas expressões forem false é que o valor false será atribuído a unusual.

Observe o uso de !! em vez de !. Devido a essa utilização, se grimThoughts > 10 for true, unusual será definido como true e a segunda expressão nunca será avaliada.

A combinação de XOR possui um operador lógico ^. Esse operador resulta em um valor true somente se as duas expressões booleanas que ele combina possuírem valores opostos. Se as duas forem true ou as duas forem false, o operador produzirá o valor false.

A combinação de NOT usa o operador lógico !, seguido de uma única expressão. Ele inverte o valor de uma expressão booleana da mesma maneira que o sinal de subtração inverte o sinal positivo ou negativo em um número.

Por exemplo, se age < 30 retornar um valor true, ! (age < 30) retornará o valor false.

Esses operadores lógicos podem parecer completamente ilógicos, quando encontrados pela primeira vez. Você terá muitas oportunidades de trabalhar com eles nos capítulos subseqüentes, especialmente no Dia 5.

Precedência de operador

Quando mais de um operador é usado em uma expressão, a linguagem Java tem uma precedência estabelecida para determinar a ordem em que os operadores são avaliados. Em muitos casos, essa precedência determina o valor global da expressão.

Por exemplo, considere a seguinte expressão:

```
y = 6 + 4 / 2;
```

A variável y recebe o valor 5 ou o valor 8, dependendo da operação aritmética executada primeiro. Se a expressão 6 + 4 vier primeiro, y terá o valor 5. Caso contrário, y será igual a 8.

Em geral, a ordem do primeiro para o último é a seguinte:

- Operações de incremento e decremento
- Operações aritméticas
- Comparações
- Operações lógicas
- Expressões de atribuição

Se duas operações possuem a mesma precedência, a que está à esquerda da expressão é realizada antes da que está à direita. A Tabela 3.7 mostra a precedência específica dos vários operadores da linguagem Java. Os operadores que estão mais acima na tabela são avaliados primeiro.

Tabela 3.7 Precedência de operador.

Operador	Notas
• [ ] ( )	Parênteses (( )) são usados para agrupar expressões; o ponto-final (.) é usado para o acesso a métodos e variáveis dentro de objetos e classes (o que será discutido amanhã); colchetes são usados para arrays. (Esse operador está discutido posteriormente nesta semana.)
++ - ! - instanceof	O operador instanceof retorna true ou false com base no fato de o objeto ser uma instância da classe nomeada ou de qualquer uma das subclasses dessa classe (o que será discutido amanhã).
new (type)expression	O operador new é usado para a criação de novas instâncias de classes; ( ) nesse caso serve para fazer coerção do valor para outro tipo. (Você aprenderá sobre ambos amanhã.)
* / %	Multiplicação, divisão, módulo
+ -	Adição, subtração.
« » >»	Deslocamento no bit à esquerda e à direita.

Tabela 3.7 Continuação

Operador	Notas
< > <= >=	Testes de comparação relacionai.
= = ! =	Igualdade.
&	AND
^	XOR
|	OR
&&	AND lógico
||	OR lógico
? :	Abreviação de if...then...else (discutido no Dia 5).
= += -= *= /= %= ^=	Várias atribuições.
& =|= «= »= >» =	Mais atribuições.

Voltando à expressão y = 6 + 4 / 2 , a Tabela 3.7 mostra que a divisão é avaliada antes da adição; portanto, o valor de y será 8.

Para mudar a ordem em que as expressões são avaliadas, coloque entre parênteses as expressões que devem ser avaliadas primeiro. Você pode aninhar um conjunto de parênteses dentro de outro para garantir que as expressões sejam avaliadas na ordem desejada — a expressão entre parênteses mais interna é avaliada primeiro.

A expressão a seguir resulta no valor 5:

```
y = (6 + 4) / 2
```

O valor 5 é o resultado, pois 6 + 4 é calculado antes que o resultado (10) seja dividido por 2.

Os parênteses também podem ser úteis para melhorar a legibilidade de uma expressão. Se a precedência de uma expressão não for imediatamente clara para você, incluir parênteses para impor a precedência desejada pode tornar a instrução mais fácil de entender.

Aritmética de strina

Conforme dito anteriormente, o operador + possui uma vida dupla fora do mundo da matemática. Ele pode ser usado para concatenar dois ou mais strings.

NOVO TERMO *Concatenar* significa vincular duas coisas. Por motivos desconhecidos, esse foi o verbo escolhido para se descrever o ato de combinar dois strings — em vez de grudar, colar, anexar, combinar, vincular e unir.

Em vários exemplos, você viu instruções semelhantes à seguinte:

```
String firstName = "Raymond";
System.out.println("Everybody loves " + fisrtName);
```

Essas duas linhas resultam na exibição do seguinte texto:

```
Everybody loves Raymond
```

O operador + combina strings, outros objetos e variáveis para formar um único string. No exemplo anterior, a literal Everybody loves é concatenada ao valor do objeto String, firstName.

Trabalhar com o operador de concatenação é mais fácil em Java, graças à maneira pela qual a linguagem pode manipular qualquer tipo de variável e valor de objeto, como se fosse um string. Se uma parte de uma operação de concatenação fórum String ou literal String, todos os elementos da operação serão tratados como se fossem strings. Por exemplo:


```
System.out.println(4 + " score and " + 7 + " years ago.");
```

Isso produz o texto de saída 4 score and 7 years ago., como se as literais inteiras 4 e 7 fossem strings.

Também existe um operador += abreviado para incluir algo no final de um string. Considere, por exemplo, a expressão a seguir:

```
myName += " Jr.";
```

Essa expressão é equivalente ao seguinte:

```
myName = myName + " Jr.";
```

Nesse exemplo, muda-se o valor de myName (que poderia ser Efrem Zimbalist), incluindo-se Jr. no final (Efrem Zimbalist Jr.).

Resumo

Qualquer pessoa que abra um conjunto de bonecas Matryoska fica um pouco desapontada ao chegar à menor boneca do grupo. De preferência, os avanços na microengenharia deverão permitir que os artesãos russos criem bonecas cada vez menores, até que alguém atinja o limite subatômico e seja declarado o vencedor.

Hoje, você chegou à menor boneca do Java, mas isso não deve ser decepcionante. O uso de instruções e expressões permite começar a criar métodos eficientes, que tornem os objetos e classes os mais eficientes possíveis.

Hoje, você aprendeu sobre a criação de variáveis e a atribuição de valores a elas, a usar literais para representar valores numéricos, de caractere, e string e a trabalhar com operadores. Amanhã, você utilizará esses conhecimentos, ao desenvolver objetos para programas em Java.

Para resumir o material de hoje, a Tabela 3.8 lista os operadores sobre os quais você aprendeu. Examine-os cuidadosamente.

Tabela 3.8 Resumo dos operadores.

Operador	*Significado*
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a
==	Igual
!=	Diferente
&&	AND lógico
||	OR lógico
!	NOT lógico
&	AND
|	OR
^	XOR
=	Atribuição
++	Incremento
--	Decremento
+=	Somar e atribuir
-=	Subtrair e atribuir
*=	Multiplicar e atribuir
/=	Dividir e atribuir
%=	Módulo e atribuição

Perauntas e respostas

P **O que acontece se você atribuir a uma variável um valor inteiro que é grande demais para que ela contenha?**

R Logicamente, você poderia pensar que a variável é convertida para o próximo tipo inteiro maior, mas não é isso que acontece. Em vez disso, ocorre um *transbordamento* — uma situação em que o número passa de um tamanho extremo para o outro. Um exemplo de transbordamento seria uma variável byte que passa de

127 (valor aceitável) para 128 (inaceitável). Ela passaria para o valor mais baixo aceitável, que é -128, e começaria a contar daí para cima. O transbordamento não é algo com que você consegue lidar prontamente em um programa; portanto, você deve garantir bastante espaço às suas variáveis, no tipo de dado escolhido.

Por que a linguagem Java possui todos esses operadores abreviados para aritmética e atribuição? Assim eles ficam difíceis de ler.

A sintaxe da linguagem Java tem por base o C + +, que tem por base o C (mais comportamento de boneca russa). C é uma linguagem especialista, que avalia o poder de programação pela legibilidade e os operadores abreviados são um dos legados dessa prioridade de projeto. Seu uso em um programa não é obrigatório, pois estão disponíveis substitutos eficientes, para que você possa evitá-los em seus próprios programas, caso prefira.

SEMANA 1

Dia 4

Lições sobre objetos

Há dois dias, a programação orientada a objetos era comparada à cerveja, pois ela é um gosto adquirido que pode causar desorientação, náusea e uma visão distorcida do mundo.

O capítulo de hoje proporciona uma renovação. Você se familiariza novamente com esse tipo de programação, enquanto trabalha com objetos em Java.

A maior parte das coisas que você faz na linguagem é feita com objetos. Você cria objetos, modifica-os, move-os por toda parte, altera suas variáveis, chama seus métodos e combina-os com outros objetos. Você desenvolve classes, cria objetos fora dessas classes e utiliza-os com outras classes e objetos.

Hoje, você vai trabalhar extensivamente com objetos. Os seguintes tópicos são abordados:

- Criação de objetos (também chamados de *instâncias)*
- Teste e modificação de variáveis de classe e instância nesses objetos
- Chamada de métodos de um objeto
- Conversão de objetos e outros tipos de dados de uma classe para outra

Criando novos objetos

Ao escrever um programa Java, você define um conjunto de classes. Conforme você aprendeu no Dia 2, as classes são modelos de objetos; de maneira geral,

você simplesmente usa a classe para criar instâncias e depois trabalha com essas instâncias. Nesta seção, portanto, você aprenderá a criar um novo objeto a partir de uma classe dada.

Lembra-se dos strings de ontem? Você aprendeu que usar uma *literal de string* (uma série de caracteres colocados entre aspas) cria uma nova instância da classe String com o valor desse string.

A classe String é incomum a esse respeito. Embora seja uma classe, há uma maneira fácil de criar instâncias dessa classe usando uma literal. As outras classes não possuem esse atalho; para criar instâncias dessas classes, você precisa fazer isso explicitamente, usando o operador new.

E quanto as literais para números e caracteres — elas não criam objetos também? Na verdade, não. Os tipos de dados primitivos para números e caracteres criam números e caracteres, mas, por questões de eficiência, eles na verdade não são objetos. Você pode colocar *wrappers* de objeto em torno deles, se precisar tratá-los como objetos (o que você aprenderá a fazer posteriormente neste livro).

Usando o operador new

Para criar um novo objeto, você usa o operador new com o nome da classe da qual deseja criar uma instância, seguida de parênteses. Por exemplo:

```
String teamName = new String( );
Random randlnfo = new Random( );
Jabberwock j = new Jabberwock( );
```

Os parênteses são importantes; não se esqueça deles. Os parênteses podem estar vazios, caso em que o objeto básico mais simples é criado, ou podem conter argumentos que determinam os valores iniciais de variáveis de instância ou outras qualidades iniciais desse objeto.

Os exemplos a seguir mostram objetos sendo criados com argumentos:

```
GregorianCalendar date = new GregorianCalendar(64, 6, 6, 7, 30)

Point pt = new Point(0,0);
```

O número e o tipo dos argumentos que você usa dentro dos parênteses com o operador new são definidos pela própria classe, usando um método especial chamado *construtor*. (Você aprenderá mais sobre construtores posteriormente, ainda hoje.) Se você tentar criar uma nova instância de uma classe com o número ou tipo errado de argumentos (ou se você não fornecer nenhum argumento e ela precisa de algum), obterá um erro quando tentar compilar seu programa Java.

Eis um exemplo de criação de vários tipos diferentes de objetos usando diferentes números e tipos de argumentos: a classe Random, parte do pacote java.útil, cria objetos que são usados para gerar números aleatórios em um

programa. Esses objetos são chamados *geradores de números aleatórios* e eles variam de 0,0 a 1,0, em valores decimais. Os números aleatórios são úteis em jogos e outros programas em que é necessário um elemento imprevisível.

Na verdade, o objeto Random não gera números aleatoriamente. Em vez disso, ele extrai um número de uma seqüência extremamente longa. Isso é chamado de *geração de números pseudo-aleatória* e é usado em muitas linguagens de programação diferentes. Para extrair um número diferente da seqüência de números aleatórios, o objeto Random ( ) deve receber um valor-semente. Essa semente pode ser enviada quando o objeto for criado.

A listagem 4.1 é um programa Java que cria objetos Random usando o operador new de duas maneiras diferentes.

DIGITE

Listagem 4.1 O texto completo de RANDOMNUMBERS.JAVA.

```
1: import java.útil.Random;
2:
3: class RandomNumbers {
4:
5:     public static void main(String arguments[ ]) {
6:         Random r1, r2;
7:
8:     r1     = new Random( );
9:        System.out.println("Random value 1: " + rl.nextDouble( ));
10:
11:       r2 = new Random(8675309);
12:       System.out.println("Random value 2: " + r2.nextDouble( ));
13:   }
14:}
```

Quando você compilar e executar o programa, a saída deverá ser semelhante à seguinte:

SAÍDA

```
Random value 1: 0.3125961341023068
Random value 2: 0.754788115099576
```

Nesse exemplo, dois diferentes objetos Random são criados, usando diferentes argumentos para a classe listada após new. A primeira instância (linha 8) utiliza new Random( ) sem argumentos, o que cria um objeto Random semeado com a hora atual. O valor de sua saída para a primeira linha dependerá da hora em que você executar o programa, pois o valor aleatório reflete a mudança na hora.

Por esse motivo, a maioria dos objetos Random usa, por padrão, a hora como semente.

A chamada do método nextDouble( ) do objeto Random( ) nas linhas 9 e 12 retorna o próximo número da seqüência pseudo-aleatória de números.

O segundo objeto Random desse exemplo tem um argumento inteiro quando é construído na linha 11. A segunda linha da saída deve ser a mesma cada vez em que ela é executada, mostrando o valor 0.754788115099576. Quando você usa uma literal como semente, a seqüência de números aleatórios é sempre a mesma. Isso pode ser útil para fins de teste.

Você pode estar com problemas para descobrir por que um número decimal long, como 0.754788115099576, poderia ser usado para gerar um aleatório. Se você multiplicar esse valor aleatório por um inteiro, o produto será um número aleatório entre 0 e esse inteiro. Por exemplo, as instruções a seguir multiplicam um número aleatório por 1 2 e salvam o produto como um inteiro:

```
Random r1 = new Random( );
int number = (int)(r1.nextDouble( ) * 12);
```

O inteiro em number será um número aleatório de 0 a 11.

O que faz o operador new

Várias coisas acontecem quando você usa o operador new. A nova instância da classe dada é criada, memória é alocada para ela e um método especial, definido na classe dada, é chamado. Esse método especial é chamado de construtor.

Construtores são métodos especiais para a criação e inicialização de novas instâncias de classes. Os construtores inicializam o novo objeto e suas variáveis, criam todos os outros objetos de que esse objeto necessita e realizam todas as outras operações que ele precisa para ser inicializado.

Cada uma das várias definições de construtor em uma classe tem um número ou tipo de argumento diferente. Quando você usa o operador new, pode especificar diferentes argumentos na respectiva lista, e o construtor correto para esses argumentos será chamado. As várias definições de construtor foram o que permitiram à classe Random( ) do exemplo anterior realizar diferentes coisas com diferentes versões da instrução new. Quando você cria suas próprias classes, pode definir quantos construtores precisar para implementar o comportamento da classe.

Uma nota sobre gerenciamento de memória

Se você está familiarizado com outras linguagens de programação orientada a objetos, pode estar se perguntando se a instrução new possui um oposto que destrua um objeto, quando ele não é mais necessário.

O gerenciamento de memória na linguagem Java é dinâmico e automático. Quando você cria um novo objeto, a linguagem Java aloca automa-

ticamente a área de memória correta para ele. Você não precisa alocar nenhuma memória para objetos explicitamente. A linguagem Java faz isso para você.

Como o gerenciamento de memória da linguagem Java é automático, você não precisa desalocar a memória que o objeto utiliza, quando acabar de usá-lo. Quando você acabar de usar o objeto, ele não terá mais nenhuma referência válida (ele não será atribuído a nenhuma variável que você ainda esteja usando nem armazenado em nenhum array). A linguagem Java tem um coletor de lixo que procura objetos não utilizados e recupera a memória que eles estão usando. Você não precisa liberar a memória explicitamente — você tem apenas de certificar-se de que não está prendendo um objeto do qual deseja se desfazer.

Acessando e definindo variáveis de classe e instância

Neste ponto, você poderia criar seu próprio objeto com variáveis de classe e instância definidas nele — mas como é que você trabalha com essas variáveis? E fácil! As variáveis de classe e instância se comportam de forma igual às variáveis locais, sobre as quais você aprendeu ontem. Você apenas faz referência a elas em seu código de um modo ligeiramente diferente do que acontece com as variáveis normais.

Obtendo valores

Para obter o valor de uma variável de instância, você usa a notação de ponto. Com essa notação, o nome de uma variável de instância ou de classe possui duas partes: o objeto, no lado esquerdo do ponto, e a variável, no lado direito.

A *notação de ponto* é um modo de se fazer referência às variáveis de instância e aos métodos de um objeto, usando um operador de ponto (.).

Por exemplo, se você tem um objeto atribuído à variável myCustomer e esse objeto possui uma variável chamada orderTotal, então a referência ao valor dessa variável é feita como segue:

```
myCustomer.orderTotal;
```

Essa forma de acessar variáveis é uma expressão (isto é, ela retorna um valor) e os dois lados do ponto também são expressões. Isso significa que você pode aninhar o acesso à variável de instância. Se a própria variável de instância orderTotal contivesse um objeto e esse objeto possuísse sua própria variável de instância, chamada 1 ayaway, você poderia fazer referência a ela como segue:

```
myCustomer.orderTotal.layaway;
```

As expressões de ponto são avaliadas da esquerda para a direita; portanto, você começa com a variável orderTotal de myCustomer, que aponta para outro objeto com a variável 1 ayaway. Você acaba com o valor dessa variável 1 ayaway.

Alterando valores

A atribuição de um valor a essa variável é igualmente fácil — basta anexar um operador de atribuição no lado direito da expressão:

```
myCustomer.orderTotal.layaway = true;
```

Esse exemplo define o valor da variável 1 ayaway como true.

A listagem 4.2 é um exemplo de programa que testa e modifica as variáveis de instância em um objeto Point. Poi nt faz parte do pacote java. awt e refere-se a um ponto de coordenadas com valores x e y.

DIGITE

Listagem 4.2 O texto completo de SETPOINTS.JAVA.

```
1: import java.awt.Point;
2:
3: class SetPoints {
4:
5: public static void main(String arguments[ ]) {
6:     Point location = new Point(4, 13);
7:
8:     System.out.println("Starting location:");
9:     System.out.println("X equals " + location.x);
10:   System.out.println("Y equals " + location.y);
11:
12:    System.out.println("\nMoving to (7, 6)");
13:    location.x = 7;
14:    location.y = 6;
15:
16:    System.out.println("\nEnding location:");
17:    System.out.println("X equals " + location.x);
18:    System.out.println("Y equals " + location.y);
19:   }
20: }
```

Quando você executar esse aplicativo, a saída deverá ser a seguinte:

SAÍDA

```
Starting location:
X equals 4
Y equals 13

Moving to (7, 6)

Ending location:
X equals 7
Y equals 6
```

Nesse exemplo, primeiramente você cria uma instância de Point onde x é igual a 4 e y é igual a 13 (linha 6). As linhas 9 e 10 apresentam esses valores individuais usando a notação de ponto. As linhas 13 e 14 alteram o valor de x

para 7 e o de y para 6, respectivamente. Finalmente, as linhas 17 e 18 apresentam os valores de x e y novamente, para mostrar como eles foram alterados.

Variáveis de classe

As variáveis de classe, conforme você já aprendeu, são aquelas definidas e armazenadas na própria classe. Seus valores se aplicam à classe e a todas as suas instâncias.

Com as variáveis de instância, cada nova instância da classe recebe uma nova cópia das variáveis de instância que a classe define. Cada instância, então, pode alterar os valores dessas variáveis sem afetar as outras instâncias. Com as variáveis de classe, existe apenas uma cópia dessa variável. Mudar o valor dessa variável o altera para todas as instâncias dessa classe.

Você define as variáveis de classe incluindo a palavra-chave `static` antes da própria variável. Por exemplo, tome a seguinte definição de classe parcial:

```
class FamilyMember {
    static String surname = "Igwebuike";
    String name;
    int age;
}
```

Cada uma das instâncias da classe FamilyMember (membro da família) possuí seus próprios valores para name e age (nome e idade) — mas a variável de classe surname (sobrenome) tem apenas um valor para todos os membros da família: "Igwebuike". Altere o valor de surname e todas as instâncias de FamilyMember serão afetadas.

A denominação dessas variáveis de `static` se refere a um dos significados da palavra *estático:* fixo em um lugar. Se uma classe possui uma variável **static,** todo objeto dessa classe terá o mesmo valor para essa variável.

Para acessar variáveis de classe, você usa a mesma notação de ponto utilizada com variáveis de instância. Para recuperar ou alterar o valor da variável de classe, você pode usar a instância ou o nome da classe no lado esquerdo do ponto. As duas linhas de saída desse exemplo apresentam o mesmo valor:

```
FamilyMember dad = new FamilyMemberf );
System.out.println("Family's surname is: " + dad.surname);
System.out.println("Family's surname is: " + FamilyMember.surname);
```

Como você pode usar uma instância para alterar o valor de uma variável de classe, é fácil fazer confusão com as variáveis de classe e de onde seus valores estão vindo — lembre-se de que o valor de uma variável de classe afeta todas as suas instâncias. Por esse motivo, é uma boa idéia usar o nome da classe, quando você fizer referência a esse tipo de variável. Isso torna seu código mais fácil de ler e os resultados estranhos mais fáceis de depurar.

Chamando métodos

Chamar um método em um objeto é semelhante a fazer referência a suas variáveis de instância: a notação de ponto é usada. O objeto cujo método você está chamando fica no lado esquerdo do ponto e o nome do método e seus argumentos ficam no lado direito. Por exemplo:

```
myCustonier.addToOrder(itemNumber, price, quantity);
```

Observe que todos os métodos devem terparênteses depois deles, mesmo que o método não receba argumentos. Por exemplo:

```
myCustomer.cancelA110rders( );
```

Se o método chamado retorna um objeto que possui métodos próprios, você pode aninhar métodos como faria com variáveis. O próximo exemplo chama o método tal kToManager, que é definido no objeto retornado pelo método cancel Al 1 Orders que, por sua vez, foi definido em myCustomer:

```
myCustomer.cancelA110rders( ).talkToManager( );
```

Você também pode combinar chamadas de método aninhadas e referências de variável de instância. No próximo exemplo, o método putOnLayaway ( ) é definido no objeto armazenado pela variável de instância orderTotal, que faz parte, ela própria, do objeto myCustomer:

```
myCustomer.orderTotal. putOnLayaway(itemNumber, price, quantity);
```

System.out.println( ), o método que você utilizou em todos os exemplos de programa para apresentar informações, é um exemplo de aninhamento de variáveis e métodos. A classe System, parte do pacote j ava. 1 ang, descreve o comportamento específico do sistema em que a linguagem Java está sendo executada. System.out é uma variável de classe que contém uma instância da classe PrintStream. Esse objeto PrintStream representa a saída padrão do sistema, que normalmente é a tela, mas pode ser redirecionada para um monitor ou arquivo de disco. Os objetos Pri ntStream possuem um método pri ntl n ( ) que envia um stnng para esse fluxo de saída.

A listagem 4.3 mostra um exemplo de chamada de alguns métodos definidos na classe Stnng. Os strings incluem métodos para testes e modificação de strings, semelhantes ao que você esperaria de uma biblioteca de string em outras linguagens.

DIGITE **Listagem 4.3** O texto completo de Checkstring.java.

```
1: class CheckString {
2:
3:     public static void main(String arguments[ ]) {
4:         String str = "In my next life, I will believe in reincarnation";
5:         System.out.println("The string is: " + str);
```

Listagem 4.3 Continuação

```
6:          System.out.println("Length of this string: "
7:              + str.1ength( ));
8:          System.out.println("The character at position 7: "
9:              + str.charAt(7));
10:         System.out.println("The substring from 24 to 31: "
11:             + str.substring(24, 31));
12:         System.out.println("The index of the character x: "
13:             + str.indexOf('x'));
14:         System.out.println("The index of the beginning of the "
15:             + "substring \"will\": " + str.indexOf("will"));
16:         System.out.println("The string in upper case: "
17:             + str.tol)pperCase( ));
18:     }
19: }
```


O seguinte será exibido no dispositivo de saída padrão de seu sistema, quando você executar o programa:

SAÍDA

```
The string is: In my next life, I will believe in reincarnation
Length of this string: 48
The character at position 7: e
The substring from 24 to 31: believe
The index of the character x: 8
The index of the beginning of the substring "will": 19
The string in upper case: IN MY NEXT LIFE, I WILL BELIEVE IN REINCARNATION
```

Na linha 4, você cria uma nova instância de Stri ng usando uma literal de string. (Esse modo é mais fácil do que usar new e colocar os caracteres individualmente.) O restante do programa simplesmente chama diferentes métodos de string para realizar diferentes operações nesse string:

- A linha 5 imprime o valor do string que você criou na linha 4: "In my next life, I wi11 believe in reincarnation".
- A linha 7 chama o método length( ) no novo objeto String. Esse string tem 48 caracteres.
- A linha 9 chama o método charAt( ), que retorna o caractere da posição dada no string. Observe que as posições de string começam em 0 e não em 1; portanto, o caractere da posição 7 é e.
- A linha 11 chama o método substring ( ), que recebe dois inteiros que indicam um intervalo e retorna o substring com esses pontos inicial e final. O método substri ng ( ) também pode ser chamado com apenas um argumento, o que retorna o substring a partir dessa posição até o final do string.

- A linha 13 chama o método index0f( ), que retorna a posição da primeira instância do caractere dado (aqui, ' x'). As literais de caractere são colocadas entre apóstrofos — se fossem colocadas aspas no x da linha 13, a literal seria considerada um String.
- Alinha 15 mostra uma utilização diferente do método index0f( ),que recebe um argumento de string e retorna o índice do início desse string.
- A linha 17 usa o método toUpperCase( ) para retornar uma cópia do string com todas as letras maiúsculas.

Métodos de classe

Os métodos de classe, assim como as variáveis de classe, aplicam-se à classe como um todo e não às suas instâncias. Os métodos de classe são usados normalmente para métodos gerais de utilitário que poderiam não operar diretamente sobre uma instância dessa classe, mas se encaixam conceitualmente nessa classe. Por exemplo, a classe String contém um método de classe chamado value0f( ), que pode receber um de muitos tipos diferentes de argumentos (inteiros, booleanos, outros objetos etc). O método value0f( ) retorna, então, uma nova instância de String, contendo o valor de string do argumento. Esse método não opera diretamente sobre uma instância existente de String, mas obter um string de outro objeto ou tipo de dado é definitivamente uma operação do tipo Stri ng, e faz sentido defini-la na classe Stri ng.

Os métodos de classe também podem ser úteis na reunião de métodos genéricos em um único lugar (a classe). Por exemplo, a classe Math, definida no pacote java.lang, contém um grande conjunto de operações matemáticas como métodos de classe — não existem instâncias da classe Math, mas você ainda pode usar seus métodos com argumentos numéricos ou booleanos. Por exemplo, o método de classe Math.max( ) recebe dois argumentos e retorna o maior dos dois. Você não precisa criar uma nova instância de Math — ele pode ser chamado de qualquer lugar em que você precisar, como no seguinte:

int maximumPrice = Math.max(firstPrice, secondPrice); a notação de ponto é usada para chamar um método de classe. Assim como acontece com as variáveis de classe, você pode usar uma instância da classe ou a própria classe no lado esquerdo do ponto. Entretanto, pelas mesmas razões observadas na discussão sobre variáveis de classe, usar o nome da classe torna seu código mais fácil de ler. As duas últimas linhas desse exemplo produzem o mesmo resultado — o string 5:

```
String s, s2;
s = "item";
s2 = s.value0f(5);
s2 = String.value0f(5);
```

Referências a objetos

Quando você trabalha com objetos, uma coisa importante a ser entendida é o uso de referências.

NOVO TERMO

Uma *referência* é um tipo de ponteiro usado para indicar o valor de um objeto.

Quando você atribui um objeto a uma variável ou passa um objeto para um método como argumento, na verdade não está usando objetos. Você não está nem mesmo usando cópias dos objetos. Em vez disso, você está utilizando referências a esses objetos.

Para ilustrar melhor a diferença, a listagem 4.4 mostra como as referências funcionam.

DIGITE

Listagem 4.4 O texto completo de Referencestest.java.

```
1: import java.awt.Point;
2:
3: class ReferencesTest {
4:     public static void main (String arguments[ ]) {
5:         Point ptl, pt2;
6:         ptl = new Point(100, 100);
7:       pt2 = ptl;
8:
9:         ptl.x = 200;
10:       ptl.y = 200;
11:       System.out.println("Pointl: " + ptl.x + ", " + ptl.y);
12:       System.out.println("Point2: " + pt2.x + ", " + pt2.y);
13:    }
14: }
```

A seguir está a saída deste programa:

SAÍDA

```
Pointl: 200, 200
Point2: 200, 200
```

Ocorre o seguinte na primeira parte desse programa:

- Linha 5 Duas variáveis Point são criadas.
- Linha 6 Um novo objeto Point é atribuído a ptl.
- Linha 7 O valor de ptl é atribuído a pt2.

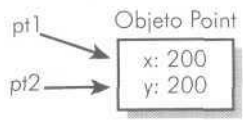
As linhas 9 a 12 representam a parte complicada. As variáveis x e y de ptl são definidas como 200 e, então, todas as variáveis de ptl e pt2 são apresentadas na tela.

Você poderia esperar que pt1 e pt2 tivessem valores diferentes. No entanto, a saída mostra que isso não acontece. Conforme você pode ver, as variáveis x e y de pt2 também foram alteradas, mesmo que nada no programa as tenha alterado explicitamente.

Isso acontece porque a linha 7 cria uma referência de pt2 para ptl, em vez de criar pt2 como um novo objeto copiado de ptl.

pt2 é uma referência ao mesmo objeto que ptl; isso está ilustrado na Figura 4.1. Qualquer variável pode ser usada para fazer referência ao objeto ou mudar suas variáveis.

Fiaura 4.1
Referências a *objetos.*



Se você quisesse que ptl e pt2 fizessem referência a objetos distintos, a separação das instruções new Point( ) poderia ser usada nas linhas 6 e 7, para criar objetos separados, como se vê a seguir:

```
ptl = new Point(100, 100);
pt2 = new Point(100, 100);
```

O uso de referências na linguagem Java se torna particularmente importante quando argumentos são passados para métodos. Você aprenderá mais sobre isso posteriormente, ainda hoje.

Nota

Não existem ponteiros explícitos ou aritmética de ponteiros na linguagem Java, como acontece em C e C++. Entretanto, usando-se referências e arrays Java, a maioria dos recursos de ponteiro é reproduzida sem todos aqueles inconvenientes.

Coerção e conversão de objetos e tipos orimitivos

Uma coisa que você descobre rapidamente sobre a linguagem Java é como ela é meticulosa com relação às informações que irá manipular. Esta linguagem espera que as coisas estejam de certa maneira e não tolera alternativas.

Quando você está enviando argumentos para métodos ou usando variáveis em expressões, deve utilizar variáveis com os tipos de dados corretos. Se um método exige um i nt, o compilador Java responderá com um erro, caso você tente enviar um valor float para o método. Do mesmo modo, se você estiver definindo uma variável com o valor de outra, elas deverão ser do mesmo tipo.

Nota

Existe uma área em que o compilador Java é decididamente direto: os strings. O tratamento de strings em métodos `println( )`, instruções de atribuição e argumentos de método é simplificado com o uso do operador de concatenação (+). Se qualquer variável em um grupo de variáveis concatenadas for um string, a linguagem Java tratará tudo como um `String`. Isso torna possível o seguinte:

```
float gpa = 2.25F;
System.out.println("Honest, dad, my GPA is a " + (gpa+1.5));
```

Às vezes, você terá um valor em seu programa Java que não é do tipo correto para o que precisa. Ele poderia ser da classe errada ou do tipo de dado errado — como um `float`, quando você precisa de um `int`.

Você usa coerção para converter um valor de um tipo para outro.

NOVO TERMO

Coerção é o processo de produção de um novo valor que tem um tipo diferente de sua origem. O significado é semelhante a representar, onde um personagem de um show de TV pode ser substituído por outro ator após uma negociação salarial ou uma infeliz prisão por obscenidade em público.

Você não altera o valor de uma variável quando ela passa pela coerção. Em vez disso, você cria uma nova variável do tipo desejado.

Embora o conceito de coerção seja razoavelmente simples, a utilização é complicada pelo fato de que a linguagem Java possui tipos primitivos (como `int`, `float` e `boolean`) e tipos de objeto (`String`, `Point`, `ZipFile` e coisas assim). Existem três formas de coerções e conversões para discutirmos nesta seção:

- Coerção entre tipos primitivos, como `int` para `float` ou `float` para `double`
- Coerção de uma instância de uma classe para uma instância de outra classe
- Conversão de tipos primitivos para objetos, seguida da extração de valores primitivos desses objetos

Ao se discutir a coerção, pode ser mais fácil pensar em termos de origens e destinos. A origem é a variável que está sofrendo coerção para outro tipo. O destino é o resultado.

Coerção de tipos primitivos

A coerção entre tipos primitivos permite que você converta o valor de um tipo primitivo para outro. Isso ocorre normalmente com os tipos numéricos e existe um tipo primitivo que nunca pode ser usado em uma coerção. Os valores booleanos devem ser `true` ou `fal se` e não podem ser usados em uma operação de coerção.

Em muitas coerções entre tipos primitivos, o destino pode conter valores maiores do que a origem, de modo que o valor é facilmente convertido. Um exemplo seria a coerção de um byte em um int. Como um byte contém valores de -128 a 127 e um int contém valores de -2,1 milhões a 2,1 milhões, há espaço mais do que suficiente para se fazer a coerção de um byte para um int.

Freqüentemente, você pode usar um byte ou um char como um int automaticamente; você pode usar um int como um long, um int como um float ou qualquer coisa como um double. Na maioria dos casos, como o tipo maior proporciona mais precisão do que o menor, nenhuma perda de informação ocorre como resultado. A exceção é a coerção de inteiros para valores em ponto flutuante — a coerção de um int ou de um long para um float ou de um long para um double pode causar alguma perda de precisão.

Um caractere pode ser usado como um int, pois cada caractere tem um código numérico correspondente que representa sua posição no conjunto de caracteres. Se a variável i possui o valor 65, a coerção char(i) produz o valor de caractere 'A'. O código numérico associado a um A maiúsculo é 65, de acordo com o conjunto de caracteres ASCII, e isso foi adotado pela linguagem Java como parte de seu suporte a caracteres.

Você deve usar uma coerção explícita para converter um valor grande no tipo menor, pois a conversão desse valor poderia resultar em uma perda de precisão. As coerções explícitas assumem a seguinte forma:

```
(typename)value
```

No termo anterior, *typename é* o nome do tipo de dados para o qual você está convertendo, como short, int e float. value é uma expressão que resulta no valor do tipo de origem. Por exemplo, o valor de x é dividido pelo valor de y e o resultado sofre coerção para um int, na seguinte expressão:

```
(int)(x / y);
```

Observe que, como a precedência da coerção é maior do que a da operação aritmética, você precisa usar parênteses aqui — caso contrário, primeiro o valor de x sofreria coerção para um int e, em seguida, seria dividido por y, o que poderia produzir facilmente um resultado diferente.

Coerção de objetos

As instâncias de classes também podem sofrer coerção para instâncias de outras classes, com uma restrição: as classes de origem e destino devem estar relacionadas pela herança. Uma classe deve ser uma subclasse da outra.

De forma análoga à conversão de um valor primitivo em um tipo maior, alguns objetos talvez não precisem sofrer coerção explicitamente. Em particu-

lar, como as subclasses contêm as mesmas informações de suas superclasses, você pode utilizar uma instância de uma subclasse em qualquer parte que uma superclasse seja esperada.

Considere, por exemplo, um método que receba dois argumentos: um do tipo Object e outro do tipo Window. Você pode passar uma instância de qualquer classe para o argumento Object, pois todas as classes Java são subclasses de Object. No caso do argumento Window, você pode passar suas subclasses, como Diaiog, FileDiaiog e Frame.

Isso vale para qualquer parte de um programa — e não apenas dentro das chamadas de método. Se você tivesse uma variável definida como classe Window, poderia atribuir objetos dessa classe ou de qualquer uma de suas subclasses para essa variável, sem coerção.

Isso vale no sentido contrário e você pode usar uma superclasse quando uma subclasse for esperada. Entretanto, há um inconveniente. Como as subclasses contêm mais comportamento do que suas superclasses, há uma perda de precisão envolvida. Esses objetos de superclasse podem não ter todo o comportamento necessário para atuar no lugar de um objeto de subclasse. Por exemplo, se você tem uma operação que chama métodos em objetos da classe Integer, usar um objeto da classe Number não incluirá muitos métodos especificados em Integer. Erros ocorrerão, se você tentar chamar métodos que o objeto de destino não possui.

Para usar objetos de superclasse onde são esperados objetos de subclasse, você precisa fazer sua coerção explicitamente. Você não perderá nenhuma informação na coerção, mas ganhará todos os métodos e variáveis que a subclasse define. Para fazer a coerção de um objeto para outra classe, você utiliza a mesma operação que usou para tipos de base:

```
[classname)objetct
```

Nesse caso, *classname* é o nome da classe de destino e *object* é uma referência ao objeto de origem. Observe que a coerção cria uma referência ao antigo objeto do tipo classname; o antigo objeto ainda continua existindo, como antes.

O exemplo a seguir faz a coerção de uma instância da classe VicePresident para uma instância da classe Employee; VicePresident é uma subclasse de Employee, com mais informações para definir que o VicePresident tem privilégios no banheiro do executivo:

```
Employee emp = new Employee( );
VicePresident veep = new VicePresident( );
emp = veep; // não é necessária coerção para uso ascendente
veep = (VicePresident)emp; // deve fazer a coerção explicitamente
```

A coerção de um objeto é necessária quando você utiliza as novas operações de desenho de imagens gráficas bidimensionais, introduzidas na linguagem Java 2. Você deve fazer a coerção de um objeto Graphics em um

objeto Graphi cs2D, antes de poder desenhar na tela. O exemplo a seguir utiliza um objeto Graphics chamado screen para criar um novo objeto Graphics2D, chamado screen2D:

```
Graphics2D screen2D = (Graphics2D)screen;
```

Graphi cs2D é uma subclasse de Graphi cs e ambas estão no pacote j ava. awt. Você vai explorar completamente o assunto no Dia 9.

Além da coerção de objetos para classes, você também pode fazer a coerção de objetos para interfaces — mas apenas se a classe desse objeto ou uma de suas superclasses implementar realmente a interface. Fazer a coerção de um objeto para uma interface significa que você pode chamar um dos métodos dessa interface, mesmo que a classe desse objeto não implemente realmente essa interface.

Convertendo tipos primitivos emobietos e vice-versa

Uma coisa que você não pode fazer sob nenhuma circunstância é a coerção de um objeto para um tipo de dado primitivo ou vice-versa. Os tipos primitivos e objetos são coisas muito diferentes na linguagem Java e você não pode fazer a coerção automaticamente entre os dois ou usá-los indiscriminadamente.

Como alternativa, o pacote java.lang inclui classes que correspondem a cada tipo de dado primitivo: Integer, Float, boolean etc. Observe que os nomes de classe possuem a letra inicial maiúscula e os tipos primitivos têm nomes em letras minúsculas. A linguagem Java trata os tipos de dado e suas versões de classe de modo muito diferente e um programa não será compilado com sucesso se você utilizar um deles, quando se esperar o outro.

Usando os métodos de classe definidos nessas classes, você pode criar um objeto para cada um dos tipos primitivos, utilizando a instrução new. A instrução a seguir cria uma instância da classe Integer com o valor 4403:

```
Integer dataCount = new Integer(4403);
```

Quando você tem um objeto criado dessa maneira, pode usá-lo como faria com qualquer objeto. Quando você quiser usar esse valor novamente como um valor primitivo, existem métodos para isso também. Por exemplo, se você quisesse obter um valor i nt a partir de um objeto dataCount, a seguinte instrução seria utilizada:

```
int newCount = dataCount.intValue( ); // retorna 4403
```

Uma transformação comum necessária em programas é a conversão de um String para um tipo numérico, como um inteiro. Quando você precisa de um int como resultado, isso pode ser feito usando-se o método parselnt( ) da classe Integer. O String a ser convertido é o único argumento enviado para o método, como no exemplo a seguir:

```
String pennsylvania = "65000";
int penn = Integer.parseInt(pennsylvania);
```

A documentação da API Java inclui detalhes sobre essas classes. Você pode encontrar essas páginas HTML na seção Documentation do site da Web da Sun Microsystems:

http://java.sun.com

Existem classes de tipo especial para boolean, Byte, Character, Double, Float, Integer, Long, Short e Void.

Comparando valores de objeto e classes

Além da coerção, existem operações que você executa sobre objetos:

- Comparação de objetos.
- Descoberta da classe de qualquer objeto dado.
- Teste para ver se um objeto é uma instância de determinada classe.

Comparando objetos

Ontem, você aprendeu a respeito dos operadores para comparar valores: igual, diferente, menor que etc. A maioria desses operadores trabalha apenas com tipos primitivos e não com objetos. Se você tentar usar outros valores como operandos, o compilador Java produzirá erros.

A exceção dessa regra são os operadores de igualdade: == (igual) e ! = (diferente). Quando usados com objetos, esses operadores não fazem o que você poderia estar esperando inicialmente. Em vez de verificar se um objeto tem o mesmo valor que outro, eles determinam se os objetos são os mesmos.

Para comparar instâncias de uma classe e obter resultados significativos, você deve implementar métodos especiais em sua classe e chamar esses métodos.

Um bom exemplo disso é a classe String. É possível ter dois objetos String diferentes que contenham os mesmos valores. Entretanto, se você usasse o operador == para comparar esses objetos, eles seriam considerados desiguais. Embora seu conteúdo combine, eles não são o mesmo objeto.

Para ver se dois objetos Stri ng possuem valores que combinam, é usado um método da classe chamado equal s ( ). O método testa cada caractere do string e retorna true se os dois strings possuírem os mesmos valores. A listagem 4.5 ilustra isso.

Listagem 4.5 O texto completo de Equalstest.java.

```
1: class EqualsTest {
2:     public static void main(String args[ ]) {
3:         String str1, str2;
4:         strl « "Free the bound periodicals.";
```

Listagem 4.5 Continuação

```
5:          str2 = strl;
6:
7:          System.out.println("Stringl: " + strl);
8:          System.out.println("String2: " + str2);
9:          System.out.println("Same object? " + (strl == str2));
10:
11:         str2 = new String(strl);
12:
13:         System.out.println("Stringl: " + strl);
14:         System.out.println("String2: " + str2);
15:         System.out.println("Same object? " + (strl == str2));
16:         System.out.println("Same value? " + strl.equals(str2));
17:     }
18: }
```

SAÍDA A saída desse programa é a seguinte:

```
Stringl: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
Stringl: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? False
Same value? true
```

A primeira parte desse programa (linhas 3 a 5) declara duas variáveis (str 1 e str2), atribui a literal Free the bound periodicals. a strl e, em seguida, atribu: esse valor a str2. Conforme você aprendeu anteriormente, strl e str2 agora apontam para o mesmo objeto e o teste de igualdade na linha 9 prova isso.

Na segunda parte, você cria um novo objeto String com o mesmo valor de strl e atribui str2 a esse novo objeto String. Agora, você tem dois objetos string diferentes em strl e str2, ambos com o mesmo valor. Testá-los para ver se eles são o mesmo objeto, usando o operador == (linha 15), retorna a resposta esperada (false — eles não são o mesmo objeto na memória). Testá-los usandc o método equals( ), na linha 16, também retorna a resposta esperada (true — eles possuem os mesmos valores).

Por que você não pode usar apenas outra literal quando altera str2, em vez de usar new? As literais de string são otimizadas na linguagem Java — se você criar um string usando uma literal e depois usar outra literal com os mesmos caracteres, a linguagem Java saberá o suficiente para devolver-lhe o primeiro objeto String. Os dois strings são o mesmo objeto — você tem de sair fora de seu caminho para criar dois objetos separados.

Determinando a classe de um objeto

Quer descobrir qual é a classe de um objeto? Aqui está o modo de se fazer isso, para um objeto atribuído à variável obj:

```
String name = obj.getClass( ).getName( );
```

O que isso faz? O método getClass( ) é definido na classe Object e, como tal, está disponível para todos os objetos. O resultado desse método é um objeto Class (onde Class é a própria classe), que possui um método chamado getName( ). getName( ) retorna um string representando o nome da classe.

Outro teste que poderia ser útil é o operador instanceof. Esse operador possui dois operandos: um objeto à esquerda e um nome de classe à direita. A expressão retorna true ou f alse dependendo de o objeto ser uma instância da classe nomeada ou de qualquer uma das subclasses dessa classe. Por exemplo:


```
"swordfish" instanceof String // true
Point pt = new Point(10, 10);
Pt instanceof String // false
```

O operador instanceof também pode ser usado para interfaces; se um objeto implementa uma interface, o operador instanceof com esse nome de interface no lado direito retorna true.

Inspecionando classes e métodos com reflexão

Um dos aprimoramentos feitos na linguagem Java após a versão 1.0 foi a introdução da reflexão, que também é chamada de *introspecção.* Independente do nome, a *reflexão* permite que uma classe Java — como um programa que você escreve — aprenda detalhes sobre qualquer outra classe.

Através da reflexão, um programa Java pode carregar uma classe sobre a qual ele nada conhece, encontrar as variáveis, métodos e construtores dessa classe e trabalhar com eles.

Isso poderia fazer mais sentido se você visse um exemplo logo de cara. A listagem 4.6 é um pequeno aplicativo Java, chamado SeeMethods.

DIGITE **Listagem 4.6** O texto completo de SeeMethods.java.

```
1: import java.lang.reflect.*;
2: import java.útil.Random;
3:
4: class SeeMethods {
5:     public static void main(String[ ] arguments)  {
6:          Random rd = new Random( );
7:          Class className = rd.getClass( );
8:          Method[ ] methods = className.getMethods( );
9:          for (int i = 0; i < methods.length; i++) {
```

Listagem 4.6 Continuação

```
10:                System.out.println("Method:  " + methods[i]);
11:           }
12:      }
13: }
```

Esse programa usa o grupo de classe java.lang.reflect.*, que fornece informações sobre os atributos, métodos e métodos construtores de qualquer classe.

O aplicativo SeeMethods cria um objeto Random na linha 6 e depois utiliza reflexão para apresentar todos os métodos públicos que fazem parte da classe. A listagem 4.7 mostra a saída do aplicativo.

SAÍDA **Listagem 4.7** A saída do aplicativo SeeMethods.

```
1: Method: public final native java.lang.Class java.lang.Object.getClass( )
2: Method: public native int java.lang.Object.hashCode( )
3: Method: public boolean java.lang.Object.equals(java.lang.Object)
4: Method: public java.lang.String java.lang.Object.toString( )
5: Method: public final native void java.lang.Object.notify( )
6: Method: public final native void java.lang.Object.notifyAll( )
7: Method: public final native void java.lang.Object.wait(long) throws
java.lang.InterruptedExcepti on
8: Method: public final void java.lang.Object.wait(long,int) throws
j ava.lang.InterruptedExcepti on
9: Method: public final void java.lang.Object.wait( ) throws
java.1ang.InterruptedExcepti on
10: Method: public synchronized void java.útil.Random.setSeed(long)
11: Method: public void java.útil.Random.nextBytes(byte[ ])
12: Method: public int java.útil.Random.nextlnt( )
13: Method: public long java.útil.Random.nextLong( )
14: Method: public float java.útil.Random.nextFloat( )
15: Method: public double java.útil.Random.nextDouble( )
16: Method: public synchronized double java.util.Random.nextGaussian( )
```

Usando a reflexão, o aplicativo SeeMethods pode aprender cada método da classe Random e todos os métodos que ela herda das superclasses Random. Cada linha da listagem mostra as seguintes informações sobre um método:

- Se ele é public.
- Que tipo de objeto ou variável o método retorna.
- Se o método é da classe atual ou de uma de suas superclasses.
- O nome do método.
- O tipo de objeto e variáveis utilizadas como argumentos ao se chamar o método.

O aplicativo SeeMethods poderia ser feito com qualquer classe de objetos. Altere a linha 6 de SeeMethods. java para criar um objeto diferente e dê uma olhada em seu interior.

A reflexão é mais usada por ferramentas como os navegadores e depuradores de classe, como um modo de aprender mais sobre a classe de objetos que está sendo verificada ou depurada. Ela também é necessária com o JavaBeans, onde a capacidade de um objeto consultar outro a respeito do que ele pode fazer (e depois pedir para que ele faça algo) é útil ao se criar aplicativos maiores. Você aprenderá mais sobre o JavaBeans noo Dia 19.

O pacote java.lang.reflect inclui as seguintes classes:

- Field Gerencia e localiza informações sobre variáveis de classe e de instância.
- Method Gerencia métodos de classe e de instância.
- Constructor Gerencia os métodos especiais de criação de novas instâncias de classes.
- Array Gerencia arrays.
- Modifier Decodifica informações de modificador sobre classes, variáveis e métodos (o que será descrito no Dia 15).

Além disso, existirão muitos métodos novos disponíveis em uma classe de objetos chamada Cl ass, que ajudarão a unir as várias classes de reflexão.

A reflexão é um recurso avançado que talvez você não utilize imediatamente em seus programas. Ela se torna mais útil quando você está trabalhando em serialização de objeto, JavaBeans e outra programação Java mais sofisticada.

Resumo

Agora que você se embebedou da implementação da programação orientada a objetos em Java, está em uma situação melhor para decidir como ela pode ser útil em sua própria programação.

Se você é do tipo pessimista, a programação orientada a objetos é um nível de abstração que fica no caminho do que você está tentando usar em uma linguagem de programação. Você aprenderá mais sobre o motivo pelo qual a OOP está completamente cristalizado na linguagem Java, nos próximos capítulos.

Se você é do tipo otimista, a programação orientada a objetos vale a pena ser usada, pelas vantagens que oferece: confiabilidade, capacidade de reutilização e manutenção aprimoradas.

Hoje, você aprendeu a tratar com objeto: criá-los, ler seus valores e alterá-los e chamar seus métodos. Você também aprendeu a converter objetos de uma classe para outra ou de um tipo de dado para uma classe.

Finalmente, você viu pela primeira vez a reflexão, um modo de fazer um objeto revelar detalhes sobre si mesmo.

Neste ponto, você possui os conhecimentos necessários para tratar das tarefas mais simples na linguagem Java. Restam apenas os arrays, as condicionais e os loops (que serão tratados amanhã) e como definir e usar classes (no Dia 6).

Perguntas e respostas

P **Estou confuso quanto às diferenças entre objetos e os tipos de dados primitivos, como int e boolean.**

R Os tipos primitivos da linguagem (byte, short, int, long, float, double, boolean e char) representam os menores elementos da linguagem. Eles não são objetos, embora possam, de muitas maneiras, ser tratados como objetos: eles podem ser atribuídos a variáveis e passados dentro e fora de métodos. Entretanto, a maioria das operações que trabalham exclusivamente com objetos não funcionará com os tipos primitivos.

Os objetos são instâncias de classes e, como tal, normalmente são tipos de dados muito mais complexos do que simples números e caracteres, freqüentemente contendo números e caracteres como variáveis de instância ou de classe.

P **Não existem ponteiros na linguagem Java? Se não existem ponteiros, que devo fazer com algo do tipo listas encadeadas, onde existe um ponteiro de um nó para outro para que se possa percorrê-las?**

R Não é correto dizer que a linguagem Java não possui ponteiros — ela não tem ponteiros explícitos. As referências de objetos são, efetivamente, ponteiros. Para criar algo como uma lista encadeada, você criaria uma classe chamada Node, que teria uma variável de instância também do tipo Node. Para encadear os objetos do nó, atribua um objeto à variável de instância do objeto imediatamente anterior a ele na lista. Como as referências de objeto são ponteiros, as listas encadeadas definidas dessa maneira se comportam conforme você poderia esperar.

SEMANA 1

Dia 5

Listas, lógica e loops

Se você escrevesse um programa Java com o que sabe até agora, provavelmente ele seria um pouco sem graça. Se você escrevesse um programa Java com o que sabe até agora, provavelmente ele seria um pouco sem graça. Esta última sentença não foi repetida devido a um erro editorial. E uma demonstração de como é fácil para os computadores repetir a mesma coisa indefinidamente. Hoje, você aprenderá a fazer uma parte de um programa Java repetir-se, usando loops.

Além disso, você aprenderá a fazer um programa decidir se vai realizar algo com base em uma lógica. (Talvez um computador decida que não é lógico repetir a mesma sentença duas vezes seguidas em um livro.)

Você também aprenderá a organizar grupos da mesma classe ou tipo de dado em listas denominadas arrays.

O primeiro item da lista de hoje são os arrays. O primeiro item da lista de hoje são os arrays.

Arrays

Neste ponto, você tratou apenas com algumas variáveis em cada programa Java. Em alguns casos, é possível utilizar variáveis individuais para armazenar informações.

Entretanto, e se você tivesse 20 itens de informações relacionadas para controlar? Você poderia criar 20 variáveis diferentes e definir seus valores iniciais, mas isso se tornaria mais inconveniente à medida que você fosse obrigado a trabalhar com mais informações. E se existissem 100 itens ou mesmo 1.000?

Os arrays representam um modo de armazenar uma lista de itens que tenham o mesmo tipo de dado primitivo ou a mesma classe. Cada item da lista fica em seu próprio slot que é numerado para que você possa acessar as informações facilmente.

Os arrays podem conter qualquer tipo de informação que seja armazenada em uma variável, mas uma vez criado o array, você só pode usá-lo para esse tipo de informação. Por exemplo, você pode ter um array de inteiros, um array de objetos Stri ng ou um array de arrays, mas não pode ter um array que contenha tanto strings quanto inteiros.

A linguagem Java implementa os arrays de forma diferente de algumas outras linguagens — como objetos que podem ser tratados exatamente como os outros objetos.

Para criar um array em Java, faça o seguinte:

1. Declare uma variável para conter o array.
2. Crie um novo objeto de array e atribua-o à variável de array.
3. Armazene informações nesse array.

Declarando variáveis de array

O primeiro passo na criação de um array é declarar uma variável que irá contê-lo. As variáveis de array indicam o objeto ou tipo de dados que o array conterá, além do nome do array. Para diferenciar das declarações de variável normais, um par de colchetes vazios ([ ]) é incluído no objeto ou tipo de dados, ou no nome da variável.

As instruções a seguir são exemplos de declarações de variável de array:

```
String difficultWordsf ];
Point hits[ ];
int donations[ ];
```

Como você pode colocar os colchetes após o tipo de informação, em vez do nome da variável, as seguintes instruções também funcionam:

```
String[ ] difficultWords;
Point[ ] hits;
int[ ] guesses;
```

Você vê os dois estilos usados em programas e como não há um consenso sobre qual estilo é mais legível, a escolha é amplamente uma questão de preferência pessoal.

Criando obietos de array

Após se declarar a variável de array, o próximo passo é criar um objeto de array e atribuí-lo a essa variável. As maneiras de se realizar esse passo são as seguintes:

- Utilizar o operador new.
- Inicializar o conteúdo do array diretamente.

Como, em Java, os arrays são objetos, você pode usar o operador new para criar uma nova instância de um array, como na instrução a seguir:

```
String[ ] playerNames = new String[10];
```

Essa instrução cria um novo array de strings com 10 slots que podem conter objetos String. Quando você criar um objeto de array usando o operador new, deverá indicar quantos slots o array conterá. Essas instruções não colocam realmente os objetos String nos slots — você precisa fazer isso depois.

Os objetos de array podem conter tipos primitivos, como inteiros ou booleanos, assim como objetos. Por exemplo:

```
int[ ] temps = new int[99];
```


Quando você cria um objeto de array usando new, todos os seus slots são inicializados automaticamente (0 para arrays numéricos, false para booleanos, '\0' para arrays de caracteres e null para objetos).

Você também pode criar e inicializar um array simultaneamente. Em vez de usar o operador new para criar o novo objeto de array, coloque os elementos do array entre chaves, separados por vírgulas. Por exemplo:

```
String[ ] chiles = { "jalapeno", "anaheim", "serrano", "habanero", "thai" };
```

> A palavra-chave null da linguagem Java se refere a um objeto null (e pode ser usada para qualquer referência de objeto). Ela não é equivalente a zero ou ao caractere '\0', como acontece com a constante NULL em C.

Cada um dos elementos dentro das chaves deve ser do mesmo tipo que a variável que contém o array. Quando você cria um array com valores iniciais dessa maneira, ele tem o mesmo tamanho do número de elementos incluídos entre chaves. O exemplo anterior cria um array de objetos String chamado chiles, que contém cinco elementos.

Acessando os elementos do array

Uma vez que você tenha um array com valores iniciais, pode recuperar, alterar e testar os valores de cada um de seus slot. O valor de um slot é acessado com o nome do array, seguido de um subscrito colocado dentro de colchetes. Esses nome e subscrito podem ser colocados em expressões, como no seguinte:

```
contestantScore[40] = 470;
```

A parte contestantScore dessa expressão é uma variável que contém um objeto de array, embora ela também possa ser uma expressão que resulte em um array. A expressão do subscrito especifica o slot a ser acessado dentro do array. Os subscritos de array começam com 0, como acontece em C e C + + ; portanto, um array com 10 elementos possui slots que são acessados usando-se subscritos de 0 a 9.

Todos os subscritos são verificados para se garantir que eles estejam dentro dos limites do array, conforme especificado quando o array foi criado. Na linguagem Java, é impossível acessar ou atribuir um valor para um slot que esteja fora dos limites do array, o que evita problemas que resultam na ultrapassagem dos limites de um array, como nas linguagens do tipo C. Observe as duas instruções a seguir:

```
String[ ] beatleSpeak = new String[10];
beatleSpeak[10] = "I am the eggman.";
```

Um programa com as duas linhas de código precedentes produziria um erro de compilação, quando beatleSpeak[10] fosse usado. O erro ocorre porque o array beatleSpeak não possui um slot 10 — ele tem 10 slots, que começam em 0 e terminam em 9. O compilador Java pegará esse erro.

O interpretador Java produzirá um erro, se o subscrito do array for calculado quando o programa estiver em execução (por exemplo, como parte de um loop) e o subscrito acabar fora dos limites do array. Na verdade, para sermos tecnicamente corretos, ele gera uma exceção. Você aprenderá mais sobre exceções no Dia 16.

Como você pode evitar a ultrapassagem acidental do final de um array em seus programas? Teste o comprimento do array em seus programas, usando a variável de instância length — ela está disponível para todos os objetos de array, independentemente do tipo. Por exemplo:

```
int len = beatleSpeak.length
```

Para reiterar: o comprimento do array é 10, mas seu subscrito só pode ir até 9. Os arrays iniciam a numeração a partir de 0. Quando você trabalhar com arrays, lembre-se disso e subtraia 1 do comprimento do array para obter seu maior elemento.

Alterando elementos de array

Conforme você viu nos exemplos anteriores, é possível atribuir um valor a um slot específico de um array, colocando-se uma instrução de atribuição após o nome e o subscrito do array, como nos exemplos a seguir:

```
myGrades[4] = 85;
sentence[0] = "The";
sentence[10] = sentence[0];
```

Uma detalhe importante a ser observado é que um array de objetos na linguagem Java é um array de referências a esses objetos. Quando você atribui um valor a um slot nesse tipo de array, está criando uma referência a esse objeto. Quando você move valores dentro de arrays — como no último dos exemplos anteriores —, está fazendo uma nova atribuição da referência e não copiando um valor de um slot para outro. Os arrays de um tipo de dado primitivo, como int ou float, copiam mesmo os valores de um slot para outro.

Os arrays são razoavelmente simples de criar e modificar, mas eles propiciam uma enorme quantidade de funcionalidade para a linguagem Java. Quanto mais usar a linguagem, mais você vai usar os arrays.

Para concluir a discussão sobre arrays, a listagem 5.1 mostra um programa simples que cria, inicializa, modifica e examina partes de um array.

DIGITE

Listagem 5.1 O texto completo de ArrayTest.java.

```
1: class ArrayTest {
2:
3:    Str1ng[ ] firstNames = { "Dennis", "Grace", "Bjarne", "James" };
4:    String[ ] lastNames = new String[firstNames.length];
5:
6:    void printNames( ) {
7:       int i = 0;
8:       System.out.println(firstNames[i]
9:        + " " + lastNames[i]);
10:      i++;
11:      System.out.println(firstNames[i]
12:        + " " + lastNames[i]);
13:      i++;
14:      System.out.println(firstNames[i]
15:        + " " + lastNames[i]);
16:      i++;
17:      System.out.println(firstNames[i]
18:        + " " + lastNames[i]);
19:   }
20:
21:   public static void main (String argumentsf ]) {
22:      ArrayTest a = new ArrayTest( );
23:      a.printNames( );
24:      System.out.println("———");
25:      a.lastNames[0] = "Ritchie";
26:      a.lastNames[l] = "Hopper";
27:      a.lastNames[2] = "Stroustrup";
28:      a.lastNames[3] = "Gosling";
29:      a.printNames( );
30:   }
31: }
```

5

A saída do programa é a seguinte:

SAÍDA

```
Dennis null
Grace null
Bjarne null
James null

Dennis Ritchie
Grace Hopper
Bjarne Stroustrup
James Gosling
```

Esse exemplo mais longo mostra como criar e usar arrays. A classe que é criada aqui, ArrayTest, possui duas variáveis de instância que contêm arrays de objetos Stri ng. A primeira, que é chamada f i rstNames, é declarada e inicializada na linha 3 para conter quatro strings. A segunda variável de instância, 1 astNames, é declarada e criada na linha 4, mas nenhum valor inicial é colocado nos slots. Observe também que o array 1astNames tem exatamente o mesmo número de slots que o array fi rstNames, pois a variável fi rstNames. length é utilizada como índice inicial do array. Quando usada em um objeto de array, a variável de instância length retorna o número de slots existentes no array.

A classe ArrayTest também possui dois métodos: printNames( )emain( ). printNames( ), definido nas linhas 6 a 19, é um método utilitário que percorre os arrays firstNames e 1astNames seqüencialmente, apresentando os valores de cada slot. Observe que o índice de array definido aqui (i) é ajustado inicialmente como 0, pois todos os slots de array da linguagem Java começam a numeração a partir de 0.

Finalmente, o método main( ) realiza o seguinte:

- A linha 22 cria uma instância inicial de ArrayTest, para que suas variáveis de instância e métodos possam ser usados.
- A linha 23 chama printNames( ) para mostrar como é o objeto inicialmente. O resultado dá as primeiras quatro linhas da saída; note que o array f i rstNames foi inicializado, mas os valores de 1 astNames são todos null. Se um array não for inicializado ao ser declarado, os valores iniciais dos slots serão vazios — null para arrays de objeto, 0 para números e false para booleanos.
- As linhas 25 a 28 definem os valores de cada um dos slots do array 1astNames como os strings efetivos.
- A linha 29 chama printNames( ) mais uma vez, para mostrar que o array 1 astNames agora possui valores e que cada prenome e sobrenome é impresso conforme o esperado. Os resultados são apresentados nas últimas quatro linhas da saída.

Se você não reconhece os nomes desse exemplo, pode achar que os autores estão fazendo referência aos seus amigos no livro. Todos eles são importantes projetistas de linguagens de programação de computador: Dennis Ritchie (C), Bjarne Stroustrup (C++), Grace Hopper (COBOL) e James Gosling (Java).

Uma última observação a ser feita a respeito da listagem 5.1 é que se trata de um exemplo terrível de estilo de programação. Normalmente, ao tratar com arrays, você pode usar loops para circular pelos elementos de um array, em vez de tratar cada um deles individualmente. Isso, em muitos casos, torna o código muito mais curto e mais fácil de ler. Quando você aprender sobre os loops, posteriormente no dia de hoje, verá uma nova versão desse exemplo.

Arrays multidimensionais

Se você já usou arrays em outras linguagens, poderá estar se perguntando se a linguagem Java pode manipular arrays multidimensionais — arrays com mais de um subscrito, possibilitando que mais de uma dimensão seja representada.

As dimensões são úteis quando se representa algo como uma grade x,y de elementos de array.

A linguagem Java não oferece suporte a arrays multidimensionais, mas você pode obter a mesma funcionalidade declarando um array de arrays. Esses arrays também podem conter arrays e assim por diante, para quantas dimensões forem necessárias.

As instruções a seguir mostram como declarar e acessar esses arrays de arrays:

```
int[ ][ ] coords = new int[12] [12];
coords[0] [0] = 1;
coords[0][1] = 2;
```

Instruções de bloco

Na linguagem Java, as instruções são agrupadas em blocos. O início e o final de um bloco são denotados com caracteres de chave — uma chave de abertura ({) para o início e uma chave de fechamento (}) para o final.

Você já usou blocos nos programas, durante os cinco primeiros dias. Você os utilizou para o seguinte:

- Conter as variáveis e métodos em uma definição de classe.
- Definir as instruções que pertencem a um método.

Os blocos também são chamados de *instruções de bloco*, pois um bloco inteiro pode ser usado em qualquer parte em que uma única instrução pode ser usada. Cada instrução dentro do bloco é então executada, de cima para baixo.

Os blocos podem ser colocados dentro de outros blocos, como você faz quando coloca um método dentro de uma definição de classe.

Um detalhe importante a ser observado a respeito de um bloco é que ele gera um escopo para as variáveis locais que são criadas dentro dele.

NOVO TERMO

Escopo é um termo de programação para a parte de um programa em que uma variável existe e pode ser usada. Se o programa sair do escopo dessa variável, ela não existirá e erros ocorrerão se você tentar acessá-la.

O escopo de uma variável é o bloco em que ela foi criada. Quando você pode declarar e usar variáveis locais dentro de um bloco, essas variáveis cessam de existir depois que o bloco acaba de ser executado. Por exemplo, o seguinte método testB1ock( ) contém um bloco:

```
void testBlock( ) {
   int x = 10;
   { // inicia o bloco
      int y = 40;
   y = y + x;
   } // finaliza o bloco
}
```

Existem duas variáveis definidas nesse método: x e y. O escopo da variável y é o bloco em que ela está e ela só pode ser usada dentro desse bloco. Haveria um erro se você tentasse utilizá-la em outra parte do método testBlock ( ). A variável x foi criada dentro do método, mas fora do bloco mais interno; portanto, ela pode ser usada em qualquer parte do método. Você pode modificar o valor de x em qualquer parte dentro do método e esse valor será mantido.

Normalmente, as instruções de bloco não são usadas sozinhas em uma definição de método, como aconteceu no exemplo anterior. Você as utiliza nas definições de classe e método, bem como em estruturas lógicas e de loop, sobre o que aprenderá a seguir.

Condicionais if

Um dos aspectos importantes da programação é a capacidade de um programa decidir o que irá fazer. Isso é feito por um tipo especial de instrução que é denominado condicional.

NOVO TERMO

Uma *condicional* é uma instrução de programação que é executada apenas se uma condição específica for satisfeita.

A condicional mais básica é a palavra-chave if. A condicional i f utiliza uma expressão booleana para decidir se uma instrução deve ser executada. Se a expressão retorna um valor true, a instrução é tratada.

A seguir, há um exemplo simples, que apresenta a mensagem "You call that a haircut?" apenas sob uma condição: se o valor da variável age for maior do que 39:

```
if (age > 39)
    System.out.println("You cal l  that a haircut?");
```

Se você quiser que algo mais aconteça no caso de a expressão i f retornar um valor f alse, uma palavra-chave opcional else pode ser usada. O exemplo a seguir utiliza i f e else:

```
if (blindDateIsAttractive == true)
    restaurant = "Benihana's";
else
    restaurant = "Burrito Hut";
```

A condicional i f executa diferentes instruções, de acordo com o resultado de um simples teste booleano.

Nota

A diferença entre as condicionais i f em Java e as do C ou C+ + é que a linguagem Java exige que o teste retorne um valor booleano (true ou false). Em C, o teste pode retornar um inteiro.

Usando if, você só pode incluir uma única instrução como o código a ser executado após o teste. (No exemplo anterior, é atribuída a variável restaurant.) Entretanto, na linguagem Java, um bloco pode aparecer em qualquer parte em que uma instrução possa aparecer. Se você quiser realizar mais do que apenas uma ação como resultado de uma instrução i f, então pode incluir essas instruções dentro de um bloco. Observe o fragmento de código a seguir, que é uma extensão do objeto Jabberwock que você criou no Dia 2:

```
if (attitude == "angry" ) {
   System.out.println("The jabberwock is angry.");
   System.out.println ("Have you made out a will?");
} else {
   System.out.println ("The jabberwock is in a good mood.");
   if (hungry)
      System.out.println("It still is hungry, though.");
   else System.out.println("It wanders off.");
}
```

Esse exemplo usa o teste (attitude == "angry") para determinar se deve mostrar que esse jabberwock está bravo (angry) ou contente (happy). Se o jabberwock estiver contente, o teste (hungry) será usado para ver se ele também está com fome (hungry) — supondo que um jabberwock com fome seja algo a ser evitado, mesmo que esteja contente. A condicional if (hungry) é outro modo de escrever if (hungry == true). No caso de testes booleanos desse tipo, omitir a última parte da expressão é um atalho de programação comum.

A listagem 5.2 mostra outro exemplo simples — este, na forma de um aplicativo completo. A classe EvenSteven contém um método utilitário chamado evenCheck ( ), que testa um valor para ver se ele é par. Se for, ele apresenta Steven! na tela.

DIGITE

Listagem 5.2 O texto completo de EvenSteven.java.

```
1: class EvenSteven {
2:
3:    void evenCheck(int val) {
4:       System.out.println("Value is "
5:         + val + ". ");
6:       if (val % 2 == 0)
7:       System.out.println("Steven!");
8:    }
9:
10:   public static void main (String arguments[ ]) {
11:      EvenSteven e = new EvenSteven( );
12:
13:      e.evenCheck(l);
14:      e.evenCheck(2);
15:      e.evenCheck(54);
16:      e.evenCheck(77);
17:      e.evenCheck(1346);
18:   }
19: }
```

A saída do programa é a seguinte:

SAÍDA

```
Value is 1.
Value is 2.
Steven!
Value is 54.
Steven!
Value is 77.
Value is 1346.
Steven!
```

O cerne da classe EvenSteven é o método evenCheck( ) (linhas 3 a 8), onde os valores são testados e uma mensagem apropriada é impressa. Diferente dos métodos definidos nos exemplos anteriores, observe que a definição de evenCheck( ) inclui um único argumento inteiro (ver linha 3). O método evenCheck( ) começa imprimindo o valor que foi passado a ele. Esse argumento é então testado, usando-se uma condicional i f para ver se se trata de um número par.

O teste de módulo, aprendido no Dia 3, retorna o resto da divisão de seus operandos. Se o resto de um número dividido por 2 for 0, trata-se de um número par.

Se o número for par, Steven! é apresentado. (Você aprenderá mais sobre a definição de métodos com argumentos, amanhã.)

Nesse aplicativo, o método mai n ( ) cria uma nova instância de EvenSteven e a testa, chamando o método evenCheck( ) repetidamente, com diferentes valores. Na saída, apenas os valores pares recebem uma mensagem Steven!.

O operador condicional

Uma alternativa à utilização das palavras-chave if e else em uma instrução condicional é usar o operador condicional, às vezes chamado de *operador ternário.* O *operador condicional é* chamado de operador ternário porque ele possui três termos.

O operador condicional é uma expressão, o que significa que ele retorna um valor — diferente do i f que é mais geral, que só pode resultar na execução de uma instrução ou de um bloco. O operador condicional é mais útil para condicionais curtas ou simples e assemelha-se à linha a seguir:

```
test ? trueresult : falseresult;
```

O item *test* é uma expressão que retorna true ou false, exatamente como o teste da instrução i f. Se o teste for true, o operador condicional retornará o valor de *trueresult.* Se o teste for false, o operador condicional retornará o valor *de falseresult.* Por exemplo, a seguinte condicional testa os valores de myScore e yourScore, retorna o maior dos dois como um valor e atribui esse valor à variável ourBestScore:

```
int ourBestScore = myScore > yourScore ? myScore : yourScore;
```

Essa utilização do operador condicional é equivalente ao seguinte código if.. .else:

```
int ourBestScore;
if (myScore > yourScore)
   ourBestScore = myScore;
else
   ourBestScore = yourScore;
```

O operador condicional tem uma precedência muito baixa — normalmente, ele é avaliado somente depois que todas as suas sub-expressões foram avaliadas. Os únicos operadores mais abaixo na precedência são os operadores de atribuição. Para um lembrete da precedência de operador, consulte a tabela de precedência do Dia 3.

Condicionais switch

Uma prática comum de programação em qualquer linguagem é testar uma variável em relação a algum valor, testá-la novamente em relação a um valor diferente, se ela não combinar com o primeiro e assim por diante. Esse processo pode se tornar difícil de manejar, se você estiver usando apenas instruções if, dependendo de quantos valores diferentes precisam ser testados. Por exemplo, você poderia acabar com um conjunto de instruções i f como o seguinte:

```
if (oper == '+')
   addargs(argl, arg2);
else if (oper == '-')
   subargs(argl, arg2);
```

```
else if (oper == '*')
   multargs(argl, arg2);
else if (oper == '/')
   divargs(argl, arg2);
```

Essa utilização de instruções if é chamada de if aninhado, porque cada instrução else contém outro i f, até que todos os testes possíveis tenham sido realizados.

Um mecanismo abreviado para as instruções i f aninhadas, que você pode usar em algumas linguagens de programação, é agrupar testes e ações em uma única instrução. Em Java, você pode agrupar ações com a instrução swi tch, que se comporta como acontece na linguagem C. A seguir, está um exemplo da utilização de swi tch:

```
switch (grade) {
   case 'A':
      System.out.println("Great job – an A!");
      break;
   case 'B':
      System.out.println("Good job — a B!");
      break;
   case 'C :
      System.out.println("Your grade was a C.");
      break;
   default: System.out.println("An F — consider cheating!");
}
```

A instrução swi tch é construída para testes. No exemplo anterior, o teste se dá no valor da variável grade. A variável de teste, que pode ser qualquer um dos tipos primitivos byte, char, short ou int, é comparada, por sua vez, com cada um dos valores case. Se uma combinação for encontrada, a instrução (ou instruções) após o teste será executada. Se nenhuma combinação for encontrada, a instrução ou instruções default serão executadas. O item default é opcional. Se ele for omitido e não houver combinação para nenhuma das instruções case, a instrução switch terminará sem executar nada.

A implementação Java de swi tch é limitada — os testes e valores só podem ser tipos primitivos simples que possam sofrer coerção para int. Você não pode usar tipos primitivos maiores, como long ou float, strings ou outros objetos, dentro de uma instrução swi tch, e também não pode testar nenhum relacionamento que não seja o de igualdade. Essas restrições limitam a instrução switch aos casos mais simples. Em contraste, as instruções i f aninhadas podem fazer toda espécie de teste, em qualquer tipo.

A seguir está uma revisão do exemplo de i f aninhado apresentado anteriormente. Ele foi reescrito como uma instrução switch:

```
switch (oper) {
   case '+':
      addargs(argl, arg2);
      break;
```

```
        case '*'_:
           multargs(argl, arg2);
           break;
        case '-':
           subtargs(argl, arg2);
           break;
        case '/':
           divargs(argl, arg2);
           break;
    }
```

Existem dois detalhes a serem observados nesse exemplo: o primeiro é que, após cada case, você pode incluir uma ou mais instruções de resultado — você pode incluir quantas precisar. Ao contrário do que acontece com i f, você não precisa colocar várias instruções entre chaves para que ele funcione. A segunda observação nesse exemplo são as instruções break incluídas em cada seção case. Sem uma instrução break em uma seção case, quando uma combinação for feita, as instruções dessa combinação e todas as que estiverem abaixo de swi tch serão executadas até que uma instrução break ou o final da instrução switch seja encontrado. Em alguns casos, isso pode ser exatamente o que você quer fazer. No entanto, na maioria dos casos, você deve incluir a instrução break para garantir que apenas o código correto seja executado. A instrução break, sobre a qual você aprenderá na seção "Saindo de loops", interrompe a execução no ponto atual e pula para o código que está fora da próxima chave de fechamento (})).

Uma utilização prática da falha sem uma instrução break ocorre quando vários valores devem executar as mesmas instruções. Para efetuar essa tarefa, você pode usar várias linhas case sem resultado; a instrução switch executará as primeiras instruções que encontrar. Por exemplo, na instrução switch a seguir, o string x is an even number. é impresso se x possuir os valores 2, 4, 6 ou 8. Todos os outros valores de x fazem o string x is an odd number. ser impresso.

```
switch (x) {
   case 2:
   case 4:
   case 6:
   case 8:
      System.out.println("x is an even number.");
      break;
   default: System.out.println("x is an odd number.");
}
```

A listagem 5.3 mostra mais um exemplo de uma instrução switch. Essa classe, NumberReader, converte valores inteiros em sua palavra real equivalente em inglês, usando um método chamado convert It ( ).

DIGITE

Listagem 5.3 O texto completo de NumberReader.java.

```
1: class NumberReader {
2:
3:    String convertNum(int val) {
4:       switch (val) {
5:          case 0: return "zero ";
6:          case 1: return "one ";
7:          case 2: return "two ";
8:          case 3: return "three ";
9:          case 4: return "four ";
10:         case 5: return "five ";
11:         case 6: return "six ";
12:         case 7: return "seven ";
13:         case 8: return "eight ";
14:         case 9: return "nine ";
15:         default: return " ";
16:      }
17:   }
18:
19:   public static void main (String arguments[ ]) {
20:         NumberReader n = new NumberReader( );
21:         String num = n.convertNum(4) + n.convertNum(l) + n.convertNum(3);
22:         System.out.println("413 converts to " + num);
23:   }
24: }.
```

A saída do programa é a seguinte:

SAÍDA

```
413 converts to four one three
```

É claro que o cerne desse exemplo é a instrução principal switch, no meio do método convertNum( ), nas linhas 4 a 16. Essa instrução switch recebe o argumento inteiro que é passado para convertNum( ) e, quando encontra uma combinação, retorna o valor de string apropriado. (Note que esse método é definido para retornar um string, em contraste com os outros métodos que você definiu até agora, que não retornavam nada. Você aprenderá mais sobre isso amanhã.)

Não há necessidade de instruções break no programa NumberReader, pois a instrução return é usada em seu lugar. A instrução return é semelhante à break, exceto por abrir a definição de método inteira e retornar um único valor. Novamente, você verá sobre isso amanhã, quando aprender como se define métodos.

Neste ponto, você provavelmente já viu suficientes métodos main ( ) para saber o que está ocorrendo, mas examine este rapidamente.

- A linha 20 cria uma nova instância da classe NumberReader.

- A linha 21 define um string chamado num, que será a concatenação dos valores de string de três números. Cada número é convertido usando-se uma chamada ao método convertNum( ).
- A linha 22 apresenta o resultado.

LOODS for

O loop for repete uma instrução um número especificado de vezes, até que uma condição seja satisfeita. Embora os loops for sejam usados freqüentemente para iteração simples, em que uma instrução é repetida um certo número de vezes, eles podem ser utilizados para praticamente qualquer tipo de loop.

Na linguagem Java, o loop for é mais ou menos o seguinte:

```
for (initialization; test; increment) {
    statement;
}
```

O início do loop for possui três partes:

- *initialization* é uma expressão que inicializa o começo do loop. Se você tem um índice de loop, essa expressão pode declará-lo e inicializá-lo, como em int i = 0. As variáveis declaradas nessa parte do loop for são locais a ele próprio; elas cessam de existir depois que o loop conclui sua execução. Você pode inicializar mais de uma variável nessa seção, separando cada expressão com uma vírgula. A instrução int i = 0, int j = 10 nessa seção, declararia as variáveis i e j e ambas seriam locais ao loop.
- *test é o* teste que ocorre após cada passagem do loop. O teste deve ser uma expressão booleana ou uma função que retorne um valor Boolean, como em i < 10. Se o teste for true, o loop será executado. Quando o teste for false, o loop parará de ser executado.
- *increment* é qualquer expressão ou chamada de função. Normalmente, o incremento é utilizado para alterar o valor do índice do loop, para aproximar o estado do loop ao retorno de false e interromper a sua execução. Semelhante à seção *initialization*, você pode colocar mais de uma expressão nesta seção, separando cada uma delas com uma vírgula.

A parte *statement* do loop for é a instrução que será executada sempre que o loop fizer uma iteração. Assim como acontece com a instrução if, você pode incluir uma única instrução ou uma instrução de bloco; o exemplo anterior usou um bloco porque isso é mais comum. O exemplo a seguir é um loop for que define todos os slots de um array String para o valor Mr.:

```
String[ ] salutation = new String[10];
int i; // a variável de índice do loop
```

```
for (i = 0; i < salutation.lerigth; i++)
   salutation[i] = "Mr.";
```

Nesse exemplo, a variável i serve como um índice de loop — ela conta o número de vezes em que o loop foi executado. Antes de cada passagem no loop, o valor do índice é comparado a salutation. 1ength, o número de elementos no array sal utati on. Quando o índice é igual ou maior do que salutation.l ength, o loop é abandonado.

O último elemento da instrução for é 1++. Isso faz o índice do loop ser incrementado em 1, sempre que for executado. Sem essa instrução, o loop nunca seria interrompido.

A instrução que está dentro do loop define um elemento do array sal utation igual a "Mr.". O índice do loop é usado para determinar o elemento que é modificado.

Qualquer parte do loop for pode ser uma instrução vazia — isto é, você pode incluir um ponto-e-vírgula sem nenhuma expressão ou instrução e essa parte do loop for será ignorada. Observe que, se usar uma instrução null em seu loop for, você mesmo poderá ter de inicializar ou incrementar todas as variáveis ou índices do loop, em todas as partes do programa.

Você também pode ter uma instrução vazia como o corpo de seu loop for, se quiser executar apenas o que está na primeira linha desse loop. Por exemplo, o loop for a seguir encontra o primeiro número primo maior do que 4.000. (Ele chama um método denominado notPrime( ), que, teoricamente, terá um modo de descobrir isso.)

```
for (i = 4001; notPrime(i); i += 2)
;
```

Um erro comum nos loops for é a colocação acidental de um ponto-e-vírgula no final da linha que inclui a instrução for. Por exemplo:

```
for (i = 0; i < 10; i++);
     x = x * i; // esta linha não está dentro do loop!
```

Nesse exemplo, o primeiro ponto-e-vírgula finaliza o loop sem executar x = x * i como sua parte. A linha x = x * i será executada apenas uma vez, pois ela está completamente fora do loop for. Cuidado para não cometer esse erro em seus programas Java.

Para terminar com os loops for, reescreva aquele exemplo com nomes, da seção de array. O exemplo original é longo e repetitivo e só funciona com um array de quatro elementos. Esta versão, apresentada na listagem 5.4, é mais curta e mais flexível (porém, retorna a mesma saída).

DIGITE

Listagem 5.4 O texto completo de NamesLoop.java.

```
1: class NamesLoop {
2:
3:    String[ ] firstNames = { "Dennis", "Grace", "Bjarne", "James" };
4:    String[ ] lastNames = new StringffirstNames.length];
5:
6:    void printNames( ) {
7:      for (int i = 0; i < firstNames.length; 1++)
8:        System.out.println(firstNames[i] + " " + lastNames[i]);
9:      }
10:
11:   public static void main (String arguments[ ]) {
12:     NamesLoop a = new NamesLoop( );
13:      a.printNames( );
14:     System.out.println(". . . . .");
15:    a.lastNamesf[0]    = "Ritchie";
16:     a.lastNames[l]  = "Hopper";
17:     a.lastNames[2]  = "Stroustrup";
18:     a.lastNames[3]  = "Gosling";
19:
20:      a.printNames( );
21:   }
22: }.
```

A saída do programa é a seguinte:

SAÍDA

```
Dennis   null
Grace   null
Bjarne   null
James   null

Dennis  Ritchie
Grace  Hopper
Bjarne  Stroustrup
James  Gosling
```

A única diferença entre esse exemplo e a listagem 5.1 está no método printNames( ). Em vez de passar, um a um, pelos slots do array, este exemplo utiliza um loop for para fazer a iteração através do array, um slot por vez, parando no último elemento. A utilização de um loop de propósito mais geral para fazer a iteração em um array permite que você use pri ntNames ( ) para qualquer array de qualquer tamanho e ainda possibilita que ele imprima todos os elementos.

Loops while e do

Os tipos restantes de loop são while e do. Assim como os loops for, os loops wh i 1 e e do permitem que um bloco de código Java seja executado repetidamente, até que uma condição específica seja satisfeita. Se você utiliza um loop for,

while ou do é principalmente uma questão de seu estilo de programação. Os loops while e do são exatamente iguais aos das linguagens C e C++, exceto que, na linguagem Java, a condição de teste deve ser um booleano.

Loops while

O loop while é usado para repetir uma instrução desde que uma condição em particular seja true. A seguir está um exemplo de loop while:

```
while (i < 10) {
   x = x * i++; // o corpo do loop
}
```

No exemplo anterior, a condição que acompanha a palavra-chave while é uma expressão booleana - i < 10. Se a expressão retorna true, o corpo do loop while é executado e é feito novo teste da condição. Esse processo se repete até que a condição seja false. Embora o loop precedente utilize chaves de abertura e fechamento para formar uma instrução de bloco, as chaves não são necessárias, pois o loop contém apenas uma instrução: x = x * i++. Contudo, o uso das chaves não cria nenhum problema e elas serão exigidas se você incluir outra instrução dentro do loop, posteriormente.

A listagem 5.5 mostra um exemplo de loop while que copia os elementos de um array de inteiros (em array1) para um array de floats (em array2), fazendo a coerção de cada elemento para um float, à medida que prossegue. O único inconveniente é que, se qualquer um dos elementos do primeiro array for 1, o loop será imediatamente interrompido nesse ponto.

DIGITE **Listagem 5.5** O texto completo de CopyArraywhile.java

```
1: class CopyArrayWhile {
2:    public static void main (String arguments[ ]) {
3:      int[ ] array1 = { 7, 4, 8, 1, 4, 1, 4 };
4:      float[ ] array2 = new float[array1.length];
5:
6:      System.out.print("array1: [ ");
7:      for (int i = 0; i < array1.length; i++) {
8:        System.out.print(array1[i] + " ");
9:      }
10:     System.out.println("]");
11:
12:     System.out.print("array2: [");
13:     int count = 0;
14:     while ( count < array1.length && array1[count] != 1) {
15:       array2[count] = (float) array1[count];
16:       System.out.print(array2[count++] + " ");
17:     }
18:     System.out.println(']");
19:   }
20: }
```

A saída do programa é a seguinte:

SAÍDA

```
arrayl:  [ 7 4 8 1 4 1 4 ]
array2:  [ 7.0 4.0 8.0 ]
```

Aqui está o que acontece no método mai n ( ):

- As linhas 3 e 4 declaram os arrays. arrayl é um array de inteiros, que são inicializados com alguns números convenientes. array2 é um array de números em ponto flutuante que tem o mesmo comprimento de arrayl, mas não possui valores iniciais.
- As linhas 6 a 10 servem para fins de saída; elas simplesmente fazem a iteração através do arrayl usando um loop for para imprimir seus valores.
- As linhas 13 a 17 é onde acontecem as coisas interessantes. Essas instruções atribuem os valores de array2 (convertendo-os para números em ponto flutuante ao longo do array) e os imprimem ao mesmo tempo. Você começa com uma variável count, que controla os elementos do índice do array. O teste no loop while controla as duas condições para a existência do loop, onde essas duas condições são a ausência de elementos em arrayl ou o encontro do valor 1 em arrayl. (Lembre-se de que isso fazia parte da descrição original do que esse programa faz.)

Você pode usar a condicional lógica && para controlar o teste; lembre-se de que && certifica-se de que as duas condições sejam true, antes que a expressão inteira seja true. Se uma das duas for fal se, a expressão retornará fal se e o loop será abandonado. O que ocorre nesse exemplo em particular? A saída mostra que os quatro primeiros elementos de arrayl foram copiados para array2, mas havia um valor 1 no meio, que impediu o loop de prosseguir. Sem o valor 1, array2 deveria terminar com todos os elementos de arrayl. Se o teste do loop whi 1 e for inicialmente fal se na primeira vez em que for realizado (por exemplo, se o primeiro elemento desse primeiro array for 1), o corpo do loop while nunca será executado. Se você precisa executar o loop pelo menos uma vez, pode fazer duas coisas:

- Duplicar o corpo do loop fora do loop whi 1 e.
- Usar um loop do (o que está descrito na seção a seguir).

A utilização do loop do é considerada a melhor solução das duas.

LOODS do. ..while

O loop do é exatamente como um loop whi 1 e, com uma diferença importante: o lugar no loop em que a condição é testada. Um loop whi 1 e testa a condição antes de realizar a repetição; portanto, se a condição for fal se na primeira vez

em que for testada, o corpo do loop nunca será executado. O corpo de um loop do é executado pelo menos uma vez, antes de testar a condição; portanto, se a condição for false na primeira vez em que for testada, o corpo do loop já terá sido executado pelo menos uma vez.

Essa é a diferença entre pedir ao pai para que empreste o carro e dizer depois a ele que você o tomou emprestado. Se o pai não gostar da idéia, no primeiro caso você não emprestará o carro. No segundo caso, se ele não gostar da idéia, você já terá tomado emprestado uma vez.

Os loops do são como segue:

```
do {
   x = x * i++; // o corpo do loop
} while (i < 10);
```

O corpo do loop é executado uma vez, antes que a condição de teste (i < 10) seja avaliada; então, se o teste for avaliado como true, o loop será executado novamente. Se ele for false, o loop será abandonado. Lembre-se de que o corpo do loop é executado pelo menos uma vez nos loops do.

A listagem 5.6 mostra um exemplo simples de loop do, que imprime uma mensagem sempre que o loop faz uma iteração (10 vezes, nesse exemplo):

DIGITE

Listagem 5.6 O texto completo de DoTest.java.

```
1: class DoTest {
2:   public static void main (String arguments[ ]) {
3:     int x = 1;
4:
5:     do {
6:       System.out.println("Looping, round " + x);
7:       x++;
8:     } while (x <= 10);
9:   }
10: }
```

A saída do programa é a seguinte:

SAÍDA

```
Looping, round 1
Looping, round 2
Looping, round 3
Looping, round 4
Looping, round 5
Looping, round 6
Looping, round 7
Looping, round 8
Looping, round 9
Looping, round 10
```

Saindo de LOOPS

Todos os loops terminam quando uma condição testada é satisfeita. Podem existir ocasiões em que algo ocorre durante a execução de um loop e você quer sair dele mais cedo. Para isso, você pode usar as palavras-chave break e continue.

Você já viu a palavra-chave break como parte da instrução switch; break interrompe a execução da instrução switch e o programa continua. A palavra-chave break, quando usada com um loop, realiza a mesma coisa — ela interrompe a execução do loop atual, imediatamente. Se você tem loops aninhados dentro de loops, a execução recomeça no próximo loop mais externo. Caso contrário, o programa simplesmente continua a ser executado na próxima instrução após o loop.

Por exemplo, lembre-se do loop while e que copiava elementos de um array de inteiros para um array de números em ponto flutuante, até que o final do array ou um valor 1 fosse encontrado. Você pode testar este último caso dentro do corpo do loop while e depois usar break para sair do loop:

```
int count = 0;
while (count < userData1.length) {
    if (userData1[count] == 1)
        break;
    userData2[count] = (float) userData1[count++];
}
```

A palavra-chave continue inicia o loop na próxima iteração. Para os loops do e while, isso significa que a execução da instrução de bloco começa novamente; no caso de loops for, a expressão de incremento é avaliada e, em seguida, a instrução de bloco é executada. A palavra-chave continue é útil quando você quer fazer um caso especial de elementos dentro de um loop. Com o exemplo anterior de cópia de um array em outro, você poderia testar se o elemento atual é igual a 1 e usar continue para reiniciar o loop após cada 1, para que o array resultante nunca contenha zero. Observe que, como você está pulando elementos no primeiro array, agora tem de controlar dois contadores de array diferentes:

```
int count = 0;
int count2 = 0;
while (count++ <= userData1.length) {
    if (userData1[count] == 1)
        continue;

    userData2[count2++] = (float)userData1[count];
)
```

LOOPS rotulados

As palavras-chave break e continue podem ter um rótulo opcional que informa à linguagem Java onde deve retomar a execução do programa. Sem um rótulo, a palavra-chave break pula para o loop mais próximo em um loop incluso ou

para a próxima instrução fora do loop. A palavra-chave conti nue reinicia o loop dentro do qual está inclusa. A utilização de break e continue com um rótulo permite que você use a palavra-chave break para ir até um ponto fora de um loop aninhado ou a palavra-chave continue para ir até um loop fora do loop atual.

Para usar um loop rotulado, inclua o rótulo antes da parte inicial do loop, com dois-pontos entre o rótulo e o loop. Então, quando você usar break ou continue, inclua o nome do rótulo após a palavra-chave em si, como no seguinte:

```
out:
    for (int i = 0; i <10; i++) {
        while (x < 50) {
            if (i * x++ > 400)
                break out;
            // loop mais interno aqui
    }
    // loop mais externo aqui
}
```

Nesse fragmento de código, o rótulo out designa o loop mais externo. Então, dentro dos loops for e while, quando uma condição em particular for satisfeita, um break fará a execução sair dos dois loops. Sem o rótulo out, a instrução break interromperia o loop mais interno e retomaria a execução no loop mais externo.

A seguir há outro exemplo: o programa apresentado na listagem 5.7 contém um loop for aninhado. Dentro do loop mais interno, se os valores dos dois contadores somados for maior do que 4, os dois loops serão interrompidos simultaneamente.

DIGITE

Listagem 5.7 O texto completo de LabelTest.java.

```
1: class LabelTest {
2:    public static void main (String arguments[ ]) {
3:
4:    thisLoop:
5:       for (int i = 1; i <= 5; i++)
6:       for (int j - 1; j <- 3; j++) {
7:          System.out.println("i is " + i + ", j is " + j);
8:          if (( i + j) > 4)
9:          break thisLoop;
10:      }
11:      System.out.println("end of loops");
12:   }
13: }
```

A saída do programa é a seguinte:

SAÍDA

```
i is 1, j is 1
i is 1, j is 2
i is 1, j is 3
i is 2, j is 1
i is 2, j is 2
i is 2, j is 3
end of loops
```

Conforme você pode ver, o loop fez a iteração até que a soma de i e j fosse maior do que 4 e, em seguida, os dois loops saíram para o bloco mais externo e a mensagem final foi apresentada.

Resumo

Agora que você foi apresentado às listas, loops e lógica, pode fazer um computador decidir se vai apresentar repetidamente o conteúdo de um array.

Você aprendeu a declarar uma variável de array, atribuir um objeto a ela e acessar e alterar elementos do array. Com as instruções condicionais if e switch, você pode desviar para diferentes partes de um programa com base em um teste booleano. Você aprendeu sobre os loops for, while e do, cada um deles permitindo que uma parte de um programa seja repetida até que determinada condição seja satisfeita.

Vale repetir: você usará todos esses três recursos freqüentemente em seus programas Java.

Você usará todos esses três recursos freqüentemente em seus programas Java.

Perguntas e respostas

P **Eu declarei uma variável dentro de uma instrução de bloco para um if. Quando o if terminou, a definição dessa variável desapareceu. Para onde ela foi?**

R Em termos técnicos, as instruções de bloco formam um novo *escopo léxico*. O que isso significa é que, se você declarar uma variável dentro de um bloco, ela será visível e utilizável apenas dentro desse bloco. Quando terminar a execução do bloco, todas as variáveis declaradas se vão.

É uma boa idéia declarar a maioria de suas variáveis no bloco mais externo em que elas serão necessárias — normalmente, no início de uma instrução de bloco. As exceções poderiam ser as variáveis muito simples, como os contadores de índice em loops for, onde a declaração delas na primeira linha do loop é um atalho **fácil.**

P **Por que** você **não usa switch com strings?**

R Os strings são objetos e, na linguagem Java, switch trabalha apenas com os tipos primitivos byte, char, short e int. Para comparar strings, você precisa usar instruções i f aninhadas, o que permite testes de expressão mais gerais, incluindo a comparação de strings.

SEMANA 1

Dia 6

Criando classes

Se você está vindo para a linguagem Java partindo de outra linguagem de programação, poderá estar entrando em uma luta de classes. O termo "classe" parece sinônimo do termo "programa", mas você poderia estar em dúvida quanto ao relacionamento entre os dois.

Em Java, um programa é constituído de uma classe principal e todas as outras classes que são necessárias para oferecer suporte à principal. Essas classes de suporte incluem todas aquelas da biblioteca de classe da linguagem Java de que você poderia precisar (como String, Math e outras).

Hoje, você verá a classe em termos do que conhece sobre o assunto. Você criará classes e lera a respeito dos seguintes tópicos:

- As partes de uma definição de classe
- A criação e o uso de variáveis de instância
- A criação e o uso de métodos
- O método main( ) utilizado em aplicativos Java
- A utilização de argumentos passados para um aplicativo Java

Definindo classes

Como você criou classes durante cada um dos capítulos anteriores, neste ponto já deve estar familiarizado com os fundamentos da definição de classe. Uma classe é definida através da palavra-chave cl ass, como no exemplo a seguir:

```
class Ticker {
   // corpo da classe
}
```

Por definição, as classes herdam da classe Object, que é a superclasse de todas as classes na hierarquia de classe Java.

Se sua classe é uma subclasse, a palavra-chave extends é usada para indicar a superclasse da nova classe. Veja a seguinte subclasse de Ticker:

```
class SportsTicker extends Ticker {
   // corpo da classe
}
```

Criando variáveis de instância e de classe

Quando você criar uma classe que herdar de uma superclasse, haverá algum tipo de comportamento que será necessário ser incluído para tornar a nova classe diferente da qual ela herda.

Esse comportamento é definido pela especificação das variáveis e métodos da nova classe. Nesta seção, você trabalhará com três tipos de variáveis: variáveis de classe, as de instância e as locais. A próxima seção entrará em detalhes sobre os métodos.

Definindo variáveis de instância

No Dia 3, você aprendeu a declarar e inicializar variáveis locais, que são aquelas que ficam dentro de definições de método. As variáveis de instância são declaradas e definidas quase do mesmo jeito que as variáveis locais. A principal diferença é sua posição na definição de classe. As variáveis serão consideradas como sendo de instância, se forem declaradas fora de uma definição de método. Normalmente, entretanto, a maioria das variáveis é definida imediatamente após a primeira linha da definição de classe. A listagem 6.1 contém uma definição simples da classe Jabberwock, que herda de sua superclasse Repti1e.

DIGITE **Listagem 6.1** O texto completo de Jabberwock. java.

```
1: class Jabberwock extends Reptile {
2:
3:    String color;
4:    String sex;
5:    boolean hungry;
6:    int age;
7: }
```

Essa definição de classe contém quatro variáveis. Como elas não estão definidas dentro de um método, tratam-se de variáveis de instância e são as seguintes:

- color A cor do jabberwock (por exemplo, laranja, amarelo, amarelo-limão ou castanho-avermelhado).
- sex Um string que indica o sexo do jabberwock.
- hungry Uma variável booleana que é true se o jabberwock estiver com fome e f alse, no caso contrário.
- age A idade do jabberwock, em anos.

Constantes

As variáveis são úteis quando você precisa armazenar informações que podem ser alteradas na execução de um programa. Se o valor nunca deve mudar durante a execução do programa, você pode usar um tipo especial de variável, chamado constante.

NOVO TERMO

Uma *constante*, que também é chamada de *variável constante*, é uma variável com um valor que nunca muda. Isso poderia parecer uma designação errônea, dado o significado da palavra "variável .

As constantes são úteis na definição de valores compartilhados por todos os métodos de um objeto — em outras palavras, para se atribuir nomes significativos a valores de objetos imutáveis. Na linguagem Java, você pode criar constantes para todos os tipos de variáveis: de instância, de classe e locais.

Nota

As variáveis locais constantes não eram possíveis na versão 1.02 da linguagem Java, mas foram incluídas na versão 1.1. Isso se torna importante, se você estiver tentando criar um applet que seja completamente compatível com Java 1.02. Você aprenderá mais sobre isso durante a Semana 2.

Para declarar uma constante, use a palavra-chave final antes da declaração de variável e inclua um valor inicial para essa variável, como nos exemplos a seguir:

```
final float pi = 3.141592;
final boolean debug = false;
final int numberOfJenny = 8675309;
```

As constantes podem ser úteis para se nomear vários estados de um objeto e depois testá-los. Suponha que você tenha um rótulo de texto que possa ser alinhado à esquerda, à direita ou ficar centralizado. Você pode definir esses valores como inteiros constantes:

```
final int LEFT = 0;
final int RIGHT = 1;
final int CENTER = 2;
```

Como local para o armazenamento do valor do alinhamento atual do texto, a variável alignment é declarada como um int:

```
int alignment;
```

Posteriormente, no corpo de uma definição de método, você pode definir o alinhamento com o seguinte:

```
this.alignment = CENTER;
```

Você também poderá testar determinado alinhamento:

```
switch (this.alignment) {
   case LEFT:
      // trata do alinhamento à esquerda
      break;
   case RIGHT:
      // trata do alinhamento ã direita
      break;
   case LEFT:
      // trata do alinhamento centralizado
      break;
}
```

O uso freqüente de constantes torna um programa mais fácil de entender. Para ilustrar esse ponto, considere qual das duas instruções é mais informativa de sua função:

```
this.alignment = CENTER;

this.alignment = 2;
```

Variáveis de classe

Conforme você aprendeu em lições anteriores, as variáveis de classe se aplicam a uma classe como um todo, em vez de serem armazenadas individualmente em objetos da classe.

As variáveis de classe servem bem para a comunicação entre diferentes objetos da mesma classe ou para se controlar informações em toda a classe entre um conjunto de objetos.

A palavra-chave static é usada na declaração de classe para declarar uma variável de classe, como no seguinte:

```
static int sum;
static final int maxObjects = 10;
```

Criando métodos

Conforme você aprendeu no Dia 4, os métodos definem o comportamento de um objeto — tudo o que acontece quando o objeto é criado e as diversas tarefas que ele pode realizar durante sua existência.

Esta seção apresenta a definição de método e como os métodos funcionam. A lição de amanhã trará outros detalhes mais avançados do que se pode fazer com os métodos.

Definindo métodos

As definições de método têm quatro partes básicas:

- O nome do método.
- O tipo de objeto ou tipo primitivo retornado pelo método.
- Uma lista de parâmetros.
- O corpo do método.

As três primeiras partes da definição de método formam o que se chama *assinatura* do método.

Para manter as coisas simples no dia de hoje, duas partes opcionais da definição de método foram omitidas: um modificador, como publ ic ou private, e a palavra-chave throws, que indica as exceções que um método pode gerar. Você aprenderá sobre essas partes de uma definição de método durante a Semana 3.

Em outras linguagens, o nome do método — que poderia ser chamado de função, sub-rotina ou procedure — é suficiente para distingui-lo de outros métodos no programa.

Em Java, você pode ter vários métodos na mesma classe com o mesmo nome, mas com diferenças no tipo de retorno ou na lista de argumentos. Essa prática é chamada de *sobrecarga de método* e você aprenderá mais sobre isso amanhã.

A seguir há uma definição de método básica:

```
returnType methodName(typel argl, type2 arg2, type3 arg3 ...) {
    // corpo do método
}
```

O item returnType é o tipo primitivo ou a classe do valor retornado pelo método. Ele pode ser um dos tipos primitivos, um nome de classe ou void, se o método não retornar nenhum valor.

Note que, se esse método retornar um objeto de array, os colchetes do array poderão ficar após returnType ou após a lista de parâmetros. Como a primeira maneira é mais fácil de ler, ela é usada nos exemplos deste livro, como no seguinte:

```
int[ ] makeRange(int lower, int upper) {
    // corpo desse método
}
```

A lista de parâmetros do método é um conjunto de declarações de variável, separado por vírgulas, entre parênteses. Esses parâmetros se tornam variáveis locais no corpo do método, recebendo seus valores quando o método é chamado.

Dentro do corpo do método, você pode ter instruções, expressões, chamadas de método para outros objetos, condicionais, loops etc. — tudo sobre o que aprendeu a respeito nas lições anteriores.

A não ser que um método tenha sido declarado com um tipo de retorno void, ele retornará algum tipo de valor quando estiver concluído. Esse valor deve ser retornado explicitamente em algum ponto de saída dentro do método, usando-se a palavra-chave return.

A listagem 6.2 mostra um exemplo de uma classe que define um método makeRange( ). Esse método recebe dois inteiros — um limite inferior e um limite superior — e cria um array que contém todos os inteiros entre esses dois limites. Os próprios limites são incluídos no array de inteiros.

DIGITE

Listagem 6.2 O texto completo de RangeClass.java.

```
1: class RangeClass {
2:    int[ ] makeRange(int lower, int upper) {
3:       int arr[ ] = new int[ (upper - lower) + 1 ];
4:
5:       for (int i = 0; i < arr.length; i++) {
6:          arr[i] = lower++;
7:       }
8:       return arr;
9:    }
10:
11:   public static void main(String arguments[ ]) {
12:      int theArray[ ];
13:      RangeClass theRange = new RangeClass( );
14:
15:      theArray = theRange.makeRange(l, 10);
16:      System.out.print("The array: [ ");
17:      for (int i = 0; i < theArray.length; i++) {
18:         System.out.print(theArray[i] + " ");
19:      }
20:      System.out.println("]");
21:   }
22:
23: }
```

A saída do programa é a seguinte:

SAÍDA

```
The array: [ 1 2 3 4 5 6 7 8 9 10]
```

O método main( ) dessa classe testa o método makeRange( ) criando um intervalo no qual os limites inferior e superior são 1 e 10 respectivamente (ver linha 6) e, em seguida, usa um loop for para imprimir os valores do novo array.

A palavra-chave this

No corpo de uma definição de método, talvez você queira fazer referência ao objeto atual — o objeto em que o método foi chamado. Isso pode ser feito para se usar as variáveis de instância desse objeto ou passar o objeto corrente como um argumento para outro método.

Para fazer referência ao objeto atual nesses casos, use a palavra-chave this onde você normalmente faria referência ao nome de um objeto.

A palavra-chave this faz referência ao objeto atual e você pode usá-la em qualquer parte em que o objeto possa aparecer: em notação de ponto, como um argumento para um método, como o valor de retorno do método atual etc. A seguir estão alguns exemplos do uso de this:

```
t = this.x;              // a variável de instância x desse objeto
this.resetData(this);    // chama o método resetData, definido
                         // nessa classe, e passa para ele
                         // o objeto atual
return this;             // retorna o objeto atual
```


Em muitos casos, você pode não precisar utilizar explicitamente a palavra-chave this, pois ela será pressuposta. Por exemplo, você pode fazer referência às variáveis de instância e às chamadas de método definidas na classe atual simplesmente pelo nome, pois o this é implícito nessas referências. Portanto, você poderia escrever os dois primeiros exemplos como segue:

```
t = x;                   // a variável de instância x desse objeto
resetData(this);         // chama o método resetData, definido
                         // nessa classe
```

A viabilidade da omissão da palavra-chave this para variáveis de instância depende de as variáveis do mesmo nome serem declaradas no escopo local. Você verá mais sobre esse assunto na próxima seção.

Como this é uma referência à instância atual de uma classe, você só deve usá-la dentro do corpo de uma definição de método de instância. Os métodos de classe, aqueles declarados com a palavra-chave static, não podem usar this.

ESCODO de variável e definições de método

Um dos detalhes que você deve conhecer para usar uma variável é seu escopo.

NOVO TERMO

Escopo é a parte de um programa em que uma variável ou outra informação pode ser usada. Quando a parte que define o escopo tem a execução concluída, a variável deixa de existir.

Quando você declara uma variável na linguagem Java, essa variável sempre tem um escopo limitado. Por exemplo, uma variável com escopo local só pode ser utilizada dentro do bloco em que foi definida. As variáveis de instância e de classe têm um escopo que se estende para a classe inteira; portanto, elas podem ser usadas por qualquer um dos métodos de dentro de sua classe.

Quando você se refere a uma variável dentro de uma definição de método, a linguagem Java procura primeiro uma definição dessa variável no escopo atual (que poderia ser um bloco), em seguida, em cada escopo mais externo e, finalmente, até a definição de método atual. Se a variável não é local, então a linguagem Java procura uma definição dela como uma variável de instância ou de classe, na classe atual. Se a linguagem Java ainda não encontrar a definição da variável, ela pesquisará cada superclasse, por sua vez.

Por causa do modo como a linguagem Java verifica o escopo de determinada variável, é possível criar uma variável em um escopo menor, que oculte (ou substitua) o valor original da primeira e introduza bugs sutis e que causem confusão em seu código.

Por exemplo, observe o pequeno programa Java da listagem 6.3.

DIGITE

Listagem 6.3 O texto completo de ScopeTest.java.

```
1: class ScopeTest {2:    int test = 10;3:4:    void printTest ( ) {5:       int test =
20;6:       System.out.println("test = " + test);
7:    }
8:
9:    public static void main(String arguments[ ]) {
10:     ScopeTest st = new ScopeTest( );
11:     st.printTest( );
12:   }
13: }
```

A seguir está a saída desse programa:

SAÍDA

```
test = 20
```

Nessa classe, você tem duas variáveis com o mesmo nome e definição. A primeira, uma variável de instância, possui o nome `test` e é inicializada com o valor 10. A segunda é uma variável local com o mesmo nome, mas com o valor 20. A variável local `test` que está dentro do método printTest( ) oculta a

variável de instância test. O método printTest( ) que está dentro de main( ) apresenta essa saída test = 20. Você pode contornar esse problema, usando this.test para fazer referência à variável de instância e usar apenas test para fazer referência à variável local.

O conflito é evitado fazendo referência explicitamente à variável de instância, através de seu escopo de objeto.

Um exemplo mais traiçoeiro ocorre quando você redefine em uma subclasse uma variável que já ocorre em uma superclasse. Isso pode criar bugs sutis em seu código; por exemplo, você poderia chamar métodos destinados a alterar o valor de uma variável de instância, mas a variável errada seria alterada. Outro bug poderia ocorrer, quando você faz a coerção de um objeto de uma classe para outra; o valor de sua variável de instância poderia mudar misteriosamente, pois estaria recebendo esse valor da superclasse e não de sua classe.

A melhor maneira de evitar esse comportamento é estar ciente das variáveis definidas em todas as superclasses de sua classe. Isso evita que você duplique uma variável que já está sendo usada em um lugar mais alto na hierarquia de classe.

Passando argumentos para métodos

Quando você chama um método com parâmetros de objeto, os objetos são passados para o corpo do método por referência. O que você fizer com os objetos dentro do método afetará os objetos originais. Lembre-se de que tais objetos incluem arrays e todos os objetos neles contidos. Quando você passa um array para um método e modifica seu conteúdo, o array original é afetado. Os tipos primitivos, por outro lado, são passados por valor.

A listagem 6.4 demonstra como isso funciona.

DIGITE **Listagem 6.4** A classe PassByReference.

```
1: class PassByReference {
2:   int onetoZero(int arg[ ]) {
3:     int count = 0;
4:
5:     for (int i = 0; i < arg.length; i++) {
6:       if (arg[i] == 1) {
7:         count++;
8:         arg[i] = 0;
9:       }
10:    }
11:    return count;
12:  }
13:  public static void main(String arguments[ ]) {
14:    int arr[ ] = { 1, 3, 4, 5, 1, 1, 7 };
15:    PassByReference test - new PassByReference( );
16:    int numOnes;
```

Listagem 6.4 Continuação

```
17:
18:     System.out.print("Values of the array: [ ");
19:     for (int i = 0; 1 < arr.length; i++) {
20:       System.out.print(arr[i] + " ");
21:     }
22:     System.out.println["]");
23:
24:     numOnes = test.onetoZero(arr);
25:     System.out.println("Number of Ones = " + numOnes);
26:     System.out.print("New values of the array: [ ");
27:     for (int i = 0; i < arr.length; i++) {
28:       System.out.print(arr[i] + " ");
29:     }
30:     System.out.printlnC']");
31:   }
32: }
```

A seguir está a saída desse programa:

SAÍDA

```
Values of the array: [1 3 4 5 1 1 7]
Number of Ones = 3
New values of the array: [0 3 4 5 0 0 7]
```

Observe a definição do método onetoZero( ),nas linhas 2 a 12, que recebe um array como argumento. O método onetoZero( ) faz duas coisas:

- Ele conta o número de 1 no array e retorna esse valor.
- Ele substitui por um 0 cada elemento 1 no array.

O método main( ) da classe PassByReference testa a utilização do método onetoZero ( ). Examine o método mai n ( ) linha por linha para que você possa ver o que está acontecendo e por que a saída fica como foi apresentada.

As linhas 14 a 16 definem as variáveis iniciais desse exemplo. A primeira é um array de inteiros; a segunda é uma instância da classe PassByReference, que é armazenada no teste de variável. A terceira é um inteiro simples para conter o número de elementos iguais a 1 presentes no array.

As linhas 18 a 22 imprimem os valores iniciais do array; você pode ver o valor dessas linhas na primeira linha da saída.

A linha 24 é onde ocorre o trabalho real; é aí que você chama o método onetoZero ( ) definido no teste de objeto e passa para ele o array armazenado em arr. Esse método retorna o número de elementos 1 do array, que é então atribuído à variável numOnes.

Tudo bem até aqui? A linha 25 imprime o número de elementos iguais a 1 (o valor recebido do método onetoZero ( )). Ela retorna 3, conforme seria de se esperar.

A última seção de linhas imprime os valores do array. Como uma referência ao objeto de array é passada para o método, mudar o array dentro desse método altera a cópia original desse array. A impressão do valores nas linhas 27 a 30 prova isso — a última linha da saída mostra que todos os elementos iguais a 1 do array mudaram para 0.

Métodos de classe

O relacionamento entre variáveis de classe e de instância é diretamente comparável ao funcionamento dos métodos de classe e de instância.

Os métodos de classe estão disponíveis para qualquer instância da própria classe e podem se tornar disponíveis para outras classes. Além disso, ao contrário de um método de instância, uma classe não exige uma instância da classe para que seus métodos sejam chamados.

Por exemplo, as bibliotecas de classe Java incluem uma classe chamada Math. A classe Math define um conjunto de operações matemáticas que você pode usar em qualquer programa ou em qualquer um dos vários tipos numéricos, como no seguinte:

```
float root = Math.sqrt(453.0);

System.out.println("The larger of x and y is " + Math.max(x, y));
```


Para definir métodos de classe, use a palavra-chave static na frente da definição do método, exatamente como usaria a mesma palavra-chave na frente de uma variável de classe. Por exemplo, o método de classe max ( ) utilizado no exemplo anterior, poderia ter a seguinte assinatura:

```
static int maxfint argl, int arg2) {
   // corpo do método
}
```

A linguagem Java fornece classes wrapperpara cada um dos tipos básicos; por exemplo, a linguagem fornece as classes Integer, Fl oat e Bool ean. Usando os métodos de classe definidos nessas classes, você pode converter objetos em tipos primitivos e converter tipos primitivos em objetos.

Por exemplo, o método de classe parselnt( ) da classe Integer pode ser usado com um string. O string é enviado para o método como um argumento e isso é usado para calcular um valor de retorno para ser enviado de volta como um int.

A instrução a seguir mostra como o método parselnt( ) pode ser usado:

```
int count = Integer.parselnt("42");
```

Na instrução anterior, o valor de Stri ng "42" é retornado por parselnt( ) como um inteiro, com o valor 42 e esse valor é armazenado na variável count.

A falta da palavra-chave static na frente de um nome de método o transforma em um método de instância. Os métodos de instância operam sobre

um objeto em particular e não em uma classe de objetos. No Dia 2, você criou um método de instância chamado feedJabberwock( ) que alimentava um jabberwock específico.

A maior parte dos métodos que operam sobre um objeto em particular ou o afetam, deve ser definida como métodos de instância. Os métodos que fornecem algum recurso geral, mas não afetam diretamente uma instância da classe, devem ser declarados como métodos de classe.

Criando aplicativos Java

Agora que você sabe criar classes, objetos, variáveis de classe e instância e métodos de classe e instância, pode reunir tudo isso em um programa Java.

Os *aplicativos*, para refrescar sua memória, são programas Java que podem ser executados sozinhos. Os aplicativos são diferentes dos applets, que exigem um navegador compatível com a linguagem Java para vê-los. Os projetos que você criou até este ponto eram aplicativos Java. Você terá a chance de se aprofundar nos applets na próxima semana. Os applets exigem um pouco mais de conhecimento para se fazê-los interagir com o navegador, assim como desenhar e atualizar com o sistema gráfico.

Um aplicativo Java é composto de uma ou mais classes e pode ter o tamanho que você quiser. Embora todos os aplicativos Java que você criou até agora não façam nada, a não ser produzir a saída de alguns caracteres na tela ou em uma janela, você também pode criar aplicativos Java que utilizam janelas, imagens gráficas e elementos de interface com o usuário, exatamente como acontece com os applets.

Entretanto, a única coisa que você precisa para fazer um aplicativo Java ser executado é de uma classe que sirva como ponto de partida para o restante do programa.

A classe que serve de ponto de partida para seu aplicativo precisa de apenas uma coisa: um método main( ). Quando um aplicativo é executado, o método main( ) é o primeiro elemento a ser chamado. Nada disso deve ser surpresa para você, neste ponto; há muito que você está criando aplicativos Java com métodos main( ).

A assinatura do método main( ) é sempre semelhante à seguinte:

```
public static void main(String arguments[ ]) {
   // corpo do método
}
```

Aqui está um relato das partes do método main( ):

- public significa que esse método está disponível para outras classes e objetos. O método main( ) deve ser declarado como public. Você aprenderá mais sobre métodos public e private na Semana 3.

- static significa que main( ) é um método de classe.
- void significa que o método main( ) não retorna um valor.
- main( ) recebe um parâmetro, que é um array de strings. Esse argumento é usado para argumentos de programa, sobre o que você aprenderá na próxima seção.

O corpo do método main( ) contém todo o código necessário para iniciar seu aplicativo, como a inicialização de variáveis ou a criação de instâncias de classe.

Quando a linguagem Java executar o método main( ), lembre-se de que ele é um método de classe. Uma instância da classe que contém main( ) não é criada automaticamente, quando seu programa é executado. Se você quiser tratar essa classe como um objeto, então terá de criar uma instância dela no método main( ).

Classes auxiliadoras

Seu aplicativo Java pode ter apenas uma classe ou, no caso de programas maiores, ele poderia ser constituído de várias classes, nas quais diferentes instâncias de cada classe seriam criadas e utilizadas enquanto o aplicativo estivesse em execução. Você pode criar quantas classes quiser para seu programa.

Se você estiver usando o JDK, as classes devem ser acessíveis a partir de uma pasta que está listada em seu CLASSPATH.

Desde que a linguagem Java possa encontrar a classe, ela será usada por seu programa quando este for executado. Entretanto, observe que somente a classe de ponto de partida precisa de um método main . Depois de serem chamados, os métodos de dentro das várias classes e objetos utilizados em seu programa assumem o controle. Embora você possa incluir métodos main( ) em classes auxiliadoras, eles serão ignorados quando o programa for efetivamente executado.

Aplicativos Java e argumentos de linha de comando

Como os aplicativos Java são programas independentes, é interessante passar argumentos ou opções para um aplicativo. Você pode usar os argumentos para determinar como o aplicativo será executado ou permitir que um aplicativo genérico funcione com diferentes tipos de entrada. Você pode usar argumentos de programa para muitos propósitos diferentes, como para ativar entrada de depuração ou indicar um nome de arquivo a ser carregado.

Passando argumentos para aplicativos Java

O modo como você passa argumentos para um aplicativo Java varia de acordo com a plataforma em que Java está sendo executada. No Windows e no UNIX, você pode passar argumentos para o programa Java através da linha de comando.

Para se passar argumentos para um programa Java no Windows ou no Solaris, os argumentos devem ser anexados à linha de comando quando o programa estiver sendo executado. A seguir está um exemplo:

```
java MyProgram argumentOne 2 three
```

Nesses exemplos anteriores, três argumentos foram passados para um programa: argumentOne, o número 2 e three. Observe que um espaço separa cada um dos argumentos.

Para agrupar argumentos que incluem espaços, os argumentos devem ser colocados entre aspas. Por exemplo, o argumento "No Shoes No Shirt No Service" produz um argumento para um programa processar; as aspas impedem que os espaços sejam usados para separar um argumento de outro. Essas aspas não são incluídas no argumento, quando ele é enviado para o programa e recebido utilizando-se o método mai n ( ).

Tratando de araumentos em seu aolicativo Java

Quando um aplicativo é executado com argumentos, a linguagem Java os armazena como um array de strings e passa o array para o método mai n ( ) do aplicativo. Dê uma outra olhada na assinatura de mai n ( ):

```
public static void main(String arguments[ ]) {
   // corpo do método
}
```

Aqui, arguments é o nome do array de strings que contém a lista de argumentos. Você chama esse array como quiser.

Dentro do método main( ), você pode então manipular os argumentos recebidos pelo seu programa, fazendo uma iteração no array de argumentos e manipulando-os de alguma maneira. Por exemplo, a listagem 6.5 é uma classe simples, que imprime os argumentos que recebe, um por linha.

DIGITE **Listagem 6.5** O texto completo de EchoArgs.java.

```
1: class EchoArgs {
2:    public static void main(String arguments[ ]) {
3:      for (int i = 0; i < arguments.length; i++) {
4:        System.out.println("Argument " + i + ": " + arguments[i]);
5:      }
6:    }
7: }
```

A seguir está um exemplo de entrada para executar esse programa:

ENTRADA

```
java EchoArgs Wilhelm Niekro Hough 49
```

Se você executar o aplicativo EchoArgs com os argumentos de linha de comando precedentes, a seguinte saída será produzida:

SAÍDA

```
Argument 0: Wilhelm
Argument 1: Niekro
Argument 2: Hough
Argument 3: 49
```

A seguir há outro exemplo de entrada para executar esse programa:

ENTRADA

```
java EchoArgs "Hoyt Wilhelm" Charlie Hough
```

A saída que ela produz:

SAÍDA

```
Argument 0: Hoyt Wilhelm
Argument 1: Charlie
Argument 2: Hough
```

Observe como os argumentos são agrupados no segundo exemplo de entrada; a colocação de aspas em Hoyt Wilhelm faz com que esse argumento seja tratado como uma unidade dentro do array de argumentos.

Nota

O array de argumentos em Java não é análogo a argv de C e UNIX. Em particular, arg[0] ou arguments[0], o primeiro elemento no array de argumentos é o primeiro argumento de linha de comando, após o nome da classe — não é o nome do programa, como seria em C. Cuidado com isso, quando escrever seus programas Java.

Um detalhe importante a ser observado: todos os argumentos passados para um aplicativo Java são armazenados em um array de strings. Para tratá-los como algo diferente de strings, você precisa convertê-los.

Por exemplo, suponha que você tenha um programa Java simples, chamado SumAverage, que recebe qualquer número de argumentos numéricos e retorna a soma e a média desses argumentos. A listagem 6.6 mostra uma primeira passagem nesse programa. Não tente compilá-lo; apenas observe o código e veja se você consegue descobrir o que ele faz.

DIGITE

Listagem 6.6 Uma primeira tentativa com SumAverage.java.

```
1: class SumAverage {
2:   public static void main(String arguments[ ]) {
3:     int sum = 0;
4:
```

Listagem 6.6 Continuação

```
5:      for (int i = 0; i < arguments.length; i++) {
6:        sum += Integer.parseInt(argumentsfi]);
7:      }
8:
9:      System.out.println("Sum is: " + sum);
10:     System.out.println("Average is: " +
11:       (float)sum / arguments.length);
12:   }
13: }
```

À primeira vista, esse programa parece bastante simples — um loop for faz a iteração no array de argumentos, soma-os e depois a soma e a média são impressas como último passo.

Se você tentasse compilar esse programa, um erro como o seguinte seria apresentado pelo compilador:

SAÍDA

```
SumAverage.java:6: Incompatible type for +=. Can't convert
java.lang.String to
int.
      sum += arguments[i];
```

Esse erro ocorre porque o array de argumentos é um array de strings. Mesmo que você vá passar inteiros para o aplicativo, esses inteiros serão convertidos em strings antes de serem armazenados no array. Você precisa convertê-los de strings em inteiros, usando um método de classe para a classe Integer, chamado parseInt. Mude a linha 6 para usar esse método, como segue:

```
sum += Integer.parseInt(arguments[i]);
```

O programa será compilado com sucesso. Para executar esse programa, introduza o seguinte em um prompt de comando: java SumAverage 123. Quando você executar o aplicativo, deverá ver a seguinte saída:

SAÍDA

```
Sum is: 6
Average is: 2
```

Resumo

Após concluir o capítulo de hoje, você já deverá ter uma boa idéia do motivo pelo qual a linguagem Java tem classe. Tudo que você cria em Java envolve o uso de uma classe principal que interage com outras classes, quando necessário. Essa é uma perspectiva de programação diferente da que você poderia estar acostumado com outras linguagens de programação.

Hoje, você reuniu tudo o que aprendeu sobre a criação de classes Java. Cada um dos tópicos a seguir foi abordado:

- Variáveis de instância e de classe, que contêm atributos de uma classe e objetos criados a partir dela. Você aprendeu como declarar variáveis, como elas diferem das variáveis locais e como declarar constantes.
- Métodos de instância e de classe, que definem o comportamento de uma classe. Você aprendeu a definir métodos — incluindo as partes de uma assinatura de método, como retornar valores de um método, como os argumentos são passados para os métodos e como usar a palavra-chave this para fazer referência ao objeto atual.
- Aplicativos Java. Você aprendeu como o método main( ) funciona e como passar argumentos para um aplicativo Java.

Amanhã, você terminará a Semana 1, aprendendo sobre alguns aspectos avançados da programação com métodos.

Até lá, a classe está dispensada.

Perauntas e respostas

P **Você mencionou que as variáveis locais constantes não podem ser criadas em applets que são compatíveis com Java 1.02. Por que eu iria querer criar programas que não utilizam os recursos atuais da linguagem Java 2?**

R A razão mais provável é que você está tentando criar um applet que funcione no maior número de navegadores. Um suporte integral às versões da linguagem além da 1.02 ainda não ocorre em navegadores como o Netscape Navigator e o Microsoft Internet Explorer, embora a JavaSoft esteja trabalhando em meios para corrigir isso. A situação será totalmente discutida no Dia 8.

P **Em minha classe, eu tenho uma variável de instância chamada origin. Também tenho um variável local chamada origin em um método, que, devido ao escopo de variável, fica oculta pela variável local. Há algum modo de obter o valor da variável de instância?**

R O modo mais fácil é evitar a atribuição de nomes às suas variáveis locais, iguais aos de suas variáveis de instância. Se achar que deve, você pode usar this.origin para fazer referência à variável de instância e origin para fazer referência à variável local.

P **Escrevi um programa que recebe quatro argumentos, mas se eu forneço poucos argumentos, ele pára com um erro de execução.**

R Testar o número e o tipo de argumentos que seu programa espera fica por sua conta em seu programa Java; a linguagem Java não fará isso automaticamente. Se o seu programa exige quatro argumentos, teste o fato de você ter mesmo fornecido quatro argumentos e retorne uma mensagem de erro, se não tiver fornecido.

SEMANA 1

Dia 7

Usando métodos para realizar tarefas

Os métodos representam a parte mais importante de qualquer linguagem orientada a objetos, pois definem toda ação que um objeto realiza.

As classes e objetos fornecem uma estrutura. As variáveis de classe e de instância proporcionam um modo de descrever o que essas classes e objetos são. Apenas os métodos podem definir o comportamento de um objeto — o que ele é capaz de fazer e o modo como interage com outras classes e objetos.

Ontem, você aprendeu alguns detalhes sobre a definição de métodos e poderia utilizar esse conhecimento para criar programas Java. Entretanto, estariam faltando alguns recursos que tornam os métodos poderosos — recursos que tornam seus objetos e classes mais eficientes e fáceis de entender. Hoje, você aprenderá a respeito dos seguintes recursos:

- *Sobrecarga de métodos.* A criação de métodos com várias assinaturas e definições, mas com o mesmo nome
- *Criação de métodos construtores.* Métodos que permitem inicializar objetos para estabelecer seu estado inicial, quando criado
- *Anulação de métodos.* A criação de uma definição diferente para um método que foi definido em uma superclasse
- *Métodos finalizadores.* Métodos que fazem a limpeza após um objeto, antes que ele seja removido do sistema

Criando métodos com o mesmo nome e araumentos diferentes

Quando você trabalha com a biblioteca de classe da linguagem Java, freqüentemente encontra classes que possuem numerosos métodos com o mesmo nome. Por exemplo, a classe java. 1ang. Stri ng possui vários métodos value0f ( ) diferentes.

Os métodos com o mesmo nome são diferenciados uns dos outros por dois detalhes:

- O número de argumentos que recebem.
- O tipo de dado ou objeto de cada argumento.

Esses dois detalhes compõem a assinatura de um método e o uso de vários métodos com o mesmo nome e assinaturas diferentes se chama *sobrecarga.*

No exemplo da classe String, os diferentes métodos value0f( ) sobrecarregados recebem tipos de dados diferentes como parâmetro.

A sobrecarga de método elimina a necessidade de métodos inteiramente diferentes que fazem basicamente a mesma coisa. A sobrecarga também torna possível ter métodos que se comportam de maneiras diferentes de acordo com os argumentos que recebem.

Os métodos value0f( ) sobrecarregados da classe String podem ser usados para a conversão de uma variedade de tipos de dados e objetos diferentes em valores Stri ng. Quando você chama um método em um objeto, a linguagem Java combina o nome do método e os argumentos para escolher qual definição de método vai ser executada.

Para criar um método sobrecarregado, você cria diferentes definições de método em uma classe, cada uma com o mesmo nome, mas com listas de argumentos diferentes. A diferença pode ser o número, o tipo de argumentos ou ambos. A linguagem Java permite a sobrecarga de método desde que cada lista de argumentos seja única para o mesmo nome de método.

Observe que a linguagem Java não considera o tipo de retorno ao diferenciar entre métodos sobrecarregados. Se você tentar criar dois métodos com a mesma assinatura e tipos de retorno diferentes, a classe não será compilada. Além disso, os nomes de variável escolhidos para cada argumento do método são irrelevantes — o que importa é o número e o tipo de argumento.

O próximo projeto é a criação de um método sobrecarregado. A listagem 7.1 mostra uma definição simples para uma classe chamada MyRect, que define uma figura retangular. A classe MyRect define uma figura retangular com quatro variáveis de instância para estabelecer os cantos superior esquerdo e inferior direito do retângulo: xl, yl, x2 e y2.

DIGITE **Listagem 7.1** O início de MyRect.java.

```
1: class MyRect {
2:    int x1 = 0;
3:    int y1 = 0;
4:    int x2 = 0;
5:    int y2 = 0;
6: }
```

Nota

Embora você possa compilar com sucesso esse programa, ele não pode ser executado, pois não existe um método main( ). Quando você tiver acabado de criar essa definição de classe, a versão final poderá ser compilada e executada.

Quando uma nova instância da classe MyRect é criada, todas as suas variáveis de instância são inicializadas como 0. Você pode incluir um método de instância buildRect( ) para definir as variáveis com seus valores corretos, como dois cantos de um retângulo. Esse método recebe quatro argumentos inteiros e retorna o objeto retângulo resultante. Como os argumentos possuem os mesmos nomes que as variáveis de instância, a palavra-chave this é usada dentro do método, ao se fazer referência às variáveis de instância. A seguir está o código de buildRect( ):

```
MyRect buildRect(int xl, int yl, int x2, int y2) {
   this.xl = xl;
   this.yl = yl;
   this.x2 = x2;
   this.y2 = y2;
   return this;
}
```

Esse método pode ser usado para criar retângulos — mas e se você quisesse definir as dimensões de um retângulo de um modo diferente? Uma alternativa seria usar objetos Point em vez das coordenadas específicas. Para implementar essa alternativa, você pode sobrecarregar buildRect( ) para que sua lista de argumentos receba dois objetos Point. Veja o seguinte:

```
MyRect buildRect(Point topLeft, Point bottomRight) {
   xl = topLeft.x;
   yl = topLeft.y;
   x2 = bottomRight.x;
   y2 = bottomRight.y;
   return this;
}
```

Para que o método anterior funcione, a classe Point deve ser importada no início do arquivo-fonte para que Java possa encontrá-la.

Outra possível maneira de definir o retângulo é usar um canto superior, uma altura e uma largura. A seguir está uma definição de um método sobrecarregado:

```
MyRect buildRect(Point topLeft, int w, int h) {
   xl = topLeft.x;
   yl = topLeft.y;
   x2 = (xl + w);
   y2 = (y1 + h);
   return this;
}
```

Para concluir esse exemplo, printRect( ) é criado para apresentar as coordenadas do retângulo e um método main( ) submete tudo à prova. A listagem 7.2 mostra a definição de classe completa, com todos os seus métodos: três métodos buildRect( ), um método printRect( ) e um método main( ).

DIGITE

Listagem 7.2 O texto completo de MyRect.java.

```
1: import java.awt.Point;
2:
3: class MyRect {
4: int xl = 0;
5: int yl = 0;
6: int x2 = 0;
7: int y2 = 0;
8:
9:   MyRect buildRect(int xl, int yl, int x2, int y2) {
10:    this.xl = xl;
11:    this.yl = yl;
12:    this.x2 = x2;
13:    this.y2 = y2;
14:    return this;
15: }
16:
17:  MyRect buildRect(Point topLeft, Point bottomRight) {
18:    xl = topLeft.x;
19:    yl = topLeft.y;
20:    x2 = bottomRight.x;
21:    y2 = bottomRight.y;
22:    return this;
23: }
24:
25:  MyRect buildRect(Point topLeft, int w, int h) {
26:    xl = topLeft.x;
27:    yl = topLeft.y;
28:   x2 = (xl + w);
29:   y2 = (yl + h);
30:    return this;
31: }
```

Listagem 7.2 Continuação

```
32:
33:   void printRect( ){
34:     System.out.print("MyRect: <" + x1 + ", " + y1);
35:     System.out.println(", " + x2 + ", " + y2 + ">");
36:   }
37:
38:   public static void main(String arguments[ ]) {
39:     MyRect rect = new MyRect( );
40:
41:      System.out.println("Calling buildRect with coordinates 25,25, 50,50:");
42:     rect.buildRect(25, 25, 50, 50);
43:      rect.printRect( );
44:      System.out.println("***");
45:
46:      System.out.println("Calling buildRect with points (10,10), (20,20):");
47:     rect.buildRect(new Point(10,10), new Point(20,20));
48:      rect.printRect( );
49:     System.out.println("***");
50:
51:      System.out.print("Calling buildRect with 1 point (10,10),");
52:    System.out.println("    width (50) and height (50):");
53:
54:     rect.buildRect(new Point(10,10), 50, 50);
55:      rect.printRect( );
56:      System.out.println("***");
57:   }
58: }
```

A seguir está a saída do programa:

SAÍDA

```
Calling buildRect with coordinates 25,25, 50,50:
MyRect: <25, 25, 50, 50>
***

Calling buildRect with points (10,10), (20,20):
MyRect: <10, 10, 20, 20>
***

Calling buildRect with 1 point (10,10), width (50) and height (50):
MyRect: <10, 10, 60, 60>
```

Conforme você pode ver nesse exemplo, todos os métodos bui 1 dRect ( ) funcionam baseados nos argumentos com que eles foram chamados. Você pode definir quantas versões de um método precisar para implementar o comportamento necessário para essa classe.

Quando você possui vários métodos que realizam coisas semelhantes, usar um único método para chamar outro é uma técnica de atalho a ser

considerada. Por exemplo, o método buildRect ( ), nas linhas 17 a 23, pode ser substituído pelo seguinte método, muito mais curto:

```
MyRect bui1dRect(Point, topLeft, Point bottomRight) {
    return buildRect(topLeft.x, topLeft.y,
        bottomRight.x, bottomRight.y);
}
```

Nesse método, a instrução return chama o método bui 1 dRect ( ) das linhas 9 a 15, com quatro argumentos inteiros. Ela produz os mesmos resultados com menos instruções.

Métodos construtores

Além dos métodos normais, você também pode definir métodos construtores em sua definição de classe.

NOVO TERMO

Um *método construtor* é aquele que é chamado em um objeto, quando ele é criado — em outras palavras, quando ele é construído.

Ao contrário dos outros métodos, um construtor não pode ser chamado diretamente. Em vez disso, a linguagem Java chama os métodos construtores automaticamente.

Java faz três coisas quando new é usado para criar uma instância de uma classe:

- Aloca memória para o objeto.
- Inicializa as variáveis de instância desse objeto, ou com os valores iniciais ou com um padrão (0 para números, nul 1 para objetos, fal se para booleanos ou '\0' para caracteres).
- Chama o método construtor da classe, que pode ser um dentre vários métodos.

Se uma classe não tem nenhum método construtor definido, um objeto ainda é criado quando a instrução new é utilizada em conjunto com a classe. Entretanto, talvez você tenha de definir suas variáveis de instância ou chamar outros métodos de que o objeto precisa para ser inicializado. Todos os exemplos que você criou até este ponto se comportaram desse modo.

Definindo métodos construtores em suas próprias classes, você pode estabelecer valores iniciais de variáveis de instância, chamar métodos baseados nessas variáveis, chamar métodos em outros objetos e definir as propriedades iniciais de um objeto. Você também pode sobrecarregar métodos construtores, como faz no caso de métodos normais, para criar um objeto que possua propriedades específicas baseadas nos argumentos fornecidos a new.

Métodos construtores básicos

Os construtores são praticamente iguais aos métodos normais, com duas diferenças básicas:

- Os métodos construtores sempre têm o mesmo nome da classe.
- Os métodos construtores não possuem um tipo de retorno.

Por exemplo, a listagem 7.3 mostra uma classe simples chamada Person. A classe Person usa um método construtor para inicializar suas variáveis de instância de acordo com argumentos de new. A classe também inclui um método para o objeto ser apresentado.

DIGITE

Listagem 7.3 A classe Person.

```
1: class Person {
2:      String name;
3:    int age;
4:
5:      Person(String n, int a) {
6:          name = n;
7:        age = a;
8:      }
9:
10:     void printPerson( ) {
11:        System.out.print("Hi, my name is " + name);
12:        System.out.println(". I am " + age + " years old.");
13:    }
14:
15:     public static void main (String arguments[ ]) {
16:         Person p;
17:        p = new Person("Luke", 50);
18:         p.printPerson( );
19:         System.out.println("___" ) ;
20:        p = new Person("Laura", 35);
21:         p.printPerson( );
22:         System.out.println("____");
23:     }
24: }
```

A seguir está a saída desse programa:

SAÍDA

```
Hi, my name is Luke. I am 50 years old.

Hi, my name is Laura. I am 35 years old.
```

A classe Person possui três métodos. O primeiro é o método construtor, definido nas linhas 5 a 8, que inicializa as duas variáveis de instância da classe,

com base nos argumentos do construtor Person( ). A classe Person também inclui um método chamado printPerson( ), para que o objeto possa "se apresentar", e um método main( ) para testar cada uma dessas coisas.

Chamando outro método construtor

Anteriormente, no dia de hoje, você viu como um método poderia realizar o trabalho de outro — um método buildRect( ) chamou outro método buildRect( ).

Você pode fazer o mesmo com qualquer tipo de método, incluindo os construtores. Se você tem um método construtor que duplica algum comportamento de um método construtor já existente, pode chamar o primeiro de dentro do corpo do segundo. A linguagem Java fornece uma sintaxe especial para se fazer isso. Use o seguinte para chamar um método construtor definido na classe atual:

```
this(argl, arg2, arg3);
```

A utilização de this com um método construtor é semelhante a como thi s pode ser usado para acessar as variáveis de um objeto atual. Na instrução anterior, os argumentos com this( ) são os argumentos do método construtor. Por exemplo, considere uma classe simples que define um círculo, usando as coordenadas (x,y) de seu centro e o comprimento de seu raio. A classe, MyCircle, poderia ter dois construtores: um onde o raio é definido e outro onde o raio é ajustado para o valor padrão 1. O código a seguir poderia ser usado:

```
class MyCircle (
   int x, y, radius;

   MyCircle(int xPoint, int yPoint, int radiusLength) {
      this.x = xPoint;
      this.y = yPoint;
      this.radius = radiusLength;
   )

   MyCircle(int xPoint, int yPoint) {
      this(xPoint, yPoint, 1);
   }
}
```

O segundo construtor em MyCircle recebe apenas as coordenadas x e y do centro do círculo. Como nenhum raio foi definido, o valor padrão 1 é utilizado. O primeiro construtor é chamado com xPoint, yPoint e a literal inteira 1, todos como argumentos.

Sobrecarreaando métodos construtores

Assim como os métodos normais, os métodos construtores também podem receber diversos números e tipos de parâmetros. Essa capacidade permite que

você crie um objeto exatamente com as propriedades desejadas ou que o objeto calcule propriedades a partir de diferentes tipos de entrada.

Por exemplo, os métodos bui 1 dRect ( ) que você definiu na classe MyRect, anteriormente no dia de hoje, seriam excelentes construtores, pois eles estão sendo usados para inicializar as variáveis de instância de um objeto com os valores apropriados. Por exemplo, em vez do método bui1dRect ( ) original definido (que recebia quatro parâmetros para as coordenadas dos cantos), você poderia criar um construtor.

A listagem 7.4 mostra uma nova classe, MyRect2, que tem a mesma funcionalidade da MyRect original, exceto que utiliza métodos construtores sobrecarregados em vez de métodos bui1dRect ( ) sobrecarregados. A saída apresentada no final é a mesma, embora o código para produzi-la tenha mudado.

DIGITE **Listagem 7.4** O texto completo de MyRect2.java.

```
1: import java.awt.Point;
2:
3: class MyRect2 {
4:   int xl = 0;
5:   int yl = 0;
6:   int x2 = 0;
7:   int y2 = 0;
8:
9:   MyRect2(int xl, int yl, int x2, int y2) {
10:   this.xl • xis
11:   this.yl = yl;
12:   this.x2 = x2;
13:   this.y2 = y2;
14: }
15:
16:  MyRect2(Point topLeft, Point bottomRight) {
17:   xl = topLeft.x;
18:   yl = topLeft.y;
19:   x2 = bottomRight.x;
20:   y2 = bottomRight.y;
21: }
22:
23:  MyRect2(Point topLeft, int w, int h) {
24:   xl = topLeft.x;
25:   yl = topLeft.y;
26:  x2 = (xl + w);
27:  y2 = (yl + h);
28: }
29:
30:  void printRect( ) {
31;   System.out.print("MyRect: <" + xl + ", " + yl);
32:   System.out.println(", " + x2 + ", " + y2 + ">");
```

Listagem 7.4 Continuação

```
33:     }
34:
35:     public static void main(String arguments[ ]) {
36:       MyRect2 rect;
37:
38:       System.out.println("Calling MyRect2 with coordinates 25,25 50,50:");
39:       rect = new MyRect2(25, 25, 50,50);
40:        rect.printRect( );
41:       System.out.println("***");
42:
43:       System.out.println("Calling MyRect2 with points (10,10), (20,20):");
44:       rect= new MyRect2(new Point(10,10), new Point(20,20));
45:        rect.printRect( );
46:       System.out.println("***");
47:
48:       System.out.print("Calling MyRect2 with 1 point (10,10)");
49:       System.out.printlnC width (50) and height (50):");
50:       rect = new MyRect2(new Point(10,10), 50, 50);
51:        rect.printRect( );
52:       System.out.println("***");
53:
54:   }
55: }.
```

O programa produz a seguinte saída:

SAÍDA

```
Calling MyRect2 with coordinates 25,25, 50,50:
MyRect: <25, 25, 50, 50>

Calling MyRect2 with points (10,10), (20,20):
MyRect: <10, 10, 20, 20>
***
Calling MyRect2 with 1 point (10,10), width (50) and height (50):
MyRect: <10, 10, 60, 60>
***
```

Anulando métodos

Quando você chama o método de um objeto, a linguagem Java procura essa definição de método na classe do objeto. Se não encontrar, ela passa a chamada de método para a hierarquia de classe até que uma definição de método seja encontrada. A herança de método permite que você defina e utilize métodos repetidamente em subclasses sem ter de duplicar o código.

Entretanto, podem existir ocasiões em que você queira que um objeto responda aos mesmos métodos, mas tenha comportamento diferente quando esse método for chamado. Nesse caso, você pode anular o método. Para anular

um método, defina um método em uma subclasse com a mesma assinatura de um método em uma superclasse. Em seguida, quando o método é chamado, o método da subclasse é encontrado e executado, em vez do que está na superclasse.

Criando métodos aue anulam métodos existentes

Para anular um método, basta criar outro em sua subclasse que tenha a mesma assinatura (nome, tipo de retorno e lista de argumentos) que um método definido pela superclasse de sua classe. Como a linguagem Java executa o primeiro método encontrado que combine com a assinatura, a nova assinatura oculta a definição do método original.

A seguir há um exemplo simples. A listagem 7.5 mostra uma classe simples com um método chamado printMe( ), que imprime o nome da classe e os valores de suas variáveis de instância.

DIGITE **Listagem 7.5** O texto completo de PrintClass.java.

```
1: class PrintClass {
2:     int x = 0;
3:     int y = 1;
4:
5:     void printMe( ) {
6:         System.out.println("x is " + x + ", y is " + y);
7:         System.out.println("I am an instance of the class " +
8:          this.getClass( ).getName( ));
9:     }
10: }
```


A listagem 7.6 mostra uma classe chamada PrintSubClass, que é uma subclasse de PrintClass. A única diferença entre PrintClass e PrintSubClass é que esta última tem uma variável de instância z.

DIGITE **Listagem 7.6** O texto completo de PrintSubClass.java.

```
1: class PrintSubClass extends PrintClass {
2:     int z = 3;
3:
4:     public static void main(String argumentsf ]) {
5:         PrintSubClass obj • new PrintSubClass( );
6:         obj.printMe( );
7:     }
8: }
```

A saída do programa é a seguinte:

SAÍDA

```
x is 0, y is 1
I am an instance of the class PrintSubClass
```

Um objeto PrintSubClass foi criado e o método printMe( ) foi chamado no método main( ) de PrintSubClass. Como PrintSubClass não define esse método, a linguagem Java procura por ele nas superclasses de PrintSubClass, começando com PrintClass. PrintClass tem um método printMe( ); portanto, ele é executado. Infelizmente, esse método não apresenta a variável de instância z, conforme você pode ver na saída anterior.

Noto

Há uma característica importante de PrintClass a ser apontada: essa classe não possui um método main( ). Ela não precisa; trata-se de um aplicativo. PrintClass é simplesmente uma classe utilitária da classe PrintSubClass, que é um aplicativo e, portanto, possui um método main( ). Apenas a classe com que você está realmente executando o interpretador Java precisa de um método main( ).

Agora, crie uma terceira classe. PrintSubClass2 é quase idêntica a PrintSubClass, mas o método printMe( ) é anulado para incluir a variável z. A listagem *7.7* mostra essa classe.

DIGITE

Listagem 7.7 A classe PrintSubClass2.

```
1: class PrintSubClass2 extends PrintClass {
2:     int z = 3;
3:
4:     void printMe( ) {
5:       System.out.println("x is " + x + ", y is " + y +
6:           ", z is " + z);
7:       System.out.println("I am an instance of the class " +
8:           this.getClass( ).getName( ));
9:     }
10:
11:    public static void main(String arguments[ ]) {
12:      PrintSubClass2 obj = new PrintSubClass2( );
13:      obj.printMe( );
14:    }
15: }
```

Agora, quando um objeto PrintSubClass for mstanciado e o método printMe( ) for chamado, aversão PrintSubClass de printMe( ) será chamada, em vez do que está na superclasse PrintClass. A saída seria a seguinte:

SAÍDA

```
x is 0, y is 1, z is 3
I am an instance of the class PrintSubClass2
```

Chamando o método oriainal

Normalmente, existem duas razões pelas quais você deseja anular um método que lima superclasse já tem implementado:

- Substituir a definição desse método original completamente.
- Aumentar o método original com comportamento adicional.

Você já aprendeu sobre o primeiro motivo — anular um método e dar-lhe uma nova definição ocultam a definição do método original. No entanto, existem ocasiões em que um comportamento deve ser incluído na definição original, em vez de se substituí-lo completamente, particularmente quando o comportamento é duplicado no método original e no método que o anula. Chamando o método original no corpo do método que irá anulá-lo, você pode incluir apenas o que precisar.

Use a palavra-chave super para chamar o método original dentro de uma definição de método. Essa palavra-chave passa a chamada de método para a hierarquia, como se vê a seguir:

```
void myMethod (String a, String b) {
   // faz algo aqui
   super.myMethod(a, b);
   // talvez faça mais alguma coisa aqui
}
```

7

A palavra-chave super, quase como a palavra-chave this, é um lugar reservado para a superclasse da classe. Você pode usá-la em qualquer parte em que utiliza thi s, mas super se refere à superclasse, em vez da classe atual.

Por exemplo, a listagem 7.8 mostra os dois diferentes métodos printMe( ) utilizados no exemplo anterior.

DIGITE

Listagem 7.8 Os métodos PrintMe( ).

```
1: // de PrintClass
2:     void printMe( ) {
3:       System.out.println("x is " + x + ", y is " + y);
4:       System.out.println("I am an instance of the class " +
5:         this.getClass(          ).getName( ));
6:     }
7:
8: //   de PrintSubClass2
9:    void printMe( ) {
10:      System.out.println("x is " + x + ", y is " + y +
          ", z is " + z);
11:      System.out.println("I am an instance of the class " +
12:           this.getClass( ).getName( ));
13:    }
```

Em vez de duplicar a maioria do comportamento do método da superclasse na subclasse, você pode reorganizar o método da superclasse para que o comportamento adicional seja facilmente incluído:

```
// de PrintClass
   void printMe( ) {
      System.out.println("I am an instance of the class " +
         this.getClass( ).getName( ));
      System.out.println("X is " + x);
      System.out.println("Y is " + y);
}
```

Então, quando você anular o método printMe( ) na subclasse, poderá chamar o método original e apenas inserir o que quiser a mais:

```
// de PrintSubClass
   void printMe( ) {
      super.printMe( );
      System.out.println("Z is " + z);
}
```

Resultaria o seguinte da chamada de printMe( ) em uma instância da subclasse:

SAÍDA

```
I am an instance of the class PrintSubClass2
X is 0
Y is 1
Z is 3
```

Anulando construtores

Tecnicamente, os métodos construtores não podem ser anulados. Como eles sempre possuem o mesmo nome da classe corrente, novos métodos construtores são criados, em vez de serem herdados. Esse sistema funciona a maior parte do tempo; quando o método construtor de sua classe é chamado, o método construtor com a mesma assinatura de todas as suas superclasses também é chamado. Portanto, a inicialização pode acontecer para todas as partes de uma classe que você herdar.

Entretanto, quando você estiver definindo métodos construtores para suas próprias classes, talvez queira mudar o modo como seu objeto é inicializado, não apenas inicializando novas variáveis incluídas por sua classe, mas também mudando o conteúdo de variáveis que já estão lá. Para fazer isso, chame explicitamente os métodos construtores da superclasse e, em seguida, mude todas as variáveis que precisarem ser alteradas.

Para chamar um método normal em uma superclasse, você usa super, *nomedométodo(argumentos)*. Como os métodos construtores não possuem um nome de método para chamar, a forma a seguir é utilizada:

```
super(arg1, arg2, ...);
```

Observe que a linguagem Java tem uma regra específica para o uso de super ( ): deve ser a primeira instrução em sua definição de construtor. Se você não chamar super ( ) explicitamente em seu construtor, a linguagem Java fará isso para você — usando super ( ) sem argumentos. Como uma chamada a um método super ( ) deve ser a primeira instrução, você não pode fazer algo como o seguinte no construtor que está anulando:

```
if (condition == true)
    super(1,2,3); // chama um construtor de superclasse
else
    super(1,2); // chama um construtor diferente
```

Semelhante ao uso de **this**(...) em um método construtor, super(...) chama o método construtor da superclasse imediata (que poderia, por sua vez, chamar o construtor de sua superclasse e assim por diante). Observe que um construtor com essa assinatura precisa existir na superclasse para que a chamada a super ( ) funcione. O compilador Java verifica isso quando você tenta compilar o arquivo-fonte.

Você não precisa chamar em sua superclasse o construtor que possui a mesma assinatura que o construtor de sua classe; você precisa apenas chamar o construtor para os valores que precisa inicializar. Na verdade, você pode criar uma classe que possua construtores com assinaturas totalmente diferentes a partir de qualquer um dos construtores da superclasse.

A listagem 7.9 mostra uma classe chamada NamedPoint, que estende a classe Point do pacote java.awt. A classe Point possui apenas um construtor, que recebe um argumento x e umy e retorna um objeto Poi nt. NamedPoi nt possui uma variável de instância adicional (um string do nome) e define um construtor para inicializar x, y e o nome.

DIGITE **Listagem 7.9** A classe NamedPoint.

```
1: import java.awt.Point;
2:
3: class NamedPoint extends Point {
4:       String name;
5:
6:       NamedPoint(int x, int y, String name) {
7:          super(x.y);
8:          this.name = name;
9:       }
10:
11:      public static void main (String[ ] arguments) {
12:         NamedPoint np = new NamedPoint(5, 5, "SmallPoint");
13:         System.out.println("x is " + np.x);
14:         System.out.println("y is " + np.y);
15:         System.out.println("Name is " + np.name);
16:      }
17: }.
```

A saída do programa é a seguinte:

SAÍDA

```
x is 5
y is 5
Name is SmallPoint
```

O método construtor definido aqui para NamedPoint chama o método construtor de Point para inicializar as variáveis de instância de Point (x e y). Embora você mesmo possa inicializar x e y facilmente, poderia não saber quais outros detalhes Point está realizando para inicializar. Portanto, sempre é uma boa idéia passar métodos construtores para a hierarquia, para certificar-se de que tudo está definido corretamente.

Métodos finalizadores

Métodos finalizadores são praticamente o oposto dos métodos construtores. Um *método construtor* é usado para inicializar um objeto e os *métodos finalizadores* são chamados imediatamente antes de o objeto ser reunido como lixo e ter sua memória reclamada.

O método finalizador é finalize( ). A classe Object define um método finalizador padrão que não faz nada. Para criar um método finalizador para suas próprias classes, anule o método finalize( ), usando a seguinte assinatura:

```
protected void finalizei ) throws Throwable {
      super.finalize( );
}
```

Nota

A parte throws Throwable dessa definição de método se refere aos erros que poderiam ocorrer quando esse método é chamado. Os erros em Java são chamados de exceções; você aprenderá mais sobre elas no Dia 16. Por enquanto, basta incluir essas palavras-chave na definição de método.

Inclua a limpeza que desejar para esse objeto, dentro do corpo desse método finalize( ). Você também pode chamar super.final i ze( ) para permitir que as superclasses de sua classe finalizem o objeto, se necessário.

Você mesmo pode chamar o método finalizei) a qualquer momento — trata-se de um método como qualquer outro. Entretanto, chamar f i nal i ze ( ) não dispara a coleta de um objeto para o lixo. Apenas a remoção de todas as referências a um objeto faz com que ele seja marcado para exclusão.

Os métodos finalizadores são melhor usados para a otimização da remoção de um objeto — por exemplo, removendo referências para outros objetos. Em muitos casos, você nem precisa usar finalizei).

Resumo

Hoje, você aprendeu todos os tipos de técnicas para usar, reutilizar, definir e redefinir métodos. Você aprendeu a respeito de métodos sobrecarregados, que reutilizam um nome de método dando-lhe diferentes argumentos, métodos construtores que definem as variáveis iniciais e outras condições iniciais de um objeto e sobre a anulação de herança de método. Você terminou com os métodos que finalizam um objeto: os finalizadores.

Após estudar por um dia a atuação do estilo de método da linguagem Java, você deve estar pronto para fazer o melhor em seus próprios programas.

Na próxima semana, você escreverá programas Java mais sofisticados, usando técnicas de Java 1.0.2 para applets e Java 2 para aplicativos. Você trabalhará com imagens gráficas, interfaces gráficas com o usuário, eventos de mouse e teclado e uso de janelas. É uma chance para os melhores momentos.

Boa sorte!

Perguntas e respostas

P **Eu criei dois métodos com as seguintes assinaturas:**

```
int total(int argl, int arg2, int arg3) {...}
fioat total(int argl, int arg2, int arg3) {...}
```

O compilador Java reclama quando tento compilar a classe com essas definições de método, mas suas assinaturas são diferentes. O que fiz de errado?

R A sobrecarga de método na linguagem Java funcionará somente se as listas de parâmetros forem diferentes — em número ou tipo de argumento. O tipo de retorno não é relevante para a sobrecarga de método. Pense sobre o seguinte: se você tivesse dois métodos exatamente com a mesma lista de parâmetros, como a linguagem Java saberia qual deles deveria chamar?

P **Posso sobrecarregar métodos anulados? (Isto é, posso criar métodos que possuam o mesmo nome que um método herdado, mas uma lista de parâmetros diferente?)**

R Claro! Desde que as listas de parâmetros variem, não importa se você definiu um novo nome de método ou um que tenha herdado de uma superclasse.

SEMANA 2

Partindo para recursos avançados da linguagem Java

8 Colocando programas interativos na Web

9 Melhorando a aparência dos programas com imagens gráficas, fontes e cores

10 Inserindo imagens, animação e som

11 Criando interfaces com o usuário simples para applets

1 2 Organizando componentes em uma interface com o usuário

1 3 Respondendo à entrada de usuário em um applet

14 Desenvolvendo interfaces com o usuário avançadas com o AWT

SEMANA 2

Dia 8

Colocando programas interativos na Web

A linguagem Java tem sido a estrela do rock das linguagens de computador desde seu aparecimento, recebendo uma publicidade normalmente reservada aos escândalos presidenciais, substitutos do açúcar e atletas profissionais que passam a ser criminosos.

O maior motivo para toda essa propaganda tem sido os applets — programas Java que são executados na World Wide Web. O primeiro contato das pessoas com a linguagem Java foi quando o Netscape Navigator começou a executar applets, no final de 1995.

Embora a linguagem Java possa hoje ser utilizada para muitas coisas, além da World Wide Web, um número considerável de programadores ainda está aprendendo esta linguagem para escrever applets.

Na última semana, você se concentrou no aprendizado da linguagem Java propriamente dita e quase todos os programas criados eram aplicativos Java. Nesta semana, você passará para o desenvolvimento de applets.

Hoje, você começará com os fundamentos:

- As diferenças entre applets e aplicativos
- Como criar um applet simples
- Como colocar um applet em uma página da Web

- Como enviar informações de uma página da Web para um applet
- Como armazenar um applet em um repositório de arquivos para obter um download mais rápido de uma página da Web

Qual é a diferença entre applets e aplicativos

A diferença entre applets e aplicativos Java reside em como eles são executados.

Os aplicativos são executados utilizando-se um interpretador Java para carregar o arquivo de classe principal do aplicativo. Normalmente, isso é feito a partir de um prompt de linha de comando, usando-se a ferramenta java do JDK, conforme você fez desde o Dia 1 deste livro.

Por outro lado, os applets são executados em qualquer navegador que ofereça suporte à linguagem Java. No momento, isso inclui as versões atuais do Netscape Navigator, Microsoft Internet Explorer e o navegador Hotjava da Sun. Eles também podem ser testados usando-se a ferramenta appletviewer, incluída no Java Development Kit.

Para que um applet seja executado, ele deve ser incluído em uma página da Web usando-se tags HTML, da mesma maneira que imagens e outros elementos são incluídos. Quando um usuário com um navegador compatível com Java carregar uma página da Web que inclua um applet, esse navegador fará o download do applet a partir de um servidor da Web e irá executá-lo no sistema do próprio usuário da Web. Não é necessário um interpretador Java separado — já existe um incorporado ao navegador. Assim como no caso de um aplicativo, um applet Java inclui um arquivo de classe e todas as outras classes auxiliares necessárias para se executá-lo. A biblioteca de classe Java padrão é incluída automaticamente.

Como os applets Java são executados dentro de um navegador Java, uma parte do trabalho de criação de uma interface com o usuário já está realizada para o programador de applets. Há uma janela na qual o applet pode ser executado, um lugar para apresentar imagens gráficas e receber informações e a interface do navegador.

E possível para um único programa Java funcionar como applet e como aplicativo. Embora procedimentos diferentes sejam usados para criar estes tipos de programas, eles não entram em conflito. Os recursos específicos dos applets seriam ignorados se o programa fosse executado como um aplicativo e vice-versa.

Restrições de segurança de applet

Como os applets Java são executados no sistema de um usuário da Web, existem sérias restrições quanto ao que um applet é capaz de fazer. Caso essas restrições não existissem, um programador de Java mal-intencionado poderia facilmente

escrever um applet que excluísse arquivos de usuário, reunisse informações privativas do sistema e encontrasse outras brechas na segurança.

Como regra geral, os applets Java são executados sob um modelo de segurança que segue a idéia de que "é melhor prevenir do que remediar". Um applet não pode fazer nada do que está listado a seguir:

- Eles não podem ler ou escrever arquivos no sistema de arquivos do usuário.
- Eles não podem se comunicar com um site da Internet que não seja aquele que serviu a página da Web que o inclui.
- Eles não podem executar nenhum programa no sistema do leitor.
- Eles não podem carregar programas armazenados no sistema do usuário, como programas executáveis e bibliotecas compartilhadas.

Todas essas regras são válidas para os applets Java em execução no Netscape Navigator ou no Microsoft Internet Explorer, os navegadores escolhidos pela maioria dos usuários da Web atualmente. Outros navegadores compatíveis com Java e ferramentas de desenvolvimento Java podem deixar que você configure o nível de segurança desejado, permitindo algum acesso a arquivo de pastas específicas ou conexões de rede com sites da Internet selecionados.

Como exemplo, a ferramenta appletviewer permite que uma lista de controle de acesso seja definida para as pastas em que um applet pode ler ou escrever arquivos. Entretanto, um desenvolvedor de applets pode supor que a maior parte do público vá usar um navegador que implemente as regras de segurança mais restritas.

Os aplicativos Java não têm nenhuma das restrições impostas aos applets. Eles podem tirar total proveito dos recursos da linguagem Java.

Embora o modelo de segurança da linguagem Java tome extremamente difícil para que um applet prejudicial danifique o sistema de um usuário, ele nunca será 100% seguro. Pesquise o termo "hostile applets" na Web e você verá uma discussão sobre os problemas de segurança nas diferentes versões da linguagem Java e como eles foram resolvidos. Você pode até encontrar exemplos de applets que causam problemas para as pessoas que estão usando navegadores Java. A linguagem Java é mais segura do que as outras soluções de programação da Web, como ActiveX, mas todos os usuários de navegador devem estar totalmente a par do problema.

Escolhendo uma versão de Java

Um programador Java que escreve applets deve responder à seguinte pergunta: para qual versão da linguagem Java devo escrever?

No momento da produção deste livro, a linguagem Java 1.02 era a única versão suportada integralmente nos navegadores Netscape Navigator e Microsoft Internet Explorer, o que compreende mais de 90% do mercado de applets. A Netscape demorou para oferecer suporte à versão 1.1 no seu navegador 4.0 e a Microsoft pode até nunca vir a oferecer suporte a ela.

A Sun desenvolveu um complemento de navegador, chamado Java Plug-in, que permite aos programadores de applet usar aprimoramentos de Java 1.1 e 2 em seus programas. O Java Plug-in está incluído no JDK 1.2, assim como no JRE 1.2.

Devido a essa divisão, a ação mais amplamente adotada entre os programadores parece ser a seguinte:

- Os applets são escritos usando-se apenas recursos de Java 1.02, pois serão executados em todos os navegadores compatíveis com Java.
- Os aplicativos são escritos usando-se Java 2, pois eles podem ser executados em qualquer sistema que possua um interpretador Java 2.

A linguagem Java 2 foi projetada de forma que um programa usando apenas recursos de Java 1.02 possa ser compilado e executado sem problemas em um interpretador Java 1.02 ou em um navegador compatível com 1.02.

Se um applet utilizar qualquer recurso que tenha sido introduzido com Java 1.1 ou 2, o programa não será executado em um navegador que não ofereça suporte a essas versões da linguagem. O único ambiente de teste que oferece suporte integral a essas versões é o último appletviewer da JavaSoft.

Essa é uma fonte de erros comum para os programadores de applets Java. Se você escrever um applet Java 2 e executá-lo em um navegador que não ofereça esse suporte, como o Microsoft Internet Explorer 4.0, obterá erros de segurança, erros de classe não encontrada e outros problemas que o impedirão de ser executado.

Neste livro, a maior parte da programação de applets é ensinada com o uso das técnicas pertinentes à linguagem Java 1.02, pois esse ainda é o padrão para programação da Web. Normalmente, os applets são programas menores, que não exigem muitos aprimoramentos introduzidos na linguagem Java 1.1 ou 2.

A programação de aplicativos usará os recursos mais recentes e melhores da linguagem Java 2. Quando os navegadores estiverem acompanhando os projetistas da linguagem, você poderá usar as bibliotecas de classe Java 2 em seus applets.

As diferenças de versão serão observadas por todo este livro. Ocasionalmente, o compilador Java notará essas diferenças quando você usar um recurso de Java 1.02 que foi substituído por uma solução melhor da Java 2, e você será avisado disso também.

Controle de segurança aprimorado

O modelo de segurança descrito até este ponto é aquele introduzido com a linguagem Java 1.02. Aversão atual da linguagem Java inclui um modo para um usuário da Web confiar em um applet, para que esse applet possa ser executado sem restrições no sistema do usuário, exatamente como acontece com um aplicativo.

A linguagem Java 2 permite que controles de segurança muito específicos sejam colocados ou removidos de applets e aplicativos. Isso será abordado no Dia 16.

Criando applets

A maior parte dos programas Java que você criou até agora foi formada de aplicativos Java — programas simples com um único método mai n ( ), utilizado para criar objetos, definir variáveis de instância e chamar outros métodos.

Os applets não possuem um método main( ) que é chamado automaticamente para iniciar o programa. Em vez disso, existem vários métodos que são chamados em diferentes pontos na execução de um applet. Você aprenderá mais sobre esses métodos hoje.

Todos os applets são subclasses da classe Applet no pacote java.applet. A classe Applet fornece dois tipos de comportamento que todos os applets devem ter:

- Funcionar como parte de um navegador e tratar de ocorrências como o recarregamento da página do navegador.
- Apresentar uma interface gráfica com o usuário e pegar entradas de usuários.

Embora um applet possa fazer uso de quantas classes forem necessárias, a classe Applet é a principal, que ativa a execução do applet. A subclasse de Appl et que você cria assume a seguinte forma:

```
public class yourApplet extends java.applet.Applet {
   // código do applet aqui

}
```

Todos os applets devem ser declarados publ i c, pois a classe Appl et é uma classe pública. Esse requisito é válido apenas para sua classe Appl et principal e qualquer uma das classes auxiliares podem ser públicas ou privativas. Mais informações sobre esse tipo de controle de acesso estão descritas no Dia 15.

Quando o interpretador Java interno de um navegador encontra um applet Java em uma página da Web, a classe desse applet é carregada, junto com todas as outras classes auxiliares que ele utiliza. O navegador cria automaticamente uma instância da classe do applet e chama métodos da classe Applet quando ocorrem eventos específicos.

Diferentes applets que utilizam a mesma classe usam diferentes instâncias; portanto, você poderia colocar mais de uma cópia do mesmo tipo de applet em uma página e cada uma delas poderia ter um comportamento distinto.

Principais atividades do applet

Em vez de um método main( ), os applets possuem métodos que são chamados quando ocorrem coisas específicas na execução do applet.

Um exemplo desses métodos é o paint( ), que é chamado quando a janela do applet precisa ser apresentada ou reapresentada.

Por definição, esses métodos não fazem nada. Por exemplo, o método paint( ) que é herdado de Applet é um método vazio. Para que algo seja apresentado na janela do applet, o método paint( ) deve ser anulado com o comportamento para apresentar texto, imagens gráficas e outros itens.

Você aprenderá aqui sobre os métodos da classe Applet que devem ser anulados, à medida que a semana avançar. As seções a seguir descrevem cinco dos métodos mais importantes na execução de um applet: inicialização, partida, interrupção, destruição e pintura.

Inicialização

A inicialização ocorre quando o applet é carregado. A *inicialização* poderia incluir a criação dos objetos de que o applet precisa, configuração do estado inicial, carregamento de imagens ou fontes ou o ajuste de parâmetros. Para fornecer comportamento para a inicialização de um applet, você anula o método init( ), como segue:

```
public void init( ) {
    // Código aqui
}
```

Partida

A partida de um applet é realizada após ele ser inicializado. *A partida* também pode ocorrer se o applet foi interrompido anteriormente. Por exemplo, um applet é interrompido se o usuário do navegador seguir um vínculo para uma página diferente, e é iniciado novamente quando o usuário retorna à página que contém o applet.

A partida pode ocorrer várias vezes durante o ciclo de vida de um applet, mas a inicialização só ocorre uma vez. Para fornecer comportamento de início para seu applet, anule o método start( ), como segue:

```
public void start( ) {
    // Código aqui
}
```

A funcionalidade que você coloca no método start( ) pode incluir o envio das mensagens apropriadas para objetos auxiliares ou informar ao applet

de alguma forma para que comece a ser executado. Você aprenderá mais sobre o início de applets no Dia 10.

Interrupção

A interrupção e a partida andam lado a lado. A *interrupção* ocorre quando o usuário sai da página que contém um applet que esteja em execução ou quando um applet pára sozinho, chamando stop( ) diretamente. Por definição, todos os threads que o applet tenha iniciado continuam a ser executados mesmo depois que o usuário sair de uma página. Anulando stop ( ), você pode suspender a execução desses threads e reiniciá-los, caso o applet seja visto novamente. A seguir está a forma de um método stop( ):

```
public void stop( ) {
    // Código aqui
)
```

Destruição

A destruição soa como algo mais escabroso do que realmente é. O método destroy ( ) permite que o applet faça uma limpeza pouco antes de ser liberado da memória ou que se saia do navegador. Você pode usar esse método para acabar com todos os threads que estiverem em execução ou para liberar todos os outros objetos que estiverem sendo executados. Geralmente, você não vai querer anular destroy( ), a não ser que existam recursos específicos que precisam ser liberados, como no caso de threads que o applet criou. Para fornecer comportamento de limpeza ao seu applet, anule o método destroy( ) como segue:

```
public void destroy( ) {
    // Código aqui
}
```

Talvez você esteja se perguntando como `destroy( )` difere do método `finalize( )`, que foi descrito no Dia 7. O método `destroy( )` se aplica apenas aos applets; `finalize( )` é um modo de propósito mais geral para que um objeto simples de qualquertipo faça a limpeza após ele mesmo.

A linguagem Java possui um Coletor de lixo automático que gerencia a memória para você. O coletor recupera a memória dos recursos depois que o programa terminou de usá-los; portanto, normalmente você não precisa usar métodos como o destroy( ).

Pintura

Através *dapintura é* que um applet apresenta algo na tela, seja texto, uma linha, um fundo colorido ou uma imagem. A pintura pode ocorrer muitas centenas de vezes, durante o ciclo de vida de um applet: uma vez que o applet é inicializado, novamente se a janela do navegador ficar atrás de outra janela na tela, outra vez se a janela do navegador for movida para uma posição diferente na tela etc. Para apresentar algo, você deve anular o método paint( ) de sua subclasse Applet. O método paint( ) é como o seguinte:

```
public void paint(Graphics g) {
    // Código aqui
}
```

Observe que, ao contrário de outros métodos descritos nesta seção, pai nt ( ) recebe um argumento: uma instância da classe Graphi cs. Esse objeto é criado e passado para pai nt ( ) pelo navegador; portanto, você não precisa se preocupar com isso. Entretanto, você sempre deve importar a classe Graphics (parte do pacote java.awt) para seu código de applet, normalmente através de uma instrução import no início de seu arquivo-fonte Java, como no seguinte:

```
import java.awt.Graphics;
```

Se você estiver importando várias classes do mesmo pacote, como as classes Abstract Windowing Toolkit, pode usar um caractere de curinga para carregar todas elas ao mesmo tempo. Por exemplo, a instrução `import java.awt.*;` torna disponível toda classe pública do pacote java.awt. Contudo, a instrução import não inclui as subclasses do pacote, de modo que a instrução `import java.awt.*;` não inclui as classes do pacote java.awt.image.

Um applet simples

No Dia 2, você criou um applet simples chamado Palindrome, que apresentava o texto "Go hang a salami, I'm a lasagna hog.". Você gerou e utilizou esse applet como um exemplo da criação de uma subclasse.

Esse applet é revisto aqui, por um motivo diferente: para examiná-lo como um exemplo de programação de applet. A listagem 8.1 mostra o código desse applet.

Listagem 8.1 O texto completo de Palindrome. java.

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class Palindrome extends java.applet.Applet {
```

Listagem 8.1 Continuação.

```
6:    Font f = new Font("TimesRoman", Font.BOLD, 36);
7:
8:    public void paint(Graphics screen) {
9:        screen.setFont(f);
10:       screen.setColor(Color.red);
11:       screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 40);
12:   }
13: }
```

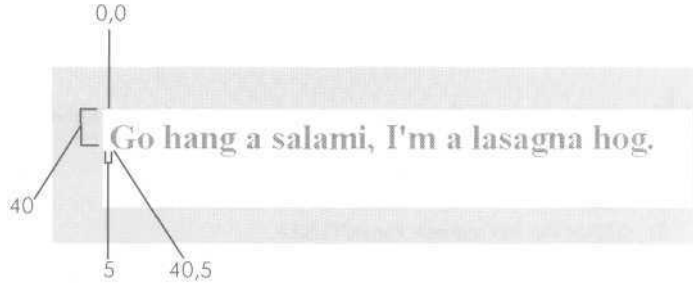
Esse applet anula o método paint( ). Como o applet apresenta apenas algumas palavras na tela, não há nada para ser inicializado. Daí que os métodos start( ),stop( ) e init( ) não são necessários.

E no método paint( ) que o trabalho real desse applet ocorre. O objeto Graphi cs passado para o método pai nt ( ) contém o estado da imagem gráfica, que controla os atributos atuais da superfície de desenho. O estado inclui detalhes sobre a fonte e a cor atuais a serem usadas para qualquer operação de desenho, por exemplo.

As linhas 9 e 10 definem a fonte e a cor desse estado da imagem gráfica. O objeto Font é mantido na variável de instância f e um objeto representando a cor vermelha (red) é armazenado na variável de classe red de Color.

A linha 11 usa a fonte e a cor atuais para desenhar o string "Go hang a salami, I'm a lasagna hog." na posição 5, 40. Observe que o ponto 0 de x,y é o canto superior esquerdo da superfície de desenho do applet, com os valores positivos de y crescendo de forma descendente; portanto, 40 está na parte inferior do applet. A Figura 8.1 mostra como a caixa de contorno do applet e o string são desenhados na página.

Fia ura 8.1
Desenhando o *applet.*



Se você implementar os métodos de applet corretos em sua classe (i ni t ( ), start( ), stop( ), paint( ) etc), seus applets funcionarão de modo transparente, sem a necessidade de um ponto de salto explícito.

Incluindo um applet em uma página da Web

Após criar a classe (ou classes) que compõe seu applet e compilá-la em arquivos de classe, você deve criar uma página da Web para nela colocar o applet.

Os applets são colocados em uma página usando-se a tag <APPLET>, um comando de programação HTML que funciona como outros elementos da HTML. Também existem várias ferramentas de desenvolvimento de página da Web, como o Microsoft FrontPage e o Macromedia Dreamweaver, que podem ser usados para se incluir applets em uma página sem usar HTML.

O objetivo da tag <APPLET> é colocar um applet em uma página da Web e controlar sua aparência em relação às outras partes da página.

Os navegadores compatíveis com Java utilizam as informações contidas na tag para localizar e executar os arquivos de classe compilados do applet. Nesta seção, você aprenderá a colocar applets Java em uma página da Web e a servir os arquivos executáveis Java na Web de modo geral.

A seção a seguir presume que você tenha pelo menos um ligeiro entendimento **da** escrita de páginas HTML ou que saiba usar uma ferramenta de desenvolvimento da Web para aproximar-se da HTML. Se você precisa de ajuda nessa área, existem diversas opções e os co-autores deste livro possuem várias ofertas, incluindo o Sams Teach *Yourself Web Publishing witn HTML 4 in 21 Days,* de Laura Lemay, e o *Aprenda* em *24 horas Microsoft FrontPage 2000,* de Rogers Cadenhead.

A taa <APPLET>

A tag <APPLET> é uma extensão especial da HTML para incluir applets Java em páginas da Web; a tag é suportada por todos os navegadores que manipulam programas Java. A listagem 8.2 mostra um exemplo simples de uma página da Web com um applet incluído.

Listagem 8.2 O texto completo de PalindromePage.html.

```
1: <HTML>
2: <HEAD>
3: <TITLE>The Palindrome Page</TITLE>
4: </HEAD>
5: <B0DY>
6: <P>Hy favorite meat-related palindrome is:
7: <BR>
8: <APPLET CODE="Palindrome.class" WIDTH=600 HEIGHT=100>
9: A secret if your browser does not support Java!
10: </APPLET>
11: </B0DY>
12: </HTML>
```

Nesse exemplo, a tag <APPLET> inclui três atributos:

- CODE Especifica o nome do arquivo de classe principal do applet.
- WIDTH Especifica a largura da janela do applet na página da Web.
- HEIGHT Especifica a altura da janela do applet.

O arquivo de classe indicado pelo atributo CODE deve estar na mesma pasta que a página da Web que contém o applet, a menos que você utilize um atributo CODEBASE para especificar uma pasta diferente. Você aprenderá como fazer isso posteriormente, ainda hoje.

WIDTH e HEIGHT são atributos obrigatórios, pois o navegador da Web precisa saber qual é o espaço que deve ser dedicado ao applet na página. É fácil desenhar em uma área fora da janela do applet em um programa; portanto, você deve se certificar de estar proporcionando uma janela suficientemente grande.

Texto, imagens e outros elementos de página da Web podem ser incluídos entre as tags <APPLET> e </APPLET>. Elas são as únicas apresentadas nos navegadores que não conseguem manipular programas Java, e incluí-las é uma boa maneira de permitir que as pessoas saibam que estão deixando de ver um applet Java, porque o navegador não oferece suporte a elas. Se você não especificar nada entre <APPLET> e </APPLET>, os navegadores que não oferecem suporte à linguagem Java não apresentam nada no lugar do applet.

No exemplo atual, o texto que aparece acima do applet indica My f avori te meat-related palindrome is:. Os usuários que possuem navegadores Java vêem o applet Palindrome abaixo desse texto. Os usuários que não possuem Java vêem o texto alternativo que foi fornecido — A secret if your browser does not support Java!.

Testando o resultado

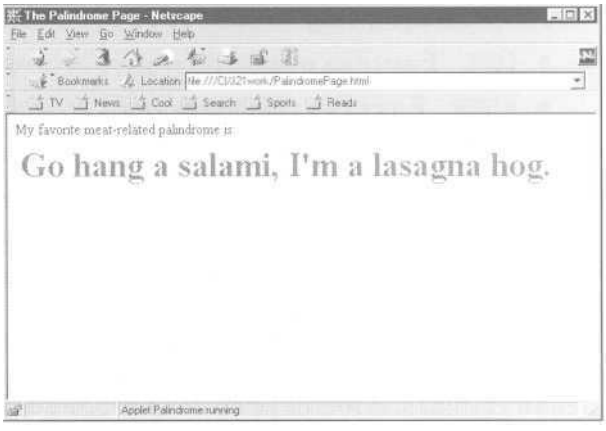
Uma vez que você tenha um arquivo de classe de applet principal e um arquivo HTML que utilize o applet, pode carregar o arquivo HTML em um navegador compatível com Java a partir de seu disco local. Usando-se o Netscape Navigator, os arquivos locais podem ser carregados com o comando Fi 1 e ! Open Page ! Choose File no Internet Explorer, escolha File ! Open ! Browse, para localizar o arquivo correto no seu sistema. O navegador carregará sua página da Web e o applet nela contido.

Se você não tem um navegador compatível com Java, deve haver um modo de carregar applets incluídos em seu ambiente de desenvolvimento. O JDK inclui a ferramenta appletviewer para testar seus applets. Ao contrário de um navegador, a ferramenta appletviewer apresenta apenas os applets que estão incluídos em uma página da Web. Ela não apresenta a página da Web em si.

A Figura 8.2 mostra a página Pai indromePage.html carregada no Netscape Navigator.

Figura 8.2

A página da Web PalindromePage.html no Navigator.



Colocando applets na Web

Depois que você tiver um applet que funcione perfeitamente ao ser testado localmente em seu próprio sistema, você pode torná-lo disponível na World Wide Web.

Os applets Java são apresentados por um servidor da Web da mesma maneira que arquivos HTML, imagens e outra mídia. Você armazena o applet em uma pasta acessível ao servidor da Web — freqüentemente, a mesma pasta que contém a página da Web que contém o applet. O servidor da Web deve estar configurado para oferecer applets Java para os navegadores que oferecem suporte à linguagem.

Existem certos arquivos que você precisa fazer upload em um servidor da Web:

- A página HTML que contém o applet.
- Todos os arquivos .class utilizados pelo applet, que não façam parte da biblioteca de classe padrão da linguagem Java.

Se você souber como publicar páginas da Web, arquivos de imagem e outros arquivos de multimídia, não terá de aprender nada novo para publicar applets Java em seu site.

Mais informações sobre a taa <APPLET>

Em sua forma mais simples, a tag <APPLET> usa os atributos CODE, WIDTH e HEIGHT para criar um espaço do tamanho apropriado e, em seguida, carregar e executar o applet nesse espaço. Entretanto, a tag <APPLET> inclui vários outros atributos que podem ajudá-lo a integrar melhor um applet no design global de uma página da Web.

Nota

Os atributos disponíveis para a tag <APPLET> são quase idênticos aos da tag HTML <IMG>.

ALIGN

O atributo ALIGN define como o applet ficará alinhado em uma página da Web com relação às outras partes da página. Esse atributo pode ter nove valores diferentes:

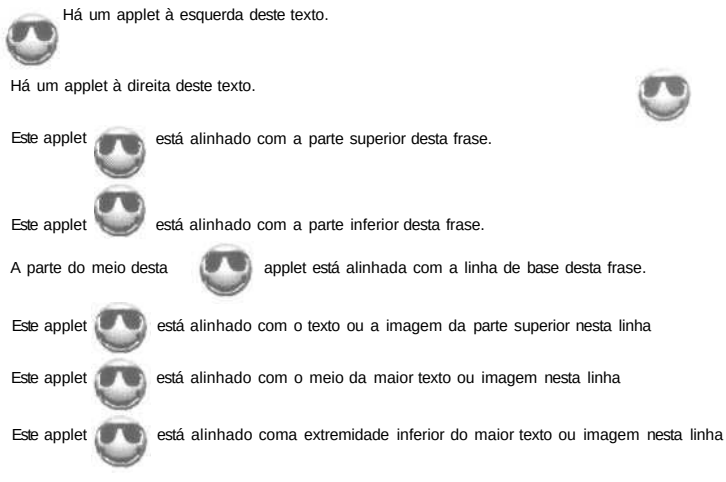
- ALIGN=LEFT Alinha o applet à esquerda do texto que o acompanha na página.
- ALIGN=RIGHT Alinha o applet à direita do texto que o acompanha na página.
- ALIGN=TEXTTOP Alinha a parte superior do applet com a parte superior do texto mais alto na linha.
- ALIGN=TOP Alinha o applet com o item mais alto na linha (que pode ser outro applet, uma imagem ou a parte superior do texto).
- ALIGN=ABSMIDDLE Alinha o meio do applet com o meio do maior item na linha.
- ALIGN=MIDDLE Alinha o meio do applet com o meio da linha de base do texto.
- ALIGN=BASELINE Alinha a parte inferior do applet com a linha de base do texto. ALIGN=BASELINE é o mesmo que ALIGN=BOTTOM, mas esse é um nome mais descritivo.
- ALIGN=ABSBOTTOM Alinha a parte inferior do applet com o menor item na linha (que pode ser a linha de base do texto ou outro applet ou imagem).

Para finalizar a formatação especificada com o atributo ALIGN, você pode usar a tag de quebra de linha HTML (<BR>) com o atributo CLEAR. Isso requer três valores:

- <BR CLEAR=LEFT> Continua a exibir o restante da página da Web na próxima margem esquerda em branco.
- <BR CLEAR=RIGHT> Continua a exibir na próxima margem direita em branco.
- <BR CLEAR=ALL> Continua a exibir na próxima margem esquerda e direita em branco.

A Figura 8.3 mostra as diversas opções de alinhamento, em que o rosto sorrindo é um applet.

Figura 8.2
Opções de *alinhamento de applet.*



Se estiver usando uma ferramenta de desenvolvimento da Web que permita colocar applets Java em uma página, você deve ser capaz de definir o atributo ALIGN escolhendo LEFT, RIGHT ou um dos outros valores dentro do programa.

HSPACE e VSPACE

Os atributos HSPACE e VSPACE são usados para definir o espaço, em pixels, entre um applet e o texto que o envolve. HSPACE controla o espaço horizontal à esquerda e à direita do applet e VSPACE controla seu espaço vertical acima e abaixo. Como exemplo, há aquele fragmento de HTML com espaço vertical de 50 e horizontal de 10:

```
<APPLET CODE="ShowSmiley.class" WIDTH=45 HEIGHT=42
ALIGN=LEFT VSPACE=50 HSPACE-10>
Requires Java
</APPLET>
```

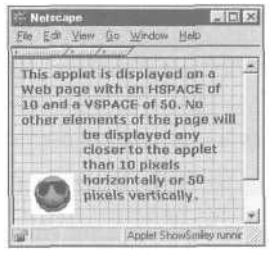
A Figura 8.4 mostra como esse applet, que apresenta um rosto sorrindo sobre um fundo branco, seria apresentado com outros elementos de uma página da Web. O fundo da página é uma grade e cada grade tem 10x10 pixels de tamanho. Você pode usar a grade para medir o espaço entre o applet e o texto na página.

CODE e CODEBASE

Os atributos CODE e CODEBASE, ao contrário de outras partes da tag <APPLET>, são usados para indicar onde o arquivo de classe principal do applet e outros arquivos podem ser encontrados. Eles são usados por um navegador compatível com Java quando ele tenta executar um applet, após carregá-lo por download a partir de um servidor da Web.

Figura 8.4

Espaço *vertical* e *horizontal.*



CODE indica o nome do arquivo de classe principal do applet. Se CODE for usado sem um correspondente atributo CODEBASE, o arquivo de classe será carregado do mesmo lugar que a página da Web que contém o applet.

Você deve especificar a extensão de arquivo .class com o atributo CODE. A seguir está um exemplo de uma tag <APPLET> que carrega um applet chamado Bix.class a partir da mesma pasta que a página da Web:

```
<APPLET CODE="Bix.class" HEIGHT=40 WIDTH=400>
</APPLET>
```

O atributo CODEBASE é usado para fazer com que o navegador procure o applet e todos os outros arquivos que ele utiliza em uma pasta diferente. CODEBASE indica uma pasta alternativa ou até um site alternativo da World Wide Web para carregar a classe e os outros arquivos. O código a seguir carrega uma classe chamada Bi x. cl ass de uma pasta chamada Torshi re:

```
<APPI_ET C0DE="Bix.class" CODEBASE="Torshire" HEIGHT=40 WIDTH=400>
</APPLET>
```

A seguir há um exemplo no qual os arquivos de classe Java são carregados a partir de um site da Web totalmente diferente daquele que contém a página:

```
<APPLET C0DE="Bix.class" C0DEBASE="http://www.torshire.com/javacl asses"
HEIGHT=40 WIDTH=400>
</APPLET>
```


A tag<OBJECT>

A tag <APPLET> é uma extensão da HTML introduzida especificamente para apresentar programas Java em páginas da Web. Atualmente, existem outros tipos de programas que podem ser executados interativamente em uma página, incluindo controles ActiveX, applets NetRexx e programas Python. Para lidar com todos esses tipos de programas sem exigir uma tag diferente para cada um deles, a tag <OBJECT> foi incluída na especificação HTML.

A tag <OBJECT> é usada para todos os objetos — programas interativos e outros elementos externos — que possa ser apresentados como parte de uma página da Web. Ela é suportada pelas versões 4.0 e superior do Netscape Navigator e do Microsoft Internet Explorer. Os navegadores mais antigos não oferecem suporte a essa nova tag; portanto, em muitos casos, você ainda poderá usar a tag <APPLET>.

A tag <OBJECT> assume a seguinte forma:

```
OBJECT CLASSID="java:Bix.class" CODEBASE="javaclasses" HEIGHT=40 WIDTH=400">
</0BJECT>
```

Alternar <APPLET> para <0BJECT> exige as seguintes alterações:

- A tag <0BJECT> deve ser usada no lugar de <APPLET>.
- O atributo CODE deve ser substituído por CLASSID. Além disso, o texto "java:" deve ser colocado antes do nome do arquivo de classe do applet. Por exemplo, se o applet estiver em GameApplet.class, o atributo CLASSID deverá ser java: GameApplet.class.

Caso contrário, os atributos permanecem os mesmos, incluindo CODE-BASE, HEIGHTTN], WIDTH, ALIGN. A tag <OBJECT> também pode usar tags <PARAM> opcionais, o que está descrito posteriormente, ainda hoje.

A listagem 8.3 contém uma página da Web que carrega o applet Palindrome usando a tag <OBJECT>. Todo o restante é igual ao exemplo anterior, na listagem 8.2.

DIGITE

Listagem 8.3. O texto completo de palindromepage2.html.

```
1: <HTML>
2: <HEAD>
3: <TITLE>The Palindrome Page</TITLE>
4: </HEAD>
5: <B0DY>
6: <P>My favorite meat-related palindrome is:
7: <BR>
8: <OBJECT CLASSID="java:Paiindrome.class" WIDTH=600 HEIGHT=100>
9: A secret if your browser does not support Java!
10: </0BJECT>
11: </B0DY>
12: </HTML>
```

Repositórios de arauivos Java

O modo padronizado de se colocar um applet Java em uma página da Web é usar <APPLET> ou <0BJECT> para indicar o arquivo de classe principal do applet. Então, um navegador compatível com Java carrega por download e executa o applet. Todas as outras classes e arquivos necessários pelo applet são carregados por download a partir do servidor da Web.

O problema na execução de applets desse modo é que cada arquivo de que um applet precisa — seja ele outra classe auxiliar, imagem, arquivo de áudio, arquivo de texto ou qualquer outra coisa — requer uma conexão separada de um navegador da Web com o servidor que contém o arquivo. Como essa enorme quantidade de tempo é necessária apenas para se fazer a própria

conexão, isso pode aumentar o tempo necessário para se carregar por download um applet e tudo o que ele precisa para ser executado.

A solução desse problema é um repositório de arquivos Java ou arquivo JAR. Um *repositório de arquivos Java* é um conjunto de classes Java e outros arquivos empacotados em um único arquivo. Usando um repositório de arquivos Java, o navegador faz uma única conexão com o servidor, em vez de várias. Reduzindo o número de arquivos que o navegador precisa carregar do servidor, você pode carregar por download e executar seu applet mais rapidamente. Os repositórios de arquivoss Java também podem ser compactados, tornando o tamanho do arquivo global menor e, portanto, mais rápido de ser carregado por download — embora demore algum tempo, no lado do navegador, para que os arquivos sejam descompactados antes de poderem ser executados.

As versões 4.0 e superiores dos navegadores Navigator e Internet Explorer incluem suporte a arquivos JAR. Para criar esses repositórios de arquivos, o JDK inclui uma ferramenta chamada `jar`, que pode empacotar e desempacotar arquivos em repositórios de arquivos Java. Os arquivos JAR podem ser compactados usando-se o formato Zip ou podem ser empacotados sem a utilização de compactação. O comando a seguir empacota todos os arquivos de classe e de imagem GIF de uma pasta em um único repositório de arquivos Java chamado `Animate.jar`:

```
jar cf Animate.jar *.class *.gif
```

O argumento `cf` especifica duas opções de linha de comando que podem ser usadas ao se executar o programa `jar`. A opção `c` indica que um repositório de arquivos Java deve ser criado e `f` indica que o nome do repositório de arquivos acompanhará um dos argumentos seguintes.

Você também pode incluir arquivos específicos em um repositório de arquivos Java, com um comando como o seguinte:

```
jar cf Smiley.jar ShowSmiley.class ShowSmiley.html spinhead.gif
```

Isso cria um repositório de arquivos `Smiley.jar` contendo três arquivos: `ShowSmiley.class`, `ShowSmiley.html` e `spinhead.gif`.

Execute o `jar` sem nenhum argumento para ver uma lista de opções que podem ser utilizadas com o programa.

Depois que você cria um repositório de arquivos Java, o atributo `ARCHIVE` é usado com a tag `<APPLET>` para mostrar onde o repositório de arquivos pode ser encontrado. Você pode usar os repositórios de arquivos Java com um applet, com tags como segue:

```
<applet code="ShowSmiley.class" archive="Smiley.jar" width=45 height=42>
</applet>
```

Essa tag especifica que um repositório de arquivos chamado `Smiley.jar` contém arquivos usados pelo applet. Os navegadores e as ferramentas de

navegação que oferecem suporte a arquivos JAR procurarão, dentro do repositório de arquivos, os arquivos necessários quando o applet entra em execução.

Embora um repositório de arquivos Java possa conter arquivos de classe, o atributo ARCHIVE não elimina a necessidade do atributo CODE. Um navegador ainda precisa saber o nome do arquivo de classe principal do applet para carregá-lo.

Outros formatos de repositório de arauivos

Antes que os desenvolvedores da linguagem Java introduzissem o formato de arquivo JAR, a Netscape e a Microsoft ofereceram suas próprias soluções de repositório de arquivos. Elas não oferecem algumas das vantagens dos repositórios de arquivos Java, mas têm a vantagem de funcionar com navegadores compatíveis com Java que ainda não oferecem suporte às versões da linguagem acima de 1.02.

As versões atuais dos navegadores da Web da Netscape oferecem suporte ao uso de repositórios de arquivos Zip com o atributo ARCHIVE, mas podem ser usadas somente para arquivos de classe e não para imagens ou outros tipos de arquivos de que um applet poderia necessitar. Dentro do Netscape, você pode usar o atributo ARCHIVE para indicar o nome do repositório de arquivos, como segue:

```
<APPLET CODE="MyApplet.class" ARCHIVE="appletstuff.zip" WIDTH=100 HEIGHT=100>
</APPLET>
```

O repositório de arquivos em si é um arquivo Zip descompactado. Os arquivos Zip padrão, que utilizam alguma forma de compactação para tornar o arquivo menor, não são reconhecidos. Além disso, classes auxiliares podem estar contidas dentro ou fora do arquivo Zip; os navegadores Netscape olharão nos dois lugares. O atributo ARCHIVE é ignorado pelos navegadores ou visualizadores de applet que possam passar por essa página da Web.

O Microsoft Internet Explorer reconhece um terceiro tipo de formato de repositório de arquivos, para a distribuição de applets Java: o arquivo CAB. CAB é a abreviatura de "cabinet", cuja tradução literal é "gabinete" e trata-se de um modo de agrupar arquivos e compactá-los para distribuição rápida através da Web.

Os repositórios de arquivos cabinet são criados com uma ferramenta da Microsoft chamada CABarc. Atualmente, ela está disponível para carregamento por download gratuito no seguinte endereço:

http://www.microsoft.com/workshop/prog/cab/

Usando CABarc, você pode compactar em um único repositório de arquivos, que possui a extensão de arquivo .cab, todos os arquivos de classe e outros arquivos necessários para um applet. Para especificar esse repositório

de arquivos, um parâmetro chamado cabbase é usado com a tag <PARAM> em HTML, e o valor de cabbase é definido como o nome do arquivo .cab. A seguir está um exemplo:

```
<APPLET CODE="DanceFever.class" WIDTH=200 HEIGHT=450>
<PARAM NAME="cabbase" VALUE="DanceFever.cab">
</APPLET>
```

Assim como o atributo ARCHIVE, o parâmetro cabbase será ignorado pelos navegadores da Web que não ofereçam suporte à sua utilização.

Os recursos de repositório de arquivos Java da Netscape e da Microsoft funcionam nas edições atuais dos navegadores da Web dessas empresas e não há garantia de seu suporte em outros navegadores. Se você utiliza uma das duas soluções, deve armazenar o repositório de arquivos e os arquivos específicos que o compõem em seu servidor da Web. Desse modo, todos que possuírem um navegador compatível com Java poderão usar o applet.

Passando parâmetros para applets

Nos aplicativos Java, você pode passar parâmetros para o método main( ) usando argumentos na linha de comando. Então, você pode analisar esses argumentos dentro do corpo de sua classe e o aplicativo agirá de acordo com os argumentos recebidos.

No entanto, os applets não possuem uma linha de comando. Eles podem receber diferentes entradas do arquivo HTML que contém a tag <APPLET> ou <OBJECT>, através da utilização de parâmetros de applet. Para definir e manipular parâmetros em um applet, você precisa de duas coisas:

- Uma tag de parâmetro especial no arquivo HTML.
- Código em seu applet para analisar esses parâmetros.

Os parâmetros de applet possuem duas partes: um nome, que é simplesmente algo que você escolhe, e um valor, que determina o valor desse parâmetro em particular. Por exemplo, você pode indicar a cor do texto em um applet usando um parâmetro com o nome color e o valor red. Você pode determinar a velocidade de uma animação usando um parâmetro com o nome speed e o valor 5.

No arquivo HTML que contém o applet incorporado, você indica cada parâmetro usando a tag <PARAM>, que possui dois atributos para o nome e para o valor, chamados (propositadamente) NAME e VALUE. A tag <PARAM> fica entre as tags <APPLET> de abertura e fechamento, como no seguinte:

```
<APPLET CODE="QueenMab.class" WIDTH=100 HEIGHT=100>
<PARAM NAME=font VALUE="TimesRoman">
<PARAM NAME=size VALUE="24">
A Java applet appears here.
</APPLET>
```

Esse exemplo em particular define dois parâmetros para o applet QueenMab: um chamado font, com o valor TimesRoman, e outro chamado size, com o valor 24.

A utilização da tag <PARAM> é a mesma para os applets que usam a tag <OBJECT> em vez de <APPLET>.

Os parâmetros são passados para seu applet quando este é carregado. No método init( ) de seu applet, você pode recuperar esses parâmetros usando o método getParameter( ). O método getParameter( ) recebe um argumento (um string representando o nome do parâmetro que você está procurando) e retorna um string contendo o valor correspondente a esse parâmetro. (Assim como no caso dos argumentos em aplicativos Java, todos os valores de parâmetro são retornados como strings.) Para obter o valor do parâmetro font do arquivo HTML, você poderia ter uma linha como a seguinte em seu método init( ):

```
String theFontName = getParameter("font");
```

Os nomes dos parâmetros especificados em PARAM e os nomes dos parâmetros presentes em getParameter( ) devem corresponder completamente, incluindo a mesma utilização de letras maiúsculas e minúsculas. Em outras palavras, <PARAM NAME="eecummings"> é diferente de <PARAM NAME="EECummings">. Se seus parâmetros não estiverem sendo passados corretamente para seu applet, certifique-se de que as letras maiúsculas e minúsculas do parâmetro sejam correspondentes.

Observe que, se um parâmetro esperado não tiver sido especificado no arquivo HTML, getParameter( ) retornará null. Mais freqüentemente, você vai querer testar a existência de um parâmetro null e fornecer um padrão razoável, como se vê a seguir:

```
if (theFontName == null)
    theFontName = "Courier";
```

Lembre-se de que getParameter( ) retorna strings; se quiser que um parâmetro seja algum outro objeto ou tipo, você mesmo precisa convertê-lo. Por exemplo, considere o arquivo HTML do applet QueenMab. Para analisar o parâmetro sizee atribuí-lo a uma variável inteira chamada theSize, você poderia usar as seguintes linhas:

```
int theSize;
String s = getParameter("size");
if (s == null)
    theSize = 12;
else theSize = Integer.parselnt(s);
```

Crie um exemplo de applet que utilize essa técnica. Você modifica o applet Palindrome de modo que ele apresente um nome específico; por exemplo, Dennis and Edna sinnedouNo, sir, prefer prison. O nome é passado para o applet por um parâmetro HTML. O projeto será chamado NewPalmdrome.

Comece copiando a classe Palindrome original, com uma alteração para refletir o novo nome de classe, como se vê na listagem 8.4.

DIGITE **Listagem 8.4.** O início do texto de newpal indrome. java.

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class NewPalindrome extends java.applet.Applet {
6:       Font f = new Font("TimesRoman", Font.BOLD, 36);
7:
8:       public void paint(Graphics screen) {
9:          screen.setFont(f);
10:          screen.setColor(Color.red);
11:          screen.drawString("Go hang a salami, I'm a lasagna hog.", 5,
             40);
12:       }
13: }
```

O primeiro item que precisa ser incluído nessa classe é um lugar em que o parâmetro palindrome possa ser armazenado. Como você vai precisar desse nome em todo o applet, inclua uma variável de instância para ele imediatamente após a variável da fonte:

```
String palindrome;
```

Para definir um valor para o nome, você precisa obter o parâmetro. O melhor lugar para se manipular os parâmetros de um applet é dentro de um método init( ). O método init( ) é definido de forma semelhante a paint( ) (public, sem argumentos, e um tipo de retorno void). Certifique-se de que, ao se testar um parâmetro, seja feito o teste de um valor null. Se um palíndromo não for indicado, o padrão, nesse caso, é apresentar Dennis and Edna sinned, como ilustra o seguinte:

```
public void init( ) {
 palindrome = getParameter("palindrome");
   if (palindrome == null)
   palindrome = "Dennis and Edna sinned";
 }
```

Quando esse método tiver sido incluído, restará apenas modificar o método pai nt( ). O método drawString( ) original era como segue:

```
screen.drawString("Go hang a salami, I'm a lasagna hog.", 5, 50);
```

Para desenhar o novo string que você armazenou na variável de instância name, basta substituir essa variável pelo string literal:

```
screen.drawString(palindrome, 5, 50);
```

A listagem 8.5 mostra o resultado final da classe NewPalindrome. Compile-a para que você tenha um arquivo de classe pronto.

DIGITE

Listagem 8.5. O texto completo de newpalindrome.java.

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.awt.Color;
4:
5: public class NewPalindrome extends java.applet.Applet {
6:    Font f = new Font("TimesRoman", Font.BOLD, 36);
7:    String palindrome;
8:
9:    public void paint(Graphics screen) {
10:       screen.setFont(f);
11:       screen.setColor(Color.red);
12:       screen.drawString(palindrome, 5, 50);
13:    }
14:
15:    public void init( ) {
16:       palindrome = getParameter("palindrome");
17:         if (palindrome •• null)
18:       palindrome = "Dennis and Edna sinned";
19:    }
20: }
```

Agora, crie o arquivo HTML que contém esse applet. A listagem 8.6 mostra uma nova página da Web para o applet NewPalindrome.

DIGITE

Listagem 8.6. O texto completo de newpalindrome.html.

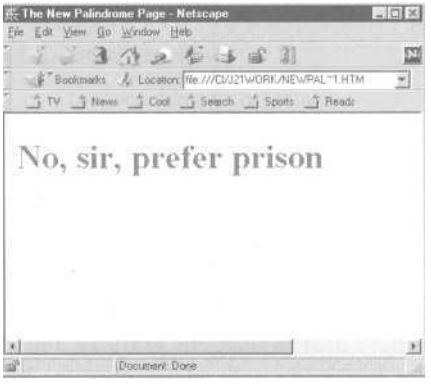
```
1: <HTHL>
2: <HEAD>
3: <TITLE>The New Palindrome Page</TITLE>
4: </HEAD>
5: <B0DY>
6: <P>
7: <APPLET CODE="NewPalindrome.class" WIDTH=600 HEIGHT=100>
8: <PARAM NAME=palindrome VALUE="No, sir, prefer prison">
9: Your browser does not support Java!
10: </APPLET>
11: </B0DY>
12: </HTML>
```

Observe a tag <APPLET>, que identifica o arquivo de classe para o applet e a largura e altura apropriadas (600 e 100, respectivamente). Imediatamente abaixo dela (linha 8) está a tag <PARAM>, que é usada para passar o palíndromo para o applet. Nesse exemplo, o parâmetro de NAME é palindrome e o VALUE é o string No, sir, prefer prison.

O carregamento desse arquivo HTML no Netscape Navigator produz o resultado ilustrado na Figura 8.5.

Figura 8.5

A *página NewPalindrome.html carregada* com o *Netscape Navigator.*



Se nenhum palíndromo for especificado no código de NewPalindrome, o padrão será Dennis and Edna sinned. A listagem 8.7 cria um arquivo HTML sem nenhuma tag de parâmetro para NAME.

Listagem 8.7. O texto completo de newpalindrome2.HTML

```
1: <HTML>
2: <HEAD>
3: <TITLE>The New Palindrome Page</TITLE>
4: </HEAD>
5: <B0DY>
6: <P>
7: <APPLET CODE="NewPalindrome.class" WIDTH=600 HEIGHT=100>
8: Your browser does not support Java!
9: </APPLET>
10: </B0DY>
```

Como nenhum nome foi fornecido aqui, o applet utiliza o padrão e o resultado é o que você poderia esperar. Dê uma olhada na Figura 8.6.

Resumo

É questionável se os applets permanecem no foco do desenvolvimento Java atualmente, há mais de dois anos do primeiro lançamento da linguagem para o público.

Entretanto, os applets continuam sendo o maior uso público de Java, pois eles aparecem em milhares de sites da World Wide Web. De acordo com o mecanismo de pesquisa AltaVista, no endereço http://www.altavista.digital.com, existem mais de 1.200.000 páginas da Web contendo applets.

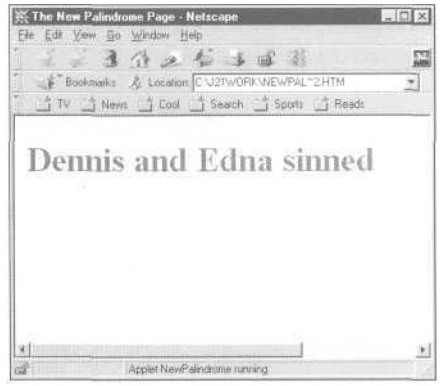


Figura 8.6
A *página NewPalindromeZ.html carregada* com o *Netscape Navigator.*

Como eles são executados e apresentados dentro de páginas da Web, os applets podem usar as imagens gráficas, a interface com o usuário e a estrutura de eventos fornecida pelo navegador da Web. Esse recurso proporciona ao programador de applet bastante funcionalidade sem muito trabalho extra.

Hoje, você aprendeu os fundamentos da criação de applets, incluindo os seguintes itens:

- Todos os applets são subclasses da classe java.applet.Applet, que fornece o comportamento de que o programa necessita para ser executado dentro de um navegador da Web.
- Os applets possuem cinco métodos principais que cobrem as atividades que um applet realiza ao ser executado: init( ),start( ),stop( ),destroy( )epaint( ). Esses métodos são anulados para fornecer funcionalidade em um applet.
- Os applets são colocados em páginas da Web usando-se a tag <APPLET> em HTML ou uma ferramenta de desenvolvimento da Web que possa manipular applets. Quando um navegador compatível com Java carrega uma página que contém um applet, ele carrega o arquivo de classe descrito com a tag <APPLET>.
- Para diminuir o tempo que leva para se carregar por download um applet de um servidor da Web, você pode usar repositórios de arquivos Java, o atributo ARCHIVE da Netscape e o parâmetro cabbase da Microsoft.
- Os applets podem receber informações de uma página da Web usando-se a tag <PARAM> em associação com um applet. Dentro do corpo de seu applet, você pode acessar esses parâmetros usando o método getParameter( ).

Perguntas e respostas

P **Tenho um applet que recebe parâmetros e um arquivo HTML que lhe passa esses parâmetros, mas quando meu applet é executado, recebo apenas valores null. O que está acontecendo?**

R Os nomes de seus parâmetros (no atributo NAME) combinam exatamente com os nomes que você está testando em getParameter( ) ? Eles devem ser exatamente iguais, incluindo letras maiúsculas e minúsculas, para que a combinação seja feita. Certifique-se também de que suas tags <PARAM> estejam entre as tags <APPLET> de abertura e fechamento e que você não tenha digitado nada errado.

P **Alguma configuração especial é necessária para que meu software servidordaWeb possa apresentar applets Java em páginas da Web?**

R Muito pouco é exigido com relação ao servidor para se oferecer applets Java junto a documentos HTML, arquivos de imagem e outros. Muitos dos pacotes de servidor disponíveis comercialmente oferecem suporte à Java. Se eles não oferecerem, a configuração do servidor para apresentar applets Java exige que a extensão de arquivo .classseja associada ao tipo MIME appl ication/ octet-stream. Consulte a documentação de seu servidor para obter mais informações a respeito de tipos MIME.

P **Como os applets não possuem uma linha de comando ou um fluxo de saída padrão, como posso produzir uma saída de depuração simples, como System.out.println( ), em um applet?**

R Dependendo de seu navegador ou de outro ambiente compatível com Java, você pode ter uma janela de console em que a saída de depuração (o resultado de System.out.println( )) aparece, ou então ela pode ser gravada como um arquivo de log. (O Netscape possui um Java Console no menu Options; o Internet Explorer usa um arquivo de log Java que você deve ativar escolhendo Options | Advanced.)

Você pode continuar a imprimir mensagens usando System.out.println( ) em seus applets — basta lembrar-se de removê-los quando tiver terminado, para que eles não confundam seus usuários reais.

SEMANA 2

Dia 9

Melhorando a aparência dos programas com imagens gráficas, fontes e cores

Uma das melhores maneiras de impressionar um leigo em programação é com um programa que apresenta imagens gráficas. Talvez o tio Walter não aprecie as sutilezas de um loop for bem construído ou uma hierarquia de classe elegante, mas mostre a ele sua seqüência animada de uma criança brincando de roda e ele ficará impressionado com sua perspicácia em programação.

Hoje, você aprenderá a ganhar amigos e influenciar as pessoas, escrevendo applets que usam imagens gráficas, fontes e cores.

Para usar recursos gráficos em seus programas, você utiliza classes do pacote java.awt, que proporciona o melhor acabamento visual da linguagem Java. Com essas classes, você desenhará texto e figuras, como círculos e polígonos, em um applet. Você aprenderá a usar diferentes fontes e cores para as figuras que desenhar.

Você também começará a usar os aprimorados recursos de desenho do Java2D, um conjunto de classes apresentado com a linguagem Java 2 que oferece alguns recursos atraentes:

- Objetos anti-alias
- Padrões de preenchimento de gradiente
- Linhas de desenho de espessuras diferentes

A classe Graphics

Um modo de pensar em um applet é como uma tela para operações gráficas. Você já usou o método drawString( ) para desenhar texto em um applet. A fonte e a cor do texto foram escolhidas antes de se desenhar os caracteres, da mesma maneira que um artista escolheria uma cor e um pincel antes de pintar.

Texto não é o único elemento que você pode desenhar em uma janela de applet. E possível desenhar linhas, elipses, círculos, arcos, retângulos e outros polígonos.

A maioria das operações básicas de desenho é composta de métodos definidos na classe Graphics. Em um applet, você não precisa criar um objeto Graphi cs para desenhar algo — como você deve se lembrar, um dos parâmetros do método paint( ) é um objeto Graphics. Esse objeto representa a janela do applet e seus métodos são usados para desenhar no applet.

A classe Graphi cs faz parte do pacote j ava. awt; portanto, todos os applets que desenham algo devem usar a instrução import para tornar a classe Graphics disponível no programa.

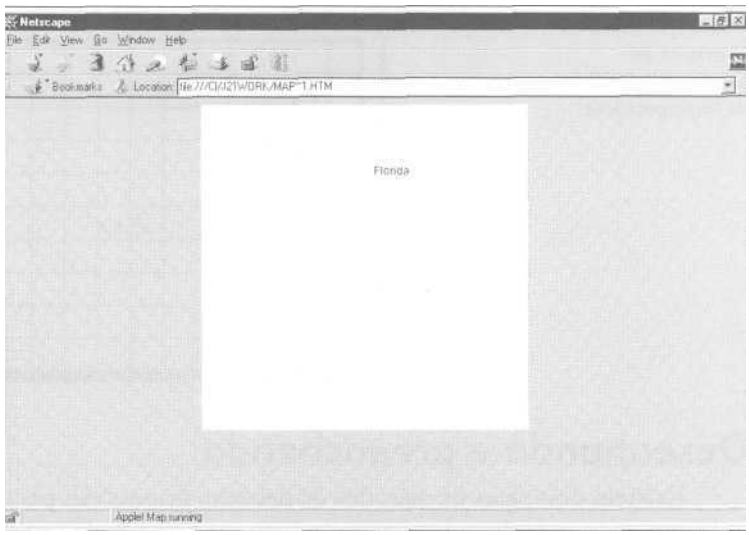
A listagem 9.1 é um applet simples que usa o método drawStri ng ( ) para apresentar texto, conforme foi feito anteriormente com o applet Palindrome.

DIGITE

Listagem 9.1. O texto inicial demap.java.

```
1: import java.awt.Graphics;
2:
3: public class Map extends java.applet.Applet {
4:     public void paint(Graphics screen) {
5:         screen.drawString("Flori da", 185, 75);
6:     }
7: }
```

Figura 9.1
Desenhando texto em uma janela App/ef.



Todos os comandos de desenho básicos que você vai aprender hoje serão métodos Graphi cs que são chamados dentro do método pai nt ( ) de um applet. Esse é um lugar ideal para todas as operações de desenho, pois o método paint( ) é chamado automaticamente sempre que a janela do applet precisa ser reapresentada. Se a janela de outro programa se sobrepõe ao applet e ele precisa ser redesenhado, a colocação de todas as operações de desenho no método paint( ) garante que não falte nenhuma parte do desenho.

Continue a aumentar o applet Map com cada um dos métodos abordados nesta seção.

O sistema de coordenadas aráficas

Assim como acontece com drawString( ), todos os métodos de desenho possuem argumentos que indicam coordenadas x,y. Alguns exigem mais do que um conjunto de coordenadas, como uma linha, que possui uma coordenada x,y para identificar seu ponto inicial e outra para seu ponto final.

O sistema de coordenadas da linguagem Java usa pixels como unidade de medida. A coordenada da origem 0,0 está no canto superior esquerdo da janela Applet. O valor das coordenadas x aumentam à direita de 0,0 e as coordenadas y aumentam para baixo. Isso difere de outros sistemas de desenho, em que a origem 0,0 fica no canto inferior esquerdo e os valores de y aumentam para cima.

Todos os valores em pixel são inteiros — você não pode usar números decimais para apresentar algo entre valores inteiros.

A Figura 9.2 representa visualmente o sistema de coordenadas gráficas da linguagem Java, com a origem em 0,0. Dois dos pontos de um retângulo estão em 20.20 e 60.60

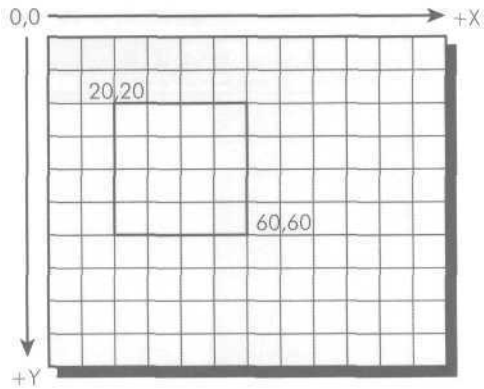


Figura 9.2
O sistema de coordenadas gráficas da linguagem Java.

Desenhando e preenchendo

Existem dois tipos de métodos de desenho disponíveis para muitas das figuras que você pode desenhar em um applet: métodos de desenho, que elaboram um contorno do objeto, e métodos de preenchimento, que preenchem o objeto com a cor atual. Em cada tipo de método, o contorno do objeto também é desenhado com a cor atual.

Nota

Você também pode desenhar arquivos gráficos de bitmap, como arquivos GIF e JPG, usando a classe Image. Você aprenderá sobre isso amanhã.

Linhas

O método drawLine( ) é usado para desenhar uma linha entre dois pontos. O método recebe quatro argumentos: as coordenadas x e y do ponto inicial e as coordenadas x e y do ponto final, como segue:

```
drawLine(x1, y1, x2, y2);
```

Esse método desenha uma linha do ponto (xl, yl) até o ponto (x2, y2). A espessura da linha fica fixa em um 1 pixel.

Insira a seguinte instrução no método paint( ) do applet Map:

```
screen.drawLine(185,80,222,80);
```

Isso desenha uma linha de 185,80 a 222,80 — um sublinhado sob o texto Fl ori da, como você pode ver na Figura 9.3, uma parte da janela Applet.

FÍGURA 9.3
Adicionando uma linha no applet.

Florida

Nota

Para evitar o choque que pode resultar da troca freqüente entre este texto e seu editor de código-fonte Java, a versão final do Map.java está listada por completo no fim desta seção. Até lá, você pode acompanhar o texto e introduzir o código Java completo de uma vez só.

Retângulos

Existem métodos Graphics para dois tipos de retângulos: retângulos normais e aqueles com cantos arredondados (como as bordas das teclas na maioria dos teclados de computador).

Você pode desenhar os dois tipos de retângulos em forma de esboço ou preenchidos com a cor atual.

Para desenhar um retângulo normal, use o método drawRect( ) para os contornos e o método fil lRect( ) para figuras preenchidas.

Esses dois métodos recebem quatro argumentos:

- As coordenadas x e y do canto superior esquerdo do retângulo.
- A largura do retângulo.
- A altura do retângulo.

Inclua a seguinte instrução no applet Map:

```
screen.drawRect(2, 2, 345, 345);
```

Isso insere um contorno retangular ajustado nas margens externas da janela Applet. Se o método f i 11 Rect ( ) tivesse sido usado, um retângulo cheio teria preenchido a maior parte da área do applet e teria se sobreposto ao texto sublinhado Florida.

Os retângulos com cantos arredondados exigem os métodos drawRoundRect( ) e fillRoundRect( ). Eles recebem os mesmos primeiros quatro argumentos que os retângulos normais, com dois argumentos incluídos no final.

Esses dois últimos argumentos definem a largura e a altura da área em que os cantos são arredondados. Quanto maior a área, mais arredondados serão os cantos. Você pode até fazer um retângulo parecer um círculo ou uma elipse, tornando esses argumentos suficientemente grandes.

A Figura 9.4 mostra vários exemplos de retângulos com cantos arredondados. Um retângulo tem uma largura de 30 e uma altura de 10 para cada canto arredondado. Outro tem largura de 20 e altura de 20 e assemelha-se mais a um círculo do que a um retângulo.

Inclua a seguinte instrução no método paint( ) do applet Map:

```
screen.drawRoundRect(182,61,43,24,10,8);
```

Isso desenha um retângulo arredondado nas coordenadas 182,61, com uma largura de 43 pixels e uma altura de 24. A área retangular de cada canto arredondado possui 10 de largura e 8 de altura. O resultado está ilustrado na Figura 9.5, um close-up de uma parte do applet.

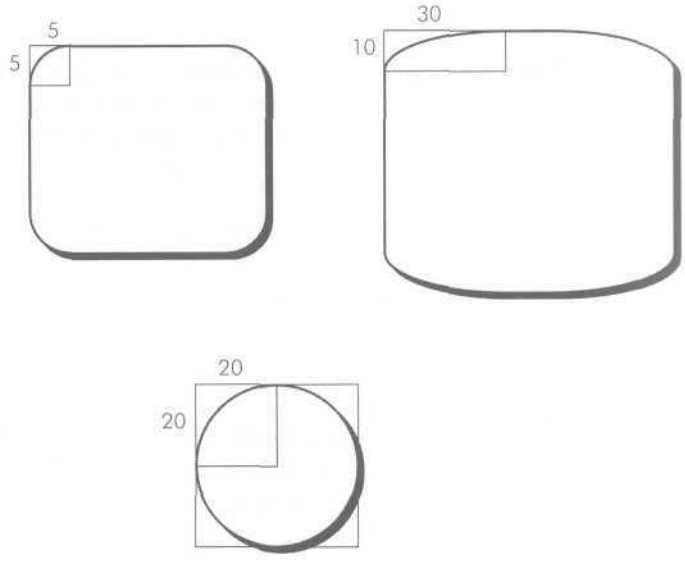


Figura 9.4
Retângulos com cantos *arredondados.*

Figura 9.5
Adicionando um retângulo arredondado ao applet Polygons.

Os polígonos podem ser desenhados com os métodos drawPolygon( ) e fill Polygon ( ).

Para desenhar um polígono, você precisa das coordenadas x,y de cada ponto dele. Os polígonos podem ser considerados como uma série de linhas interligadas — uma linha é desenhada do ponto inicial até o ponto final; esse ponto final é usado para iniciar uma nova linha etc.

Você pode especificar essas coordenadas de duas maneiras:

- Como um par de arrays de inteiros, um contendo todas as coordenadas x e outro contendo todas as coordenadas y.
- Como um objeto Polygon que é criado usando-se um array de inteiros de coordenadas x e um array de inteiros de coordenadas y.

O segundo método é mais flexível, pois permite que sejam inseridos pontos individualmente em um polígono, antes que ele seja desenhado.

Além das coordenadas x e y, você deve especificar o número de pontos do polígono. Você não pode especificar mais coordenadas x,y do que os pontos disponíveis, ou mais pontos do que as coordenadas x,y definidas para eles. Um erro de compilador resultará em qualquer um dos casos.

Para criar um objeto Pol ygon, o primeiro passo é criar um polígono vazio com uma instrução new Pol ygon ( ), como no seguinte:

```
Polygon poly = new Polygon( );
```

Como alternativa, você pode criar um polígono a partir de um conjunto de pontos, usando arrays de inteiros. Isso exige uma chamada ao construtor Polygon(iní/" *], int[ ], int)*, que especifica o array de pontos x, o array de pontos y e o número de pontos. O exemplo a seguir mostra o uso desse construtor:

```
int x[ ] = { 10, 20, 30, 40, 50 };
int y[ ] = { 15, 25, 35, 45, 55 };
int points = x.length;
Polygon poly = new Polygonfx, y, points);
```

Depois que um objeto Polygon foi criado, você pode adicionar pontos a ele usando o método addPoint( ) desse objeto. Isso requer coordenadas x,y como argumentos e inclui o ponto no polígono. A seguir está um exemplo:

```
poly.addPoint(60, 65);
```

Quando você tiver um objeto Polygon que possua todos os pontos necessários, poderá desenhá-lo com os métodos drawPolygon( ) ou fi 11 Polygon ( ). Isso exige apenas um argumento — o objeto Polygon, como se vê a seguir:

```
screen.drawPolygon(poly);
```

Se você usar drawPolygon( ) em Java 1.02, poderá fechar o polígono tornando sua última coordenada x,y igual à primeira. Caso contrário, o polígono ficará aberto em um lado.

O método f i 11 Pol ygon ( ) fecha o polígono automaticamente, sem exigir pontos concordantes.

O comportamento de drawPolygon( ) mudou após a versão 1.02 da inguagem Java. Nas versões 1.1 e 2, drawPolygon( ) fecha automaticamente um polígono, da mesma maneira que ocorre com fillPolygon ( ). Se quiser criar um polígono aberto com essas versões da linguagem, você pode usar o método drawPolyl ine( ). Ele funciona exatamente como drawPolygon ( ) funcionava na linguagem Java 1.02.

Adicione as seguintes instruções ao método pai nt ( ) do applet Map para ver os polígonos em ação:

```
int x[ ] = { 10, 234, 253, 261, 344, 336, 295, 259, 205, 211,
    195, 191, 120, 94, 81, 12, 10 };
int y[ ] = { 12, 15, 25, 71, 209, 278, 310, 274, 188, 171, 174,
    118, 56, 68, 49, 37, 12 };
int pts = x.length;
Polygon poly = new Polygon(x, y, pts);
screen.drawPolygon(poly);
```

A classe Polygon faz parte do pacote java.awt; portanto, você deve torná-la disponível incluindo a seguinte instrução no início do applet Map:

```
import java.awt.Polygon;
```

A Figura 9.6 mostra como fica o applet Map com o polígono incluído em tudo o que já está desenhado.

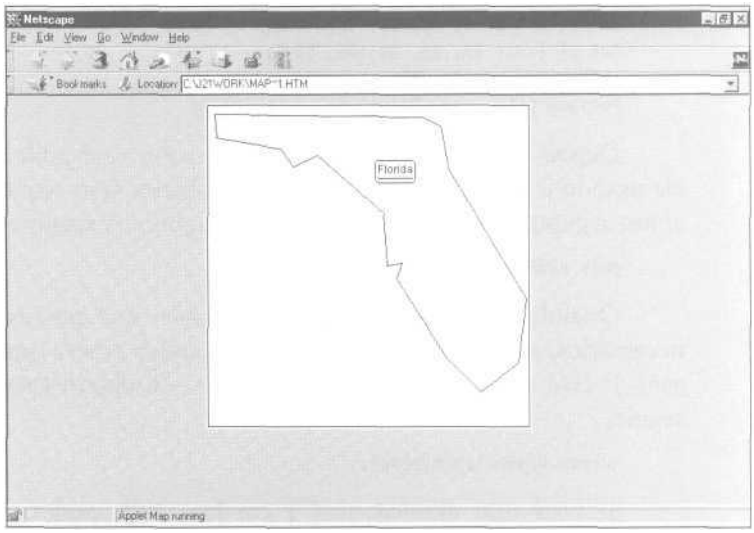


Figura 9.6
Adicionando um polígono ao app/ef.

Elipses

Os métodos drawOval ( ) e f i11 Oval ( ) são usados para desenhar círculos e elipses.

Esses métodos exigem quatro argumentos:

- As coordenadas x,y da elipse.
- A largura e a altura da elipse, que são de mesmo tamanho nos círculos.
- Como uma elipse não possui nenhum canto, você pode estar se perguntando a que as coordenadas x,y se referem. As elipses são manipuladas da mesma maneira que os cantos de retângulos arredondados. As coordenadas x,y se referem ao canto superior esquerdo da área em que a elipse é desenhada, e ficará à esquerda e acima da elipse real em si.

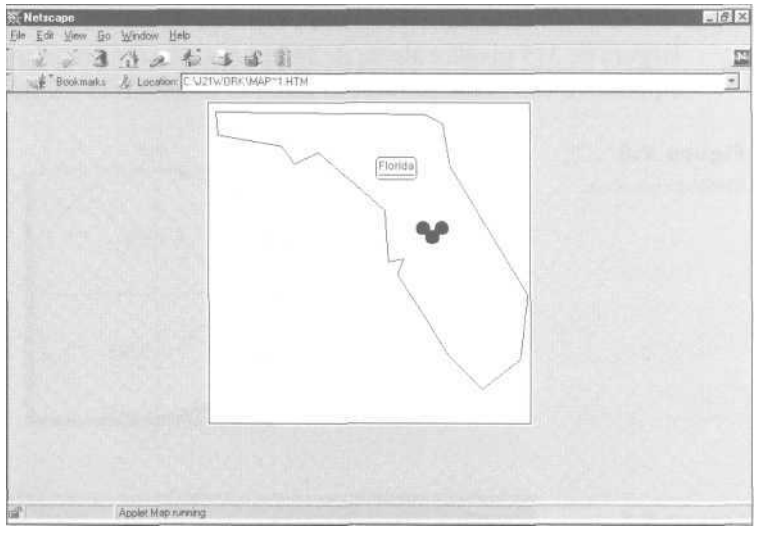
Volte ao applet Map e inclua as seguintes instruções:

```
screen.fi11 Oval(235,140,15,15);
screen.fillOval(225,130,15,15);
screen.fi11Oval(245,130,15,15);
```

Tratam-se de métodos f i 11 e não de métodos draw; portanto, eles criam três círculos negros interligados em um sombreado na Flórida central, como se vê na Figura 9.7.

Figura 9.7

Adicionando um trio de círculos ao applet.



Arcos

De todas as operações de desenho, os arcos são os mais complexos para se construir. Um arco é parte de uma elipse e é implementado em Java como uma elipse parcialmente desenhada.

Os arcos são desenhados com os métodos drawArc( ) e fillArc( ), que recebem seis argumentos:

- As coordenadas x,y da elipse.
- A largura e a altura da elipse.
- O ângulo em que se deve iniciar o arco.
- O número de graus percorridos pelo arco.

Os quatro primeiros argumentos são iguais aos de uma elipse e funcionam da mesma maneira.

O ângulo inicial do arco varia de 0 a 359 graus, na direção anti-horária. Em uma elipse circular, 0 grau é o mesmo que 3 horas, 90 graus é o mesmo que 12 horas, 180 graus é o mesmo que 9 horas e 270 graus é o mesmo que 6 horas.

O número de graus percorridos por um arco varia de 0 a 359 graus na direção anti-horária e de 0 a -359 graus na direção horária.

A Figura 9.8 mostra como os dois últimos argumentos são calculados.

Os arcos preenchidos são desenhados como se fossem seções de uma torta; em vez de unir os dois pontos finais, ambos são unidos no centro da elipse que forma o arco.

A seguir, está um exemplo de uma chamada de método drawArc( ):

```
screen.drawArc(20,25,315,150,5,-190);
```

Essa instrução desenha um arco de elipse com as coordenadas 20,25, largura de 315 pixels e altura de 150 pixels. O arco começa na marca de 5 graus e percorre 190 graus na direção horária. O arco está exibido na Figura 9.9.

Figura 9.8
/Medindo um arco.

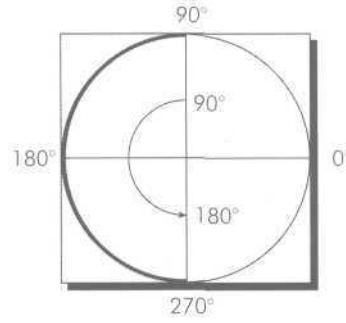
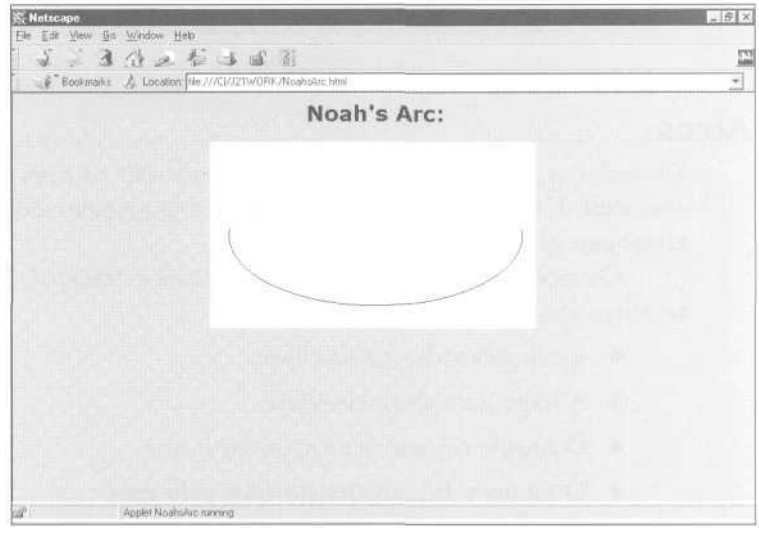


Figura 9.9
Um arco.



Como último detalhe do applet Map, vários arcos pequenos, com quatro argumentos que não mudam, serão desenhados:

- A elipse de cada arco terá uma largura e uma altura de 10 pixels, tornando as elipses circulares.
- Cada arco começará em 0 grau e percorrerá 180 graus no sentido horário, transformando-os em meios-círculos.

As coordenadas x,y do arco mudarão e dois loops for circularão por um intervalo de valores x e y.

Insira as seguintes instruções no método paint( ) do applet Map:

```
for (int ax = 50; ax < 150; ax += 10)
    for (int ay = 120; ay < 320 ; ay += 10)
      screen.drawArc(ax, ay, 10, 10, 0, -180);
```

Colocar um loop for dentro de outro pode parecer confuso. Aqui estão as seis primeiras coordenadas x,y que são criadas pelo loop:

50,120

50,130

50,140

50,150

50,160

50,170

Como você pode ver, a coordenada x — especificada por ax — não muda. Ela não mudará até que o loop ay inteiro tenha percorrido seu caminho. Quando isso acontecer, ax será aumentado de 10 e o loop ay será executado completamente outra vez.

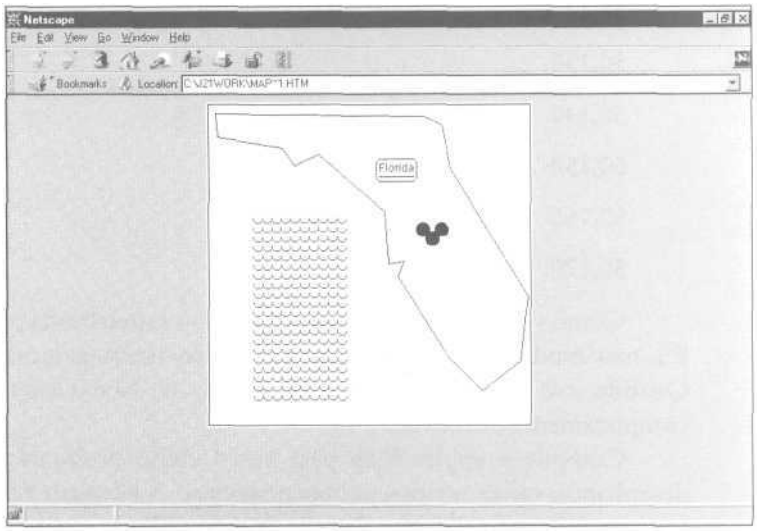
Compile o applet Map para ver o efeito produzido por esses loops, desenhando vários meios-círculos pequenos. A listagem 9.3 mostra o código-fonte completo final de Map.java, incluindo todas as instruções de desenho que foram abordadas durante esta seção.

DIGITE **Listagem 9.3.** O texto completo e final de map.java.

```
1: import java.awt.Graphics;
2: import java.awt.Polygon;
3:
4: public class Map extends java.applet.Applet {
5:     public void paint(Graphics screen) {
6:       screen.drawString("Florida", 185, 75);
7:       screen.drawLine(185,80,222,80);
8:       screen.drawRect(2, 2, 345, 345);
9:       screen.drawRoundRect(182,61,43,24,10,8);
10:      int x[ ] = { 10, 234, 253, 261, 344, 336, 295, 259, 205, 211,
11:        195, 191, 120, 94, 81, 12, 10 };
12:      int y[ ] = { 12, 15, 25, 71, 209, 278, 310, 274, 188, 171, 174,
13:        118, 56, 68, 49, 37, 12 };
14:      int pts = x.length;
15:      Polygon poly = new Polygon(x, y, pts);
16:      screen.drawPolygon(poly);
17:      screen.fillOval(235,140,15,15);
18:      screen.fillOval(225,130,15,15);
19:      screen.fillOval(245,130,15,15);
20:      for (int ax = 50; ax < 150; ax += 10)
21:        for (int ay = 120; ay < 320 ; ay += 10)
22:          screen.drawArc(ax, ay, 10, 10, 0, -180);
23:    }
24: }
```

A Figura 9.10 mostra o applet Map que foi pintado com os métodos básicos de desenho da linguagem Java.

Figura 9.10
O *applet Map.*



Embora nenhum cartografo tema por seu trabalho por causa desta elaboração de mapa, the applet combina uma amostra da maioria dos recursos de desenho que estão disponíveis através da classe Graphi cs. Um applet como esse poderia ser expandido usando-se objetos Font e Color, e as operações de desenho poderiam ser reorganizadas para melhorar o produto final.

Copiando e fazendo a limpeza

A classe Graphics também inclui alguma funcionalidade de recortar e colai envolvendo a janela Applet:

- O método copyArea( ), que copia uma região retangular da janela Applet em outra região da janela.
- O método clearRect( ), que faz a limpeza de uma região retangulai da janela Applet.

O método copyArea( ) recebe seis argumentos:

- As coordenadas x,y da região retangular a ser copiada.
- A largura e a altura dessa região.
- As distâncias horizontal e vertical, em pixels, a se afastar da regiãc antes de apresentar uma cópia dela.

A instrução a seguir copia uma região de 100x100 pixels em uma área a 50 pixels à direita e 25 pixels para baixo:

```
screen.copyArea(0,0,100,100,50,25);
```

O método clearRect( ) recebe os mesmos quatro argumentos que os métodos drawRect( ) e fi11 Rect( ), e ele preenche a região retangular com a cor de fundo atual do applet. Você aprenderá a definir a cor de fundo posteriormente, ainda hoje.

Você pode determinar o tamanho da janela através do método size( ), se quiser fazer a limpeza da janela Applet inteira. Isso retorna um objeto Dimension, que possui variáveis width e height; elas representam as dimensões do applet.

Para fazer a limpeza do applet inteiro, você pode usar o método si ze ( ), que retorna um objeto Dimension representando a largura e a altura do applet. Em seguida, você obtém os valores reais da largura e da altura, usando as variáveis de instância width e height, como na instrução a seguir:

```
screen.clearRect(0, 0, size( ).width, size( ).height);
```

O método size( ) foi renomeado após a linguagem Java 1.02. Ele ainda funciona na linguagem Java 2, mas o compilador dará a advertência depreciation, que quer dizer que um método de substituição está presente. O método getSize( ) na linguagem Java 2 funciona exatamente como size( ). Aalteração do nome é parte do esforço da Sun para ter nomes de métodos consistentes em toda a biblioteca de classes.

Texto e fontes

Os objetos da classe java.awt.Font são usados para se utilizar o método drawString( ) com diferentes fontes. Os objetos Font representam o nome, o estilo e o tamanho em pontos de uma fonte. Outra classe, FontMetri cs, fornece métodos para se determinar o tamanho dos caracteres que estão sendo apresentados com uma fonte específica, o que pode ser usado para detalhes como formatação e centralização de texto.

Criando obietos Font

Um objeto Font é criado enviando-se três argumentos para seu construtor:

- O nome da fonte
- O estilo da fonte
- O tamanho em pontos da fonte

O nome da fonte pode ser específico, como Arial ou Garamond Old Style, e será usado se a fonte estiver presente no sistema em que o programa Java está sendo executado.

Também existem nomes que podem ser usados para se selecionar fontes internas da linguagem Java: TimesRoman, Helvetica, Courier, Dialog e Dialog-Input.

Para a linguagem Java 1.1 e versões posteriores, os nomes das fontes TimesRoman, Helvetica e Courier devem ser substituídos por com serifas, sem serifas e monoespaçadas, respectivamente. Estes nomes genéricos especificam o estilo da fonte sem nomear uma determinada família de fontes para representá-lo. Esta opção é melhor porque algumas famílias de fontes podem não estar presentes em todas as implementações de Java, de modo que a melhor opção pode ser usada para o estilo de fonte selecionado (como serifas).

Três estilos de Font podem ser selecionados usando-se as constantes Font.PLAIN, Font.BOLD e Font. ITALIC. Essas constantes são inteiros e você pode incluí-las para combinar efeitos.

O último argumento do construtor Font( ) é o tamanho em pontos da fonte.

A instrução a seguir cria uma fonte Dialog de 24 pontos, em negrito e itálico.

```
Font f = new Font("Dialog", Font.BOLD + Font.ITALIC, 24);
```

Desenhando caracteres e strinas

Para definir a fonte atual, o método setFont ( ) da classe Graphi cs é usado com um objeto Font. A instrução a seguir utiliza um objeto Font chamado f t:

```
screen.setFont(ft);
```

O texto pode ser apresentado em uma janela Applet usando-se os métodos drawString( ). Esse método utiliza a fonte selecionada no momento; ele usa o padrão, se nenhuma fonte tiver sido selecionada. Usando-se setFont ( ), uma nova fonte atual pode ser definida a qualquer momento.

O método paint( ) a seguir cria um novo objeto Font, define a fonte corrente para esse objeto e desenha o string "I'm very font of you." nas coordenadas 10,100.

```
public void paint(Graphics screen) {
    Font f = new Font("TimesRoman", Font.PLAIN, 72);
    screen.setFont(f);
    screen.drawString("I'm very font of you.", 10, 100);
}
```

Os dois últimos argumentos do método drawString( ) são as coordenadas x e y. O valor x é o início da margem esquerda do texto e y é a linha de base do string inteiro.

Localizando informações sobre uma fonte

A classe FontMetr i cs pode ser usada para se obter informações detalhadas sobre a fonte atual, como a largura ou a altura dos caracteres que ela pode apresentar.

Para se usar os métodos dessa classe, um objeto FontMetrics deve ser criado utilizando-se o método getFontMetri cs ( ). O método recebe um único argumento: um objeto Font.

A Tabela 9.1 mostra algumas das informações que você pode encontrar usando métricas de fonte. Todos esses métodos devem ser chamados em um objeto FontMetrics.

Tabela 9.1 Métodos de métrica de fonte.

Nome do *método*	Ação
stringWidth(String)	Dado um string, retorna a largura total desse string, em pixels.
charWidth(char)	Dado um caractere, retorna a largura desse caractere.
getHeight( )	Retorna a altura total da fonte.

A listagem 9.4 mostra como as classes Font e FontMetrics podem ser usadas. O applet SoLong apresenta um string no centro da janela Applet, usando FontMetrics para medir a largura do string com a fonte atual.

DIGITE

Listagem 9.4. O texto completo de solong.java.

```
1:  import java.awt.Font;
2:  import java.awt.Graphics;
3:  import java.awt.FontMetrics;
4:
5:  public class SoLong extends java.applet.Applet {
6:
7:      public void paint(Graphics screen) {
8:          Font f = new Font("Courier", Font.BOLD, 18);
9:          FontMetrics fm = getFontMetrics(f);
10:         screen.setFont(f);
11:         String s = "So long, and thanks for ali the fish.";
12:         int x = (size( ).width - fm.stringWidth(s)) / 2;
13:         int y = size( ).height / 2;
14:         screen.drawString(s, x, y);
15:     }
16: }
```

A Figura 9.11 mostra duas cópias do applet SoLong em uma página da Web, cada uma com janelas de tamanhos diferentes.

Ométodosize( ), nas linhas 12 e 13, deve ser substituído por getSize( ), se você estiver escrevendo um applet Java 1.1 ou posterior. Determinar o tamanho da janela do applet dentro dele mesmo é preferível, para se definir o tamanho exato no applet, pois isso é mais flexível. Você pode mudar o código HTML do applet na página da Web sem mudar o programa e ele ainda funcionará perfeitamente.

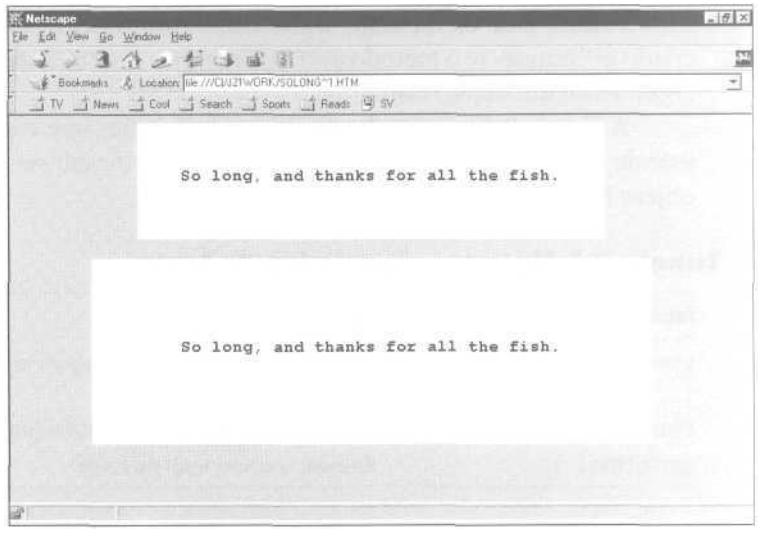


Figura 9.11
Duas *cópias* do applet SoLong.

Cor

As classes Color e ColorSpace do pacote java.awt podem ser usadas para tornar seus applets e aplicativos mais coloridos. Com essas classes, você pode definir a cor atual a ser usada nas operações de desenho, bem como a cor de fundo de um applet e de outras janelas. Você também pode converter uma cor de um sistema de descrição de cores para outro.

Por definição, a linguagem Java usa cores de acordo com um sistema de definição de cores chamado sRGB. Nesse sistema, uma cor é descrita pela quantidade de vermelho, verde e azul que contém — é daí que provém as letras R, G e B (do inglês Red, Green e Blue, respectivamente, vermelho, verde e azul). Cada um dos três componentes pode ser representado como um inteiro entre 0 e 255. A cor preta corresponde a 0,0,0 — a ausência completa de vermelho, verde ou azul. A cor branca é representada por 255,255,255 — a quantidade máxima de todas as três. Você também pode representar valores sRGB usando três números em ponto flutuante, variando de 0 a 1.0. Usando sRGB, a linguagem Java pode representar milhões de cores entre os dois extremos.

Um sistema de descrição de cores é chamado de *espaço de cor* e o sRGB é apenas um desses espaços. Há também o CMYK, um sistema usado pelas impressoras, que descreve as cores pela quantidade de ciano, magenta, amarelo e preto que elas contêm. A linguagem Java 2 oferece suporte à utilização de qualquer espaço de cor desejado, desde que seja usado um objeto ColorSpace que defina o sistema de descrição. Você também pode converter de qualquer espaço de cor para sRGB e vice-versa.

A representação de cores interna da linguagem Java usando sRGB é apenas um espaço de cor que está sendo usado em um programa. Um disposi-

tivo de saída, como um monitor ou uma impressora, também possui seu próprio espaço de cor.

Quando você apresenta ou imprime algo com uma cor designada, o dispositivo de saída pode não oferecer suporte a essa cor. Nesse caso, ela será substituída por uma cor diferente ou um padrão de *pontilhamento* será usado para se aproximar da cor indisponível. Isso acontece freqüentemente na World Wide Web, quando uma cor que não está disponível é substituída por um padrão de pontilhamento de duas ou mais cores que se aproximam da cor ausente.

A realidade prática do gerenciamento de cor é que a cor designada com sRGB não estará disponível em todos os dispositivos de saída. Se você precisar de controle mais preciso da cor, pode usar ColorSpace e outras classes do pacote java.awt.color, introduzido na linguagem Java 2.

Para a maioria dos programas, o uso interno de sRGB para definir cores será suficiente.

Usando objetos Color

Para se definir a cor de desenho atual, deve ser criado um objeto Color representando a cor ou, então, você deve usar uma das cores padrão disponíveis na classe Color.

Existem duas maneiras de se chamar o método construtor Color para criar uma cor:

- Usando três inteiros que representam o valor sRGB da cor desejada.
- Usando três números em ponto flutuante que representam o valor sRGB desejado.

Você pode especificar o valor sRGB de uma cor usando três valores int ou float. As instruções a seguir mostram exemplos de cada um deles:

```
Color c1 = new Color(0.807F,1F,0F);
Color c2 = new Color(255,204,102);
```

O objeto c1 descreve uma cor verde-néon e c2 é cor de açúcar queimado.

É fácil confundir literais de ponto flutuante como OF e 1F com números hexadecimais, os quais foram discutidos no Dia 3. As cores são freqüentemente expressas em hexadecimais, como quando uma cor de fundo é definida em uma página da Web usando a tag HTML <B0DY>. Nenhuma das classes e métodos Java com que você trabalha recebem argumentos hexadecimais; assim, quando você vir uma literal 1F ou OF, estará lidando com números em ponto flutuante.

Testando e definindo as cores atuais

A cor atual do desenho é designada pelo método setColor( ) da classe Graphics. Esse método deve ser chamado no objeto Graphi cs que representa a área em que você está desenhando. Em um applet, esse objeto é o que é passado para o método pai nt( ).

Um modo de definir a cor é usando uma das cores padrão disponíveis como variáveis de classe na classe Color.

Essas cores utdizam as seguintes variáveis Color (com os valores sRGB indicados entre parênteses):

```
black (0,0,0)                magenta (255,0,255)
blue (0,0,255)               orange (255,200,0)
cyan (0,255,255)             pink (255,175,175)
darkGray (64,64,64)          red (255,0,0)
gray (128,128,128)           white (255,255,255)
green (0,255,0)              yellow (255,255,0)
lightGray (192,192,192)
```

A instrução a seguir define a cor atual do objeto screen usando as variáveis de classe padrão:

```
screen.setColor(Color.pink);
```

Se você criou um objeto Color, ele poderá ser definido de maneira semelhante:

```
Color brush = new Color(255,204,102);
screen.setColor(brush);
```

Depois que você definir a cor atual, todas as operações de desenho ocorrerão nessa cor.

Você pode definir a cor de fundo para uma janela Applet usando os métodos setBackground( ) e setForeground( ) do próprio applet. Eles são herdados pela classe Applet a partir de uma de suas superclasses; portanto, todos os applets que você criar irão herdá-los.

O método setBackground( ) define a cor de fundo da janela Applet. Ele recebe um único argumento, um objeto Color:

```
setBackground(Color.white);
```

Há também um método setForeground( ) que é chamado nos componentes da interface com o usuário, em vez de objetos Graphics. Ele funciona como o método setColor( ), mas altera a cor de um componente de interface, como um botão ou uma janela.

Como um applet é uma janela, você pode usar setForeground( ) no método **init(**) para definir a cor para as operações de desenho. Essa cor é usada até que outra seja escolhida com setForeground( ) ou setColor( ).

Se você quiser descobrir qual é a cor atual, poderá usar o método getColor( ) em um objeto Graphics, ou os métodos getForeground( ) ou getBackground( ) da classe Applet.

A instrução a seguir define a cor atual de screen — um objeto Graphics — como sendo igual à cor de fundo de um applet:

```
screen.setColor(getBackground( ));
```

Operacões aráficas avançadas usando Java2D

Um dos aprimoramentos oferecidos com a linguagem Java 2 é o Java2D, um conjunto de classes para proporcionar imagens gráficas bidimensionais, imagens e texto de alta qualidade em seus programas. As classes Java2D ampliam os recursos das classes java.awt existentes, que manipulam imagens gráficas, como aquelas sobre as quais você aprendeu hoje. Elas não substituem as classes existentes — você pode continuar usando as outras classes e programas que as implementam.

Os recursos do Java2D incluem o seguinte:

- Padrões de preenchimento especiais, como gradientes e padronizações.
- Toques que definem a largura e o estilo de um traço de desenho.
- Anti-alias para suavizar as irregularidades dos objetos desenhados.

Espaços de coordenadas de usuário e dispositivo

Um dos conceitos introduzidos com o Java2D é a diferença entre espaço de coordenadas de um dispositivo de saída e o espaço de coordenadas a que você faz referência ao desenhar um objeto.

NOVO TERMO

Espaço de coordenadas é qualquer área bidimensional que possa ser descrita usando-se coordenadas x,y.

Para todas as operações de desenho até este ponto e para todas as operações anteriores à linguagem Java 2, o único espaço de coordenadas usado era o espaço de coordenadas de dispositivo. Você especificava as coordenadas x,y de uma superfície de saída, como uma janela Applet, e essas coordenadas eram usadas para desenhar linhas, texto e outros elementos.

O Java2D exige um segundo espaço de coordenadas, a que você faz referência ao criar um objeto e ao desenhá-lo efetivamente. Ele é chamado de *espaço de coordenadas de usuário.*

Antes que qualquer desenho bidimensional tenha ocorrido em um programa, o espaço de dispositivo e o espaço de usuário possuem as coordenadas 0,0 no mesmo lugar — o canto superior esquerdo da área de desenho.

As coordenadas 0,0 do espaço de usuário podem se mover como resultado de as operações de desenho bidimensionais estarem sendo conduzidas. Os

eixos x e y podem até se deslocar por causa de uma rotação bidimensional. Você aprenderá mais sobre os dois diferentes sistemas de coordenadas à medida que trabalhar com o Java2D.

Fazendo a coerção de um objeto Grafics2D

As operações de desenho sobre as quais você aprendeu até aqui são chamadas em um objeto Graphics que representa a área em que se está desenhando — como uma janela Applet. No caso do Java2D, esse objeto deve ser usado para criar um novo objeto Graphi cs2D, como no método pai nt ( ) a seguir:

```
public void paint(Graphics screen) {
    Graphics2D screen2D = (Graphics2D)screen;
}
```

O objeto screen2D desse exemplo foi produzido através de coerção. Trata-se do objeto screen convertido da classe Graphics para a classe Graphics2D.

Todas as operações gráficas do Java2D devem ser chamadas em um objeto Graphi cs2D. A classe Graphics2D faz parte do pacote java.awt.

Especificando os atributos de renderizacão

O próximo passo no desenho bidimensional é especificar como um objeto desenhado será renderizado. Os desenhos que não são bidimensionais podem selecionar apenas um atributo: a cor. 2D oferece uma ampla variedade de atributos para designar cor, espessura de linha, padrões de preenchimento, transparência e muitas outras características.

Cores bidimensionais

As cores são especificadas usando-se o método setColor( ), que funciona de forma igual ao método Graphics de mesmo nome. A seguir está um exemplo:

```
screen2D.setColor(Color.black);
```

Embora alguns dos métodos bidimensionais funcionem como seus correspondentes não-bidimensionais, eles devem ser chamados em um objeto Graphics2D para usar os recursos do Java2D.

Padrões de preenchimento

Os padrões de preenchimento controlam como um objeto desenhado será preenchido. No Java2D, você pode usar uma cor uniforme, um preenchimento de gradiente, textura ou um padrão de sua própria imaginação.

Um padrão de preenchimento é definido usando-se o método setPai nt ( ) de Graphi cs2D com um objeto Pai nt como único argumento. A interface Pai nt

é implementada por qualquer classe que possa ser um padrão de preenchimento, incluindo GradientPaint, TexturePaint e Color. O terceiro poderia surpreendê-lo, mas usar um objeto Col or com setPai nt ( ) é o mesmo que preencher com uma cor uniforme como padrão.

NOVO TERMO

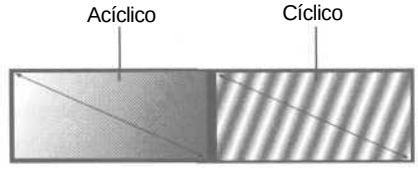
Um *preenchimento de gradiente é* uma mudança gradual de uma cor em um ponto de coordenadas para outra cor em um ponto de coordenadas diferente. A mudança pode ocorrer uma vez entre os pontos, o que é chamado de *gradiente acídico,* ou pode acontecer repetidamente, o que é um *gradiente cíclico.*

A Figura 9.12 mostra exemplos de gradientes acíclicos e cíclicos entre a cor branca e uma cor mais escura. As setas indicam os pontos entre os quais as cores se deslocam.

Os pontos de coordenadas de um gradiente não se referem diretamente aos pontos em que o objeto Graphics2D está sendo desenhado. Em vez disso, eles se referem ao espaço de usuário e podem estar até mesmo fora do objeto que está sendo preenchido com um gradiente.

Figura 9.12

Deslocamentos de *gradiente acídico e cíclico.*



A Figura 9.13 ilustra isso. Os dois retângulos do applet são preenchidos usando-se o mesmo objeto Gradi entPai nt como guia. Um modo de pensar em um padrão de gradiente é como uma peça de tecido estendida sobre uma superfície plana. As figuras que estão sendo preenchidas com um gradiente é a padronagem do tecido recortada da fazenda, e mais de um padrão pode ser recortado do mesmo tecido.

Figuro 9.13

Do/s *retângulos usando* o mesmo *GradientPaint.*

Uma chamada ao construtor Gradi entPai nt assume o seguinte formato:

GradientPaint(xl,yl,colorl,x2,y2,color2);

O ponto x1,y1 é onde a cor representada por color1 começa e x2,y2 é onde a mudança termina em color2.

Se você quiser usar uma mudança de gradiente cíclico, um argumento extra é incluído no final:

```
GradientPaint(x1,y1,color1,x2,y2,color2,true);
```

O último argumento é um valor booleano que é true para uma mudança cíclica. Um argumento false pode ser usado para mudanças acíclicas ou, então, você pode ignorar completamente esse argumento — as mudanças acíclicas representam o comportamento padrão.

Após ter criado um objeto GradientPaint, você o define como atributo de pintura atual usando o método setPaint( ). As instruções a seguir criam e selecionam um gradiente:

```
GradientPaint pat = new GradientPaint(0f,0f.Color.white,
    100f,45f,Color.blue);
screen2D.setPaint(pat);
```

Todas as operações de desenho subseqüentes no objeto screen2D usarão esse padrão de preenchimento até que outro seja escolhido.

Definindo um traço de desenho

Conforme você já aprendeu, as linhas desenhadas em todas as operações gráficas não-bidimensionais possuem 1 pixel de largura. O Java2D inclui a capacidade de variar a largura da linha de desenho, usando o método setStroke( ) com um BasicStroke.

Um construtor BasicStroke simples recebe três argumentos:

- Um valor float representando a largura da linha, sendo a norma l.0
- Um valor int determinando o estilo de terminação desenhado no final de uma linha
- Um valor int determinando o estilo da junção entre dois segmentos de linha

NOVO TERMO

Os argumentos de estilo de terminação e junção utilizam variáveis de classe BasicStroke. Os estilos *determinação* se aplicam à extremidades das linhas que não se conectam a outras linhas. Os estilos de *junção* se aplicam às extremidades das linhas que se unem a outras linhas.

Os estilos de terminação possíveis são CAP_BUTT para ausência de pontos extremos, CAP_ROUND para círculos em torno de cada extremo e CAP_SQUARE para quadrados. A Figura 9.14 mostra cada estilo de terminação. Como você pode ver, a única diferença visível entre os estilos CAP_BUTT e CAP_SQUARE é que CAP_SQUARE é mais longo, pois é desenhada uma terminação quadrada.

Figura 9.14
Estilos de terminação.

Os estilos de junção possíveis são JOIN_MITER para unir segmentos através da ampliação de suas bordas externas, JOIN_ROUND para arredondar um canto entre dois segmentos e JOIN_BEVEL para unir segmentos a uma linha reta. A Figura 9.15 mostra exemplos de cada estilo de junção.

Figura 9.15
Estilos de junção de extremidades.

As instruções a seguir criam um objeto BasicStroke e o transformam no traço atual:

```
BasicStroke pen = BasicStroke(2.0f,
    BasicStroke.CAP_BUTT,
    BasicStroke.JOINJROUND);
screen2D.setStroke(pen);
```

O traço possui uma largura de 2 pixels, extremidades planas e cantos de segmento arredondado.

Criando obietos para desenhar

Depois que você criou um objeto Graphics2D e especificou os atributos de renderização, os dois últimos passos são criar o objeto e desenhá-lo.

Os objetos desenhados no Java2D são criados através de sua definição como figuras geométricas, usando as classes do pacote java.awt.geom. Você pode desenhar cada um dos itens criados anteriormente no dia de hoje, incluindo linhas, retângulos, elipses, arcos e polígonos.

A classe Graphics2D não possui diferentes métodos para cada uma das figuras que você pode desenhar. Em vez disso, você define a figura e a utiliza como um argumento dos métodos draw( ) ou fill( ).

Linhas

As linhas são criadas usando-se a classe Line2D.Float. Essa classe recebe quatro argumentos: as coordenadas x,y de uma extremidade, seguidas das coordenadas x,y da outra. A seguir há um exemplo:

```
Line2D.Float ln = new Line2D.Float(60F,5F,13F,28F);
```

Essa instrução cria uma linha entre as coordenadas 60,5 e 13,28. Note que um F é usado com as literais enviadas como argumentos — caso contrário, o compilador Java as consideraria como inteiros.

Retângulos

Os retângulos são criados usando-se as classes Rectangle2D.Float ou Rectangle2D.Double. A diferença entre as duas é que uma recebe argumentos float e a outra recebe argumentos double.

Rectangle2D.Float recebe quatro argumentos: a coordenada x, a coordenada y, a largura e a altura. A seguir está um exemplo:

```
Rectangle2D.Float rc = new Rectangle2D.Float(10F,13F,40F,20F);
```

Isso cria um retângulo nas coordenadas 10,13 com 40 pixels de largura e 20 pixels de altura.

Elipses

NOVO TERMO

Os objetos ovais são chamados *elipses* no Java2D e eles podem ser criados com a classe El 1 i pse2D. Float. Há quatro argumentos: a coordenada x, a coordenada y, a largura e a altura.

A instrução a seguir cria uma elipse nas coordenadas 113,25 com uma largura de 22 pixels e uma altura de 40 pixels:

```
Ellipse2D.Float ee = new Ellipse2D.Float(113,25,22,40);
```

Arcos

Os arcos são criados com a classe Arc2D.Float. Eles são criados de maneira semelhante ao correspondente não-bidimensional, mas há um recurso extra: você pode definir como o arco é fechado.

Arc2D.Float recebe sete argumentos. Os quatro primeiros se aplicam à elipse da qual o arco faz parte: coordenada x, coordenada y, largura e altura. Os três últimos argumentos são o grau inicial do arco, o número de graus que ele percorre e um inteiro descrevendo como ele é fechado.

O número de graus percorridos pelo arco é especificado no sentido horário, usando-se números positivos. Isso é o oposto do modo como um arco não-bidimensional é manipulado.

O último argumento utiliza uma de três variáveis de classe: Arc2D.OPEN para um arco não fechado, Arc2D.CHORD para conectar as extremidades do arco com uma linha reta e Arc2D.PIE para conectar o arco com o centro da elipse, como uma fatia de torta. A Figura 9.16 mostra cada um desses estilos.

Figura 9.16

Estilos de fechamento de arco.

Nota

O estilo de fechamento Arc2D.0PEN não se aplica aos arcos preenchidos. Um arco preenchido que possui Arc2D.0PEN como estilo será fechado usando o mesmo estilo que Arc2D.CH0RD.

A instrução a seguir cria um objeto Arc2D.Float:

```
Arc2D.Float = new Arc2D.Float(27,22,42,30,33,90,Arc2D.PIE);
```

Isso cria um arco de uma elipse nas coordenadas 27,22 com 42 pixels de largura e 30 pixels de altura. O arco começa em 33 graus, se estende por 90 graus no sentido horário e será fechado como uma fatia de torta.

Polígonos

Os polígonos são criados no Java2D pela definição de cada movimento de um ponto para outro do polígono. Um polígono pode ser formado por linhas retas, curvas 'quadráticas' e curvas de bezier.

Os movimentos para criar um polígono são definidos como um objeto General Path, que também faz parte do pacote java.awt.geom.

Um objeto General Path pode ser criado sem quaisquer argumentos, como se vê a seguir:

```
GeneralPath polly = new GeneralPath( );
```

O método moveTo( ) de General Path é usado para criar o primeiro ponto no polígono. A instrução a seguir seria usada se você quisesse iniciar polly nas coordenadas 5,0:

```
polly.moveTo(5f, Of);
```

Após a criação do primeiro ponto, o método 1 ineTo( ) é utilizado para criar linhas que terminam em um novo ponto. Esse método recebe dois argumentos: as coordenadas x e y do novo ponto.

As instruções a seguir adicionam três linhas ao objeto polly:

```
polly.lineTo(205f,  Of);
polly.lineTo(205f,  90f);
polly.lineTo(5f,  90f);
```

Os métodos 1 ineTo( ) e moveTo( ) exigem argumentos float para especificar pontos coordenados.

Se você quiser fechar um polígono, o método closePath ( ) é usado sem nenhum argumento, como se vê a seguir:

```
polly.closePath( );
```

Esse método fecha um polígono conectando o ponto atual ao ponto especificado pelo método moveTo( ) mais recente. Você pode fechar um polígono sem esse método, usando um método lineTo( ) que conecta ao ponto original.

Uma vez criado um polígono aberto ou fechado, você pode desenhá-lo como qualquer outra figura, usando os métodos draw( ) e fi11 ( ). O objeto polly é um retângulo com pontos em 5,0; 205,0; 205,90 e 5,90.

Desenhando objetos

Após ter definido os atributos de renderização, como cor e espessura de linha, e ter criado o objeto a ser desenhado, você está pronto para desenhar algo em toda a sua glória bidimensional.

Todos os objetos desenhados usam os mesmos métodos da classe Graphics2D: draw( ) para contornos e fill( ) para objetos preenchidos. Eles recebem um objeto como único argumento.

No Java2D, os strings são desenhados usando-se o método drawStri ng ( ). Ele recebe três argumentos: o objeto Stri ng a ser desenhado e suas coordenadas x,y. Assim como acontece com todas as coordenadas no Java2D, devem ser especificados números em ponto flutuante, em vez de inteiros.

Um exemplo de desenho bidimensional

Anteriormente, no dia de hoje, você criou um mapa da Flórida usando os métodos de desenho que estão disponíveis na classe Graphics. O próximo applet a ser criado é uma versão revisada desse mapa, que usa técnicas de desenho bidimensional.

Como todas as classes Java2D são novidade da versão 2 da linguagem Java, esse applet só poderá ser visto com um navegador da Web que ofereça suporte à Java 2. Quando este livro estava sendo preparado, a ferramenta appletviewer, incluída no JDK 1.2, era a única maneira de vê-lo.

A listagem 9.5 contém o applet Map2D. Trata-se um programa mais longo do que muitos deste livro, pois as operações bidimensionais exigem mais instruções para se fazer um desenho.

DIGITE **Listagem 9.5.** O texto completo de map2d.java.

```
1: import java.awt.*;
2: import java.awt.geom.*;
3:
4: public class Map2D extends java.applet.Applet {
5:     public void paint(Graphics screen) {
6:       Graphics2D screen2D = (Graphics2D)screen;
7:       setBackground(Color.blue);
8:       // Desenha ondas
9:       screen2D.setColor(Color.white);
10:       BasicStroke pen • new BasicStroke(2F,
11:         BasicStroke.CAPBUTT, BasicStroke.JOINROUND);
12:       screen2D.setStroke(pen);
13:     for (int ax = 10; ax < 340; ax += 10)
14:       for (int ay = 30; ay < 340 ; ay +• 10) {
```

Listagem 9.5. Continuação

```
15:            Arc2D.Float wave = new Arc2D.Float(ax, ay,
16:              10, 10, 0, 180, Arc2D.0PEN);
17:            screen2D.draw(wave);
18:          }
19:        // Desenha a Flórida
20:        GradientPaint gp • new GradientPaint(0F,0F,Color.green,
21:          50F,50F,Color.orange.true);
22:        screen2D.setPaint(gp);
23:        GeneralPath fl = new GeneralPath( );
24:        fl.moveTo(10F,12F);
25:        fl.lineTo(234F,15F);
26:        fl.lineTo(253F,25F);
27:        fl.lineTo(261F,71F);
28:        fl.lineTo(344F,209F);
29:        fl.lineTo(336F,278F);
30:        fl.lineTo(295F,310F);
31:        fl.lineTo(259F,274F);
32:        fl.lineTo(205F,188F);
33:        fl.lineTo(211F,171F);
34:        fl.lineTo(195F,174F);
35:        fl.lineTo(191F,118F);
36:        fl.lineTo(120F,56F);
37:        fl.lineTo(94F,68F);
38:        fl.lineTo(81F,49F);
39:        fl.lineTo(12F,37F);
40:        fl.closePath( );
41:        screen2D.fill(fl);
42:        // Desenha elipses
43:        screen2D.setColor(Color.black);
44:        BasicStroke pen2 = new BasicStroke( );
45:        screen2D.setStroke(pen2);
46:        Ellipse2D.Float ei = new Eliipse2D.Float(235,140,15,15);
47:     Ellipse2D.Float    e2 = new Ellipse2D.Float(225,130,15,15);
48:        Ellipse2D.Float e3 = new Ellipse2D.Float(245,130,15,15);
49:        screen2D.fill(el);
50:        screen2D.fill(e2);
51:        screen2D.fill(e3);
52:     }
53: }
```

Para ver o applet, você precisa criar uma página HTML que o contenha, usando a listagem 9.6. Como ele utiliza classes e métodos Java 2, o applet pode ser visto apenas com um navegador que ofereça suporte a essa versão da linguagem. Quando este livro estava no prelo, a ferramenta appletviewer, incluída com o JDK, era a única que executava applets Java 2. A ferramenta appletviewer manipula as tags <APPLET> e ignora as outras tags HTML; portanto, não há motivo para se criar uma página complicada para algo que você vê apenas com essa ferramenta.

DIGITE **Listagem 9.6.** O texto completo de map2d.html.

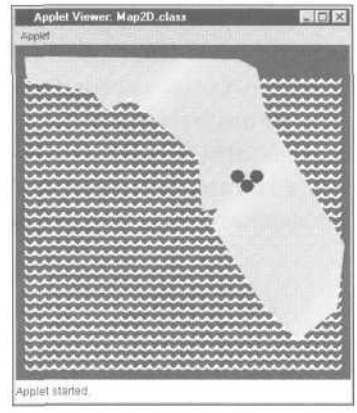
```
1: <applet code="Map2D.class" height=370 width=350>
2: </applet>
```

Algumas observações sobre o applet Map2D:

- A linha 2 importa as classes do pacote java.awt.geom. Essa instrução é exigida porque import java.awt.*; na linha 1 manipula apenas classes e não os pacotes disponíveis em java.awt.
- A linha 6 cria o objeto screen2D que é usado para todas as operações de desenho bidimensionais. Trata-se de uma coerção do objeto Graphics que representa a janela Applet.
- As linhas 10 a 12 criam um objeto BasicStroke que representa uma espessura de linha de 2 pixels e, em seguida, torna esse traço atual com o método setStroke( ) de Graphics2D.
- As linhas 13 a 17 usam dois loops for aninhados para criar ondas a partir de arcos individuais. Essa mesma técnica foi usada para o applet Map, mas em Map2D, existem mais arcos cobrindo a janela Applet.
- As linhas 20 e 21 criam um padrão de preenchimento de gradiente, variando da cor verde, nas coordenadas 0,0, até a cor laranja, nas coordenadas 50,50. O último argumento do construtor, true, faz o padrão de preenchimento ser repetido tantas vezes quanto for necessário para preencher um objeto.
- A linha 22 define o padrão de preenchimento de gradiente atual, usando o método setPaint( ) e o objeto gp que acabou de ser criado.
- As linhas 23 a 41 criam o polígono que possui a forma do estado da Flórida e o desenham. Esse polígono será preenchido com faixas que variam de verde a laranja, devido ao padrão de preenchimento que está selecionado.
- A linha 43 define a cor atual como preto. Isso substitui o padrão de preenchimento de gradiente para a próxima operação de desenho, pois as cores também são padrões de preenchimento.
- A linha 44 cria um novo objeto BasicStroke( ) sem nenhum argumento, o que, por padrão, resulta em uma espessura de linha de 1 pixel de largura.
- A linha 45 define a espessura de linha atual de acordo com o novo objeto BasicStroke, pen2.
- As linhas 46 a 51 criam três elipses nas coordenadas 235,140, 225,130 e 245,130. Cada uma delas tem 15 pixels de largura e 15 pixels de altura, o que as transforma em círculos.

A Figura 9.17 mostra a saída do applet Map2D no appletviewer.

Fiaura 9.17
O *applet Map2D.*



Resumo

Agora, você possui algumas ferramentas para melhorar a aparência de um applet. Você pode desenhar com linhas, retângulos, elipses, polígonos, fontes, cores e padrões em uma janela Applet, usando classes bidimensionais e não-bidimensionais.

As operações de desenho não-bidimensionais exigem o uso de métodos da classe Graphics, com argumentos que descrevem o objeto que está sendo desenhado.

O Java2D usa os mesmos dois métodos para cada operação de desenho — draw( ) e fill ( ). Os diferentes objetos são criados usando-se classes do pacote java.awt.geom e eles são usados como argumentos para os métodos de desenho de Graphics2D.

Posteriormente no livro, você aprenderá a desenhar em outros componentes de um programa Java, como fez na janela Applet. Isso permitirá que você também use as técnicas de hoje em um aplicativo Java.

Você terá mais oportunidades de impressionar o tio Walter amanhã, quando as lições de arte incluírem animação e exibição de arquivos de imagem.

Perguntas e respostas

P **Quero desenhar uma linha de texto com uma palavra em negrito no meio. Sei que preciso de dois objetos Font — um para a fonte normal e outro para a fonte em negrito — e que precisarei redefinir a fonte atual nesse ínterim. O problema é que drawString( ) exige uma posição x e uma posição y para o início de cada string e não consigo encontrar nada que faça referência ao "ponto atual". Como posso saber onde devo iniciar a palavra em negrito?**

Os recursos de exibição de texto da linguagem Java são bastante primitivos. A linguagem Java não possui o conceito de ponto atual; portanto, você mesmo tem de descobrir onde ocorreu o final de um string para que possa iniciar o próximo. Os métodos stringWidth ( ) podem ajudá-lo nesse problema, tanto para descobrir a largura do string que você acabou de desenhar como para acrescentar espaço depois dele.

SEMANA 2

Dia 10

Inserindo imagens, animação e som

A primeira revelação da linguagem Java para muitas pessoas foi a visão de texto animado e imagens que se movimentam em uma página da Web. Esses tipos de animação são simples, exigindo apenas alguns métodos para implementar em Java, mas esses métodos são a base de qualquer applet que exija atualizações dinâmicas na tela. Começar com uma animação simples é um bom modo de se chegar a applets mais complicados.

Na linguagem Java, a animação é realizada usando-se partes inter-relacionadas do AWT (Abstract Windowing Toolkit). Hoje, você aprenderá como as várias partes da linguagem Java trabalham em conjunto para que se possa criar figuras que se movem e applets atualizados dinamicamente.

A criação de animação é divertida e fácil na linguagem Java, mas há um limite para o que você pode fazer com os métodos internos da linguagem para linhas, fontes e cores. Para obter uma animação interessante, você precisa fornecer suas próprias imagens para cada um de seus quadros — e se tiver sons também será interessante.

Hoje, você irá explorar os seguintes tópicos:

- Como funcionam as animações na linguagem Java — os métodos paint( ) e repaint( ), início e interrupção de applets dinâmicos e como usar e anular esses métodos em seus próprios applets

- Threads — o que são eles e como podem melhorar o comportamento de seus applets com outros applets e com o sistema em geral
- Como reduzir o tremido da animação — um problema comum na animação em Java
- Uso de imagens de bitmap, como arquivos GIF ou JPEG — sua obtenção do servidor, carregamento na linguagem Java, exibição em seu applet e uso em animação
- Uso de sons — sua obtenção e reprodução nos momentos apropriados

Criando animação em Java

A animação na linguagem Java é um processo relativamente simples que exige as seguintes etapas:

- Desenhar algo usando arquivos de texto, objetos ou imagens.
- Informar ao sistema de janelas para que apresente o que você desenhou.

Tais etapas são repetidas com diferentes itens sendo desenhados, para dar a impressão de movimento. Você pode variar o período de tempo entre diferentes quadros na seqüência animada ou deixar que a linguagem Java desenhe o mais rápido que puder.

Pintando e repintando

Como você aprendeu, o método pai nt ( ) é chamado automaticamente, quando a área de exibição de um applet precisa ser redesenhada. Esse método é chamado quando um applet começa, pois a janela está em branco e precisa ser desenhada pela primeira vez. Ele também pode ser chamado quando a janela do applet aparece, após ser ocultada pela janela de outro programa.

Você pode solicitar ao sistema de janelas da linguagem Java para que repinte a janela usando o método repai nt ( ).

A linguagem polida é usada aqui por um motivo — repaint( ), na verdade, é um pedido e não um comando. O sistema de janelas Java recebe esse pedido e o processa assim que for possível, mas se forem empilhados mais pedidos de repai nt( ) do que a linguagem Java pode manipular, alguns poderão ser pulado. Na maior parte dos casos, o atraso entre a chamada a repai nt( ) e a reapresentação real da janela é desprezível.

Para mudar a aparência do que é exibido em uma área, como uma janela Applet, você desenha o que quer apresentar, chama repai nt ( ), desenha mais alguma coisa, chama repai nt ( ) novamente e assim por diante.

Todas essas ações não ocorrem no método paint( ), pois ele é responsável apenas por desenhar um único quadro da animação — o quadro mais recente. O trabalho real ocorre em outro lugar no applet.

Nesse outro lugar, que poderia ser seu próprio método, você cria objetos, desenha-os, executa quaisquer outras tarefas necessárias e conclui chamando repaint( ).

Embora você mesmo possa chamar o método paint( ), deve fazer todos os pedidos para desenhar a área de exibição usando chamadas arepaint( ). O método repaint( ) é mais fácil de usar — ele não exige um objeto Graphics como argumento (ao contrário de paint( )) e cuida de todo comportamento necessário para atualizar a área de exibição. Você verá isso posteriormente ainda hoje, quando chamar repaint( ) para criar uma seqüência animada.

Iniciando e interrompendo a execução de um applet

Como você se lembra do Dia 8, os métodos start( ) e stop( ) são chamados quando um applet inicia e interrompe sua execução.

Esses métodos estão vazios quando são herdados de java. applet. Applet; portanto, você precisa anulá-los para fazer qualquer coisa no início ou na conclusão de seu programa. Você não usou start( )oustop( ) ontem, porque os applets precisavam utilizar paint( ) apenas uma vez.

No caso da animação e de outros applets Java que estão processando e são executados com o passar do tempo, start ( ) e stop ( ) são necessários para disparar o início da execução do applet e para interrompê-la quando a página que o contém for abandonada.

Controlando a animação com threads

A animação representa uma utilização ideal para threads, o significado da linguagem Java para a manipulação de mais de uma tarefa de programação simultânea.

Um *thread* é uma parte de um programa definida para execução independente, enquanto o restante do programa faz outra coisa. Isso também é chamado de *multitarefa*, pois o programa pode manipular mais de uma tarefa simultaneamente.

fácil para que o ambiente de runtime do applet manipule o programa, pois todo o trabalho intensivo é isolado em seu próprio thread.

Escrevendo applets com threads

Para usar um thread em um applet, você pode fazer cinco modificações em seu arquivo de classe:

- Mudar a declaração de classe, inserindo o texto implements Runnable.
- Criar um objeto Thread para conter o thread.
- Anular o método start( ) do applet para criar um thread e começar a executá-lo.
- Anular o método stop( ) para definir o thread que está em execução como null.
- Criar um método run( ) que contenha as instruções que fazem o applet ser executado continuamente.

A palavra-chave implements é semelhante à palavra-chave extends, pois ela modifica a cl ass que é declarada na mesma linha. A seguir está um exemplo de uma classe que utiliza extends e implements:

```
public class DancingBaby extends java.applet.Applet
    implements Runnable {
    // ...
}
```

Embora a declaração de classe tenha sido dividida em duas linhas, tudo o que aparece desde a palavra-chave public até a chave "{" define a classe.

Runnabl e é um tipo especial de classe chamado interface. Como você pode se lembrar do Dia 2, uma *interface* é um meio de uma classe herdar métodos que, de outro modo, não seriam herdados de suas superclasses.

Esses métodos podem ser implementados por qualquer classe que precise do comportamento. Nesse exemplo, a interface Runnabl e é implementada pelas classes que funcionarão como um thread. Runnabl e fornece uma definição para o método run( ), que é chamado para iniciar um thread.

A classe Thread faz parte do pacote java.lang padrão; portanto, ela não precisa se tornar disponível através de uma instrução import. Iniciar a criação de um objeto Thread é tão simples quanto fornecer um nome, como na instrução a seguir:

```
Thread runner;
```

Esse objeto pode ser criado no método start( ) do applet. A variável runner terá o valor null até que o objeto tenha sido realmente criado.

O lugar ideal para criá-lo é no método start( ) do applet. O método a seguir verifica se o thread foi criado. Se não foi, ele o cria:

```
public void start( ) {
    if (runner == null) {
        runner = new Thread(this);
        runner.start( );
    }
}
```

A palavra-chave this usada no construtor Thread( ) é um modo de se fazer referência ao objeto em que o método está sendo executado — o próprio applet. Usando-se this, o applet é identificado como a classe que fornece o comportamento para executar o thread.

Para se executar um thread, seu método start( ) é chamado, como na seguinte instrução do exemplo anterior:

```
runner.start( );
```

Chamar o método start ( ) de um thread faz com que outro método seja chamado — o método run( ) da classe que está manipulando o thread.

Nesse exemplo, o applet implementa a interface Runnabl e e foi vinculado ao objeto runner através da palavra-chave this. Um método run( ) deve ser incluído no applet. A seguir está um exemplo:

```
public void run( ) {
    // o que seu applet realmente faz
)
```

O método run( ) é o núcleo do applet com thread. Ele deve ser usado para impulsionar a seqüência animada, definindo tudo o que é necessário para os desenhos e para mudar as coisas entre cada quadro.

Depois que o método run( ) tiver sido preenchido com todo comportamento de que o thread necessita, a última etapa na confecção de um applet com thread é usar seu método stop( ) para interromper esse thread.

O modo de interromper um thread é definir seu objeto como null. Isso não interromperá o thread, mas você pode projetar um método run ( ) de tal modo que ele continue executando apenas enquanto seu objeto Thread não for igual a null.

Cuidado

Há um método stop ( ) que poderia ser chamado em objetos Thread para interrompê-los, mas a JavaSoft o desaprovou a partir da versão 2 da linguagem Java. O uso do método Stop( ) de um thread cria instabilidades no ambiente de runtime do programa e pode introduzir em seu funcionamento, erros que são difíceis de detectar.

Os programadores são fortemente desestimulados a usar stop ( ) para interromper um thread na linguagem Java, mesmo em programas Java 1.02 e 1.1. A alternativa utilizada hoje é recomendada em notas de lançamento publicadas com a linguagem Java 2 e deve ser conveniente para toda programação com threads.

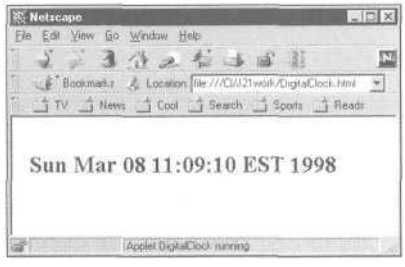
Através da inserção de implements Runnable, da criação de um objeto Thread associado ao applet e do uso dos métodos start( ),stop( ) e run( ), um applet se torna um programa com thread.

Reunindo tudo

A programação com threads deve se tornar mais clara quando você a vir realmente em ação. A listagem 10.1 contém um applet animado simples que apresenta a data e a hora, com atualizações constantes. Isso cria um relógio digital, como se vê na Figura 10.1.

Figura 10.1

O applet *DígitalClock no Netscape Navigator.*



Esse applet usa os métodos pai nt( ),start( )estop( ). Ele também usa threads.

DIGITE **Listagem 10.1.** O texto completo de digitalclock.java.

```
1: import java.awt.Graphics;
2: import java.awt.Font;
3: import java.útil.Date;
4:
5: public class DigitalClock extends java.applet.Applet
6:     implements Runnable {
7:
8:     Font theFont = new Font("TimesRoman",Font.BOLD,24);
9:     Date theDate;
10:    Thread runner;
11:
12:    public void start( ) {
13:     if (runner == null) {
14:       runner = new Thread(this);
15:       runner.start( );
16:     }
17: }
18:
19:   public void stop( ) {
20:     if (runner != null) {
21:       runner • null;
22:     }
23:   }
```

Listagem 10.1. Continuação

```
24:
25:    public void run( ) {
26:      Thread thisThread = Thread.currentThread( );
27:      while (runner == thisThread) {
28:        repaint( );
29:        try {
30:          Thread.sleep(lOOO);
31:        } catch (InterruptedException e) { }
32:      }
33:    }
34:
35:    public void paint(Graphics screen) {
36:      theDate • new Date( );
37:      screen.setFont(theFont);
38:    screen.drawString(""    + theDate.toString( ), 10, 50);
39:    }
40: }
```

Para testar o applet, coloque-o em uma página da Web, em uma janela Applet com width=380 e height=100.

Nota

Esse applet usa a classe 0ate( ) para obter a data e a hora atuais, o que o torna compatível com a linguagem Java 1.02. Para as versões posteriores da linguagem, a classe Cal endar deve ser usada, pois ela oferece melhor suporte aos sistemas de calendário internacionais. Existe uma versão compatível com a linguagem Java 2 do applet DigitalClock, chamado Di gi tal Cl ockl2. java, no site da Web deste livro, no endereço http://www.prefect.com/java21.

A animação é um bom exemplo do tipo de tarefa que precisa de seu próprio thread. Considere o loop whi 1 e( ) infinito do applet DigitalClock. Se você não usasse threads, o loop while( ) seria executado no thread de sistema padrão Java, que também é responsável por pintar a tela, tratar de entrada de usuário, como no caso de cliques de mouse, e manter tudo atualizado internamente. Infelizmente, se você executar esse loop whi 1 e( ) no thread de sistema principal, ele monopolizará os recursos da linguagem Java e impedirá que tudo o mais — incluindo a pintura — aconteça. Você nunca veria nada na tela, pois a linguagem Java ficaria esperando que o loop while( ) terminasse, antes de fazer qualquer outra coisa.

Nesta seção, você verá esse applet da perspectiva das partes relacionadas à animação e depois tratará das partes que gerenciam threads.

As linhas 8 e 9 definem duas variáveis de instância básicas: theFont e theDate, que contêm objetos representando a fonte atual e a data atual, respectivamente. Você aprenderá mais sobre isso posteriormente.

Os métodos start( ) e stop( ) aqui presentes iniciam e interrompem um thread; o grosso do trabalho do applet fica no método run ( ) (linhas 25 a 33).

Em run( ) é que a animação realmente ocorre. Observe o loop while dentro desse método (começando com a instrução da linha 27); a expressão runner == thisThread retornará o valor true até que o objeto runner seja definido como null (o que ocorre no método stop( ) do applet). Um único quadro de animação é criado dentro desse loop whi 1 e.

A primeira coisa que acontece no loop é que a instrução repaint( ) é chamada na linha 28, para repintar o applet. As linhas 29 a 31, complicadas como podem parecer, não realizam nada, exceto fazer uma pausa por 1.000 milissegundos (1 segundo) antes que o loop seja repetido.

O método sleep( ) da classe Thread é o que faz o applet dar uma pausa. Sem um método sleep( ) específico, o applet seria executado o mais rápido possível. O método sleep( ) controla exatamente a rapidez com que a animação ocorre. As instruções try e catch em torno dele permitem que a linguagem Java gerencie os erros, se eles ocorrerem. Essas instruções serão descritas no Dia 16.

No método pai nt ( ), nas linhas 35 a 39, uma nova instância da classe Date é criada para conter a data e a hora atuais — observe que ele foi especificamente importado na linha 3. Esse novo objeto Date é atribuído à variável de instância theDate.

Na linha 37, a fonte atual é definida usando-se o valor da variável theFont e a data em si é exibida na tela — observe que você precisa chamar o método toString( ) de Date para apresentar a data e a hora como um string. Sempre que paint( ) é chamado, um novo objeto theDate é criado, contendo a data e a hora atuais.

Examine as linhas desse applet que criam e gerenciam threads. Primeiramente, examine a definição de classe em si, nas linhas 5 e 6. Observe que a definição de classe implementa a interface Runnabl e. Todas as classes que você criar, que utilizem threads, deverão incluir Runnabl e.

A linha 10 define uma terceira variável de instância para essa classe, chamada runner, de tipo Thread, que conterá o objeto thread desse applet.

As linhas 12 a 23 definem os métodos de texto padronizado start( ) e stop( ), que nada fazem a não ser criar e destruir threads. Essas definições de método serão semelhantes de uma classe para outra, pois apenas estabelecem a infra-estrutura para os threads utilizados pelo programa.

Finalmente, o grosso do trabalho de seu applet fica dentro do método run( ), nas linhas 25 a 33.

Reduzindo o tremido da animação

Quando o applet DigitalClock é executado, você vê um certo tremido no texto, quando ele é apresentado. A intensidade do tremido depende da qualidade do

ambiente de runtime da linguagem Java em que o programa é executado, assim como da velocidade do processador. No entanto, provavelmente ele será irritante, mesmo em um PC rápido e uma máquina virtual Java bem implementada.

O tremido é um dos efeitos colaterais do modo como as imagens são atualizadas em um programa Java e representa um dos problemas que você enfrentará quando criar animação.

O tremido e como evitá-lo

O tremido é causado pelo modo como a linguagem Java repinta cada quadro de um applet. No início da lição de hoje, você aprendeu que, quando o método repaint( ) é chamado, ele chama o método pai nt( ).

Na verdade, há um intermediário envolvido. Quando repaint( ) é chamado, ele chama o método update( ), que limpa a tela de qualquer conteúdo existente, preenchendo-a com a cor de fundo da janela Applet. O método update( ) chama, então, paint( ).

O processo de limpeza da tela em update( ) é o principal culpado no problema do tremido. Como a tela é limpa entre os quadros, as partes da tela que não mudam entre eles alternam rapidamente entre serem pintadas e serem limpas — em outras palavras, elas tremem.

Existem duas maneiras principais de evitar o tremido em seus applets Java:

- Anular o método update( ) para que ele não limpe a tela ou limpe apenas as partes dela que você alterou.
- Anular os métodos update( ) e paint( ) e usar bufferização dupla.

O modo mais fácil de reduzir o tremido é anular o método update ( ) para que ele não limpe a tela. A melhor maneira para se cuidar disso é usar dupla bufferização.

Como anular imdateí)

O método update ( ) padrão de qualquer applet assume a seguinte forma:

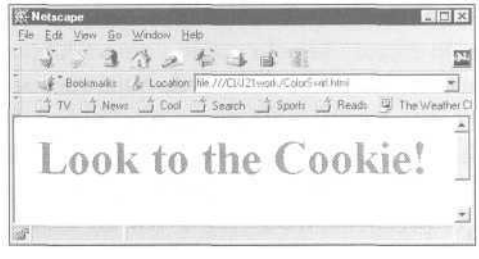
```
public void update(Graphics g) {
    g.setColor(getBackground( ));
    g.fillRectfO, 0, size( ).width, size( ).height);
    g.setColor(getForeground( ));
    paint(g);

}
```

O método update( ) limpa a tela preenchendo a janela do applet com a cor de fundo, define a cor de volta para o normal e, em seguida, chama paint( ). Quando você anula update( ) com sua própria versão do método, precisa certificar-se de que sua versão faz algo semelhante. Nas próximas duas seções, você trabalhará em alguns exemplos de anulação de update ( ) para reduzir o tremido.

Uma solução: não limpar a tela

A primeira solução para reduzir o tremido é não limpar a tela. Essa solução só funciona para alguns applets, é claro. Por exemplo, o applet ColorSwirl apresenta um único string (Look to the cookie!), mas o string é apresentado em cores diferentes, que se alternam gradualmente de forma dinâmica. Esse applet treme terrivelmente ao ser executado. A listagem 10.2 mostra o código-fonte inicial desse applet e a Figura 10.2 mostra o resultado.

Figura 10.2
Saída do applet *ColorSwirl usando* o Netscape Navigator.



DIGITE

Listagem 10.2. O texto completo de colorswirl .java.

```
1: import java.awt.Graphics;
2: import java.awt.Color;
3: import java.awt.Font;
4:
5: public class ColorSwirl extends java.applet.Applet
6:     implements Runnable {
7:
8:     Font f = new Font("TimesRoman", Font.BOLD, 48);
9:     Color colors[ ] = new Color[50];
10:    Thread runner;
11:
12:    public void start( ) {
13:     if (runner == null) {
14:       runner = new Thread(this);
15:       runner.start( );
16:     }
17: }
18:
19:    public void stop( ) {
20:     runner = null;
21: }
22:
23:    public void run( ) {
24:     // inicializa o array de cores
25:     float c = 0;
26:     for (int i = 0; i < colors.length; i++) {
27:       colors[i] =
28:        Color.getHSBColor(c, (float)1.0,(float)1.0);
29:       c += .02;
```

Listagem 10.2. Continuação

```
30:       }
31:
32:       // circula pelas cores
33:       int i = 0;
34:       Thread thisThread = Thread.currentThread( );
35:       while (runner == thisThread) {
36:         setForeground(colors[i]);
37:         repaint( );
38:
39:         i++;
40:         try {
41:           Thread.sleep(200);
42:         } catch (InterruptedException e) { }
43:         if (i == coiors.length ) i = 0;
44:       }
45:     }
46:
47:     public void paint(Graphics screen) {
48:       screen.setFont(f);
49:       screen.drawString("Look to the Cookie!", 15, 50);
50:     }
51: }
```

Para testar esse applet, coloque-o em uma página da Web com uma tag <APPLET> com atributos hei ght=150 width=450. Três detalhes a respeito desse applet podem parecer estranhos para você:

- A linha 9 define uma variável de instância colors, que é um array de 50 elementos. Quando o applet começa sua execução, a primeira coisa que você faz no método run ( ) (nas linhas 25 a 30) é preencher esse array com objetos Color. Criando todas as cores antecipadamente, você pode simplesmente desenhar texto nessa cor, um por vez; é mais fácil computar previamente todas as cores de forma simultânea. (E, na verdade, esse loop for poderia fazer mais sentido em um método init( ), pois isso precisa acontecer apenas uma vez.) A escolha de 50 cores é arbitrária — o programa pode circular por 20 ou 250 cores com a mesma facilidade.
- Para criar os diferentes objetos coloridos, um método na classe Col or, chamado getHSBColor( ) é usado, em vez de se apenas usar new com diversos valores sRGB. O método de classe getHSBColor( ) cria um objeto Col or baseado nos valores de matiz, saturação e brilho, em vez das cores vermelho, verde e azul padrão. Incrementando o valor do matiz, enquanto mantém a saturação e o brilho constantes, você pode criar uma gama de cores sem ter de gerar o valor sRGB de cada uma. Esse é apenas um modo rápido e fácil de criar o array de cores.

- Para criar a animação, o applet circula pelo array de cores, definindo a cor de primeiro plano de cada objeto Color por sua vez, e chamando repaint( ). Quando chega ao final do array, ele começa novamente (linha 45), para que o processo se repita novamente *ad infinitum.*

Agora que você entende o que o applet faz, é hora de corrigir o tremido. O tremido está ocorrendo porque, sempre que o applet é pintado, há um momento em que a tela é limpa. Em vez de o texto circular normalmente da cor vermelha, passando por um rosa pink, para púrpura, ele está passando de vermelho para cinza, de rosa pink para cinza, de púrpura para cinza etc.

Como a limpeza da tela é o que está causando o problema, a solução é fácil: anule update( ) e remova a parte em que a tela é limpa. Ela nem mesmo precisa ser limpa, pois nada está mudando, exceto a cor do texto. Com o comportamento de limpeza de tela removido deupdate( ),esse método precisa apenas chamar paint( ). Aqui está o método update( ) no applet ColorSwirl revisado:

```
public void update(Graphics screen) {
        paint(screen);
}
```

A inserção dessas três linhas pára o tremido.

Você pode encontrar a primeira versão de ColorSwirl . java com esse nome no site da Web deste livro, no endereço http://www. prefect.com/java21, e *a* versão aprimorada, no mesmo lugar, como BetterSwirl .java.

Ainda hoje, você aprenderá outro método para reduzir o tremido — uma técnica chamada *bufferização dupla.*

Recuoerando e usando imaaens

O tratamento de imagens básico na linguagem Java é conduzido através da classe Image, que faz parte do pacote java.awt. Ao trabalhar com um applet, você pode usar métodos das classes Applet e Graphics para carregar e exibir imagens.

Obtendo imaaens

Para exibir uma imagem em seu applet, primeiramente você deve carregá-la através da World Wide Web em seu programa Java. As imagens são armazenadas como arquivos separados dos arquivos de classe Java; portanto, você precisa informar à linguagem Java onde encontrá-los.

Quando você usa a classe Image, a imagem deve estar no formato.GIF ou .JPG.

Um endereço da Web é representado na linguagem Java como um objeto URL, um acrônimo que significa Uniform Resource Locator. A classe URL faz parte do pacote java.net; portanto, assim como Image, ela exige uma instrução import para ser usada em um programa Java.

O objeto URL é criado pelo envio de um endereço de página da Web para o método construtor URL. A seguir um exemplo:

```
URL u = new URL("http://www.prefect.com/java21/images/book.gif");
```

Quando você tem um objeto URL, pode usá-lo para criar um objeto Image que representa o arquivo gráfico.

A classe Appiet fornece um método chamado getImage( ) para carregar uma imagem em um objeto Image. Existem duas maneiras de usá-lo:

- O método getImage( ) com um único argumento (um objeto do tipo URL) recupera a imagem que está nesse URL.
- O método getImage( ) com dois argumentos: o URL* de base (também um objeto URL) e um string representando o caminho relativo ou o nome de arquivo da imagem real.

Embora o primeiro modo possa parecer mais fácil, o segundo é mais flexível. Se você usar um endereço da Web específico em seu appiet, deverá mudá-lo e recompilar o programa, caso seu site da Web seja movido.

A classe Appl et possui dois métodos que podem ser usados para criar um URL de base sem utilizar um endereço específico no programa:

- O método getDocumentBase( ) retorna um objeto URL que representa a pasta que contém a página da Web que está apresentando o appiet. Por exemplo, se a página estiver localizada no endereço http://www. prefect.com/java21/, getDocumentBase( ) retornará um URL que aponta para esse caminho.
- O método getCodeBase( ) retorna um objeto URL que representa a pasta em que o arquivo de classe principal do appiet está localizado.

Caminhos de arauivo relativos

O caminho relativo que você utiliza como segundo argumento de getlmage( ) mudará de acordo com o que for usado no primeiro argumento.

Por exemplo, considere uma página da Web no endereço http://www. prefect.com/java21/index.html que contém um arquivo de imagem chamado http: //www. pref ect. com/j ava21/book. g i f. Se você quiser carregar essa imagem em um appiet, a seguinte instrução pode ser usada:

```
Image img = new URL(getDocumentBase( ), "book.gif");
```

Como outro exemplo, se o arquivo book.gif fosse movido para http://www.prefect.com/java21/images/book.gif, a seguinte instrução poderia ser usada:

```
Image img = new URL(getDocumentBase( ), "images/book.gif");
```

A escolha entre getDocumentBase( ) ou getCodeBase( ) depende de suas imagens estarem armazenadas em subpastas de seu applet Java ou em subpastas da página da Web do applet.

Nota

O uso de getDocumentBase( ) ou getCodeBase( ) também torna possível que a imagem seja carregada quando você testá-la em seu próprio computador. Você não precisa armazená-la em um site da World Wide Web, antes de ver se ela funciona.

Usando um desses métodos, você torna possível mover seu applet junto com sua página da Web e não fazer nenhuma alteração no programa.

Nota

Se você usa um repositório de arquivos Java para apresentar seu applet, pode incluir arquivos de imagem e outros arquivos de dados do repositório de arquivos. Esses arquivos serão extraídos do repositório de arquivos automaticamente, com todos os arquivos de classe de .JAR.

Desenhando imaaens

Após ter carregado uma imagem em um objeto Image, você pode exibi-la em um applet usando o método drawlmage( ) da classe Graphics.

Para exibir uma imagem em seu tamanho real, você chama o método drawlmage( ) com quatro argumentos:

- O objeto Image a ser apresentado.
- A coordenada x.
- A coordenada y.
- A palavra-chave thi s.

Se um arquivo gráfico estiver armazenado no objeto img, o seguinte método paint ( ) pode ser utilizado para exibi-lo:

```
public void paint(Graphics screen) {
    screen.drawlmage(img, 10, 10, this);
}
```

As coordenadas x,y usadas com drawlmage( ) são comparáveis à utilização das coordenadas x,y para exibir um retângulo. O ponto representa o canto superior esquerdo da imagem.

Você pode apresentar uma imagem em um tamanho diferente, usando seis argumentos extras:

- O objeto Image a ser exibido.
- A coordenada x.
- A coordenada y.
- Width.
- Height.
- A palavra-chave this.

Os argumentos width e height descrevem a largura e a altura, em pixels, que a imagem deve ocupar quando for exibida. Se não corresponderem ao tamanho real, a imagem mudará de escala para caber na largura e na altura especificadas. Isso não altera a imagem, de modo que você pode usar várias chamadas de drawImage( ) para apresentar um objeto Image em vários tamanhos diferentes.

Dois métodos da classe Image são úteis ao se apresentar uma imagem com um tamanho diferente do real. O método getHeight( ) retorna a altura da imagem como um inteiro e o método getWidth( ) retorna a largura.

Uma nota sobre observadores de imagem

O último argumento do método drawImage( ) é a palavra-chave this. Como você pode se lembrar de dias anteriores, this pode ser usado dentro de um objeto para fazer referência a ele mesmo.

A palavra-chave this é utilizada em drawImage( ) para identificar se o applet pode controlar uma imagem quando ela é carregada da World Wide Web. O carregamento de imagens é controlado através da interface ImageObserver. As classes que implementam essa interface, como a Applet, podem controlar o progresso de uma imagem. Isso seria útil para criar um programa que exibisse uma mensagem como "Loading images..." (Carregando imagens...), enquanto seus arquivos gráficos estivessem sendo carregados.

O suporte existente a ImageObserver deve ser suficiente para utilizações simples de imagens em applets; portanto, a palavra-chave this é usada como um argumento de drawImage( ).

Fazendo as imagens funcionarem

Antes de entrarmos na animação de imagens, um applet simples mostrará um exemplo de carregamento de uma imagem a partir de um URL e sua apresentação em dois tamanhos diferentes. O applet Fillmore, da listagem 10.3, exibe uma imagem do presidente dos EUA Millard Fillmore, o último dos Whigs.

DIGITE

Listagem 10.3. O texto completo de fillmore.java.

```
1: import java.awt.Graphics;
2: import java.awt.Image;
3:
4: public class Fillmore extends java.applet.Applet {
5:     Image whig;
6:
7:     public void init( ) {
8:       whig = getImage(getCodeBase( ),
9:         "images/fillmore.jpg");
10:    }
11:
12:    public void paint(Graphics screen) {
13:      int iWidth - whig.getWidth(this);
14:      int iHeight = whig.getHeight(this);
15:      int xPos = 10;
16:      // 25%
17:      screen.drawImage(whig, xPos, 10,
18:        iWidth / 4, iHeight / 4, this);
19:      // 100%
20:      xPos += (iWidth / 4) + 10;
21:      screen.drawImage(whig, xPos, 10, this);
22:    }
23: }
```

Para se aprontar para o teste do applet Fillmore, faça o seguinte:

- Crie uma nova subpasta na pasta \J21work, chamada images.
- Copie o arquivo f i 11 more. j pg para essa pasta, a partir do site da Web deste livro, no endereço http://www.prefect.com/java21. Como alternativa, você pode usar qualquer arquivo .JPG que já esteja em seu sistema.
- Crie uma página da Web que carregue o applet, com atributos height=400 width=420 para a tag <APPLET>. Se você usar um de seus arquivos .JPG, talvez precise ajustar os atributos height e width para dar espaço suficiente para a imagem na janela Applet.

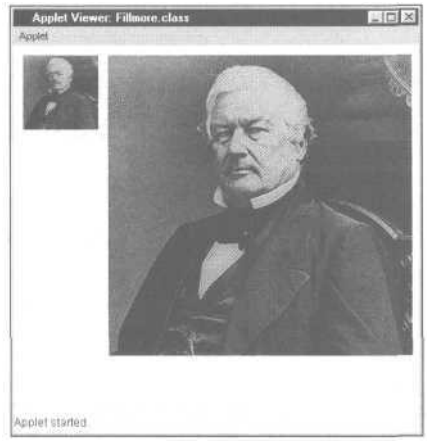
A Figura 10.3 mostra a saída do applet, que apresenta o arquivo gráfico fillmore.jpg em dois tamanhos: 25% e 100%.

A linha 5 do applet associa a variável whig à classe Image. Você não precisa usar a instrução new para criar um objeto aqui, pois recebe um objeto Image do método get Image ( ), nas linhas 8 e 9.

As linhas 13 e 14usamgetWidth( )egetHeight( ), dois métodos da classe Image, e armazenam seus valores em variáveis inteiras. Elas são necessárias para **criar uma** versão em menor escala da imagem, nas linhas 17 e 18.

A linha 15 cria a variável xPos, que armazena a coordenada x para usar as duas versões do presidente Fillmore. Na linha 20, essa variável é aumentada para que a foto grande apareça a 10 pixels à direita da versão menor.

Figura 10,3
O opplet *Fillmore.*



Criando animação com o uso de imaaens

Criar animação com imagens é o mesmo que animar fontes, cores e outros objetos. Você usa os mesmos métodos e procedimentos para pintar, repintar e reduzir problemas de tremido. A única diferença é que você tem uma pilha de imagens para virar, em vez de um conjunto de operações de desenho.

A melhor maneira de mostrar como animar imagens é através de um exemplo. O próximo projeto é o mais longo que você já empreendeu, mas ele é abordado em detalhes. O applet Neko é uma boa demonstração da programação de applets com threads, tratamentos de imagens e animação.

Um exemplo: Neko

As imagens com que você trabalhará no applet Neko serão familiares para muitos usuários de Macintosh e para algumas pessoas de outras plataformas também. Neko, cujo nome provém da palavra japonesa correspondente a gato em português, é um programa para Macintosh escrito por Kenji Gotoh, que apresenta um gatinho animado. Neko caça o cursor do mouse na tela, dorme e se ocupa de outro comportamento no estilo Tamagotchi.

Para esse exemplo, você criará animação com base na imagem gráfica original do Neko. O Neko baseado em Java correrá a partir do lado esquerdo da janela Applet, parará no meio, bocejará, coçará sua orelha, dormirá um pouco e depois correrá para a direita.

Etapa 1: Reúna suas imagens

Antes de começar a escrever código Java para criar uma animação, você deve ter todas as imagens que formam a animação em si. Para essa versão de Neko, existem nove delas (o original tem 36), como se vê na Figura 10.4.

Figura 10.4
As imagens do applet *Neko.*

Para estar pronto para esse projeto, copie os nove arquivos de imagem a seguir, do site da Web deste livro, no endereço http: //www. p ref ect. com/java21, para a pasta \J21work\images, que você criou anteriormente: Awakel.gif, Rightl.gif, Right2.gif, Scratchl.gif, Scratch2.gif, Sleepl.gif, Sleep2.gif, Stop.gif e Yawn.gif.

Etapa 2: Organize e carregue as imagens em seu applet

A idéia básica no applet Neko é pegar a série de imagens e apresentá-las uma por vez para dar a impressão de movimento.

Um modo de fazer isso na linguagem Java é armazenar as imagens em um array de objetos Image e usar outro objeto Image, chamado currentlmg, para controlar a imagem atual a ser exibida.

No caso do applet Neko, as imagens serão armazenadas em um array chamado nekoPi cs e um objeto Image, chamado currentlmg, conterá a imagem atual. As instruções a seguir declaram esses objetos:

```
Image nekoPics[ ] = new Image[9];
Image currentImg;
```

O array de imagens possui nove slots (0 a 8), que contêm as imagens.

Como a animação do Neko desenha imagens de gato em diferentes posições na tela, as coordenadas x,y atuais também serão controladas nas variáveis inteiras x e y. O Neko se moverá totalmente da esquerda para a direita; portanto, a variável inteira y terá sempre o mesmo valor: 50. As instruções a seguir criam essas variáveis:

```
int x;
int y = 50;
```

O método i ni t( ) do applet será usado para carregar as imagens no array nekoPics. Você pode fazer chamadas separadas de getImage( ) para cada um dos nove nomes de arquivo, mas um modo menos redundante de se fazer isso é criar um array String que contenha todos os nove nomes de arquivo. Esse array será usado em um loop for para fornecer os nomes de arquivo para o método getlmage( ). A seguir há o resultado:

```
public void init( ) {
   String nekoSrc[ ] = { "rightl.gif", "right2.gif",
```

```
    "stop.gif",  "yawn.gif",  "scratchl.gif",
    "scratch2.gif",  "sleepl.gif",  "sleep2.gif",
    "awake.gif"  };

  for (int i=0; i < nekoPics.length; i++) {
    nekopics[i]  = getImage(getCodeBase( ),
      "images/" + nekoSrc[i]);
  }
 }
```

Como as imagens foram armazenadas na subpasta images, isso faz parte da referência a arquivo enviada para o método getlmage( ).

Etapa 3: Anime as imagens

Com as imagens carregadas, a próxima etapa no applet é iniciar a animação dos seus elementos. Como esse é um applet com thread, o método run( ) será usado para esse propósito.

O Neko executa cinco ações sucessivas no programa:

- Corre a partir do lado esquerdo da tela.
- Pára no meio e boceja.
- Coca quatro vezes.
- Dorme.
- Acorda e corre para o lado direito da tela.

Cada uma das atividades do Neko será contida em seu próprio método. Isso torna possível reutilizar algumas das ações — como o Neko correndo pelo applet — e reorganizar a ordem das coisas, se desejado.

O primeiro método criado é nekoRun( ), que recebe dois inteiros como argumentos: start e end. Isso determina a coordenada x em que o Neko começa e pára de correr. Usando argumentos, você torna o método reutilizável. Aqui está o código inicial do corpo do método:

```
void nekorun(int start, int end) {
  // a fazer
}
```

Existem duas imagens que representam Neko correndo: Rightl.gif e Right2.gif. Elas são armazenadas nos elementos 0 e 1 do array nekoPic.

Para fazer parecer que o gato virtual está correndo pela janela, essas imagens são apresentadas em sucessão. Ao mesmo tempo, a coordenada x da imagem aumenta, para que a imagem saltitante seja desenhada cada vez mais para a direita. Um loop for é usado para circular entre os valores start e end e aumentar a coordenada x.

Para trocar de imagem, o objeto currentlmg controla a imagem atual e troca de um valor para outro a cada vez, através do loop for.

Uma chamada a repaint( ) faz a imagem controlada por currentlmg aparecer.

A última coisa a fazer no método nekoRun ( ) é dar uma pausa dentro do loop for, antes que as imagens sejam trocadas e uma nova apareça.

Como cada um dos métodos de movimentação do Neko precisa de uma pausa, um método pause ( ) é inserido no applet, que pode ser reutilizado. Esse método usa o método Thread.sleep( ), como se vê a seguir:

```
void pause(int time) {
    try {
      Thread.sleep(time);
    } catch (InterruptedException e) { }
}
```

Depois que a chamada a pause( ) é inserida, o método nekoRun( ) é composto do seguinte:

```
void nekoRun(int start, int end) {
    for (int i = start; i < end; i+=10) {
      x = i;
      // troca de imagens
      if (currentImg == nekoPics[0])
          currentImg = nekoPics[l];
      else currentimg = nekoPics[0];
      repaint( );
      pause(150);
    }
}
```

A última parte da instrução for( ) incrementa o loop em 10 pixels a cada vez, o que empurra as imagens para a direita nessa distância a cada atualização. Essa escolha, assim como a pausa de 150 milissegundos na chamada do método pause( ), foi obtida pelo método de tentativa e erro para se determinar o que ficava melhor quando passa a seqüência de animação.

Você viu que o método nekoRun ( ) armazena o quadro atual da animação no objeto currentImg, antes de chamar repaint( ). O método paint( ) do applet realizará o verdadeiro trabalho de apresentar essa imagem, como se vê a seguir:

```
public void paint(Graphics screen) {
    if (currentImg != null)
        screen.drawImage(currentImg, x, y, this);
}
```

Antes de chamar drawImage( ), o método faz um teste para certificar-se de que esse objeto currentImg existe. Se ele não fizesse isso, o objeto teria um valor null.

Agora que existe um método para se aproximar do movimento do Neko e que os métodos pause( ) e paint( ) foram definidos, algumas instruções podem ser incluídas no método run( ), como segue:

```
// Corre de um lado da tela para o meio
nekoRun(0, size( ).width / 2);
```

Cuidado

O método size( ) da classe Applet foi desaprovado após a versão 1.02 da linguagem Java; portanto, se você estiver escrevendo esse applet para ser compatível com Java 2, ele pode ser substituído por getSize( ), o que muda a chamada de nekoRun( ) para o seguinte:

```
nekoRun(0, getSize( ).width / 2);
```

A segunda atividade que Neko realiza é parar e bocejar. Cada uma delas é um quadro da animação que não é repetido; portanto, eles serão incluídos no método run( ) diretamente, em vez de serem implementados como seus próprios métodos.

O que precisa acontecer para a apresentação de cada uma dessas imagens é o seguinte:

- Mudar o valor de currentlmg para o objeto Image que deve aparecer.
- Chamar repaint( ).
- Fazer uma pausa por um período de tempo definido.

A seguir há o código:

```
// pára e faz uma pausa
currentlmg = nekoPics[2];
repaint( );
pause(lOOO);
// boceja
currentlmg = nekoPics[3];
repaint( );
pause(lOOO);
```

A terceira parte da animação será Neko se cocando, o que alterna entre os elementos 4 e 5 do array nekoPics. Não há movimentação horizontal durante essa parte da animação; portanto, o único detalhe que o método nekoScratch ( ) precisa manipular é o número de vezes que vai apresentar o Neko se cocando.

O método nekoScratch ( ) recebe um único argumento: o número de vezes a cocar. Esse argumento é usado no loop for que apresenta as duas imagens nekoPics. As instruções a seguir compõem o método:

```
void nekoScratchfint numTimes) {
    for (int i = numTimes; i > 0; i-) {
      currentlmg = nekoPics[4];
      repaintf );
      pause(150);
      currentlmg = nekoPics[5];
      repaint( );
      pause(150);
    )
}
```

Dentro do método run ( ) do applet, o método nekoScratch ( ) é chamado com um argumento igual a 4:

```
// coça quatro vezes
nekoScratch(4);
```

O método seguinte, nekoSleep( ) alternará duas imagens para mostrar o Neko dormindo. Essas imagens ocupam os elementos 6 e 7 do array e cada uma aparecerá seguida de uma pausa de 150 milissegundos. Isso utiliza técnicas que você já viu em outros métodos:

```
void nekoSleep(int numTimes) {
    for (int i = numTimes; i > 0; i-) {
      currentImg = nekoPics[6];
      repaint( );
      pause(250);
      currentImg = nekoPics[7];
      repaint( );
      pause(250);
 }

}
```

O método nekoSleep( ) é chamado no método run( ) do applet comum argumento igual a 5, como se vê a seguir:

```
// dorme por 5 "turnos"
nekoSleep(5);
```

Como a última das atividades de Neko, a imagem em que ele está acordando, em nekoPics[8], aparecerá no método run( ), seguida de outra chamada ao método nekoRun( ) para fazer o gato sair de cena pela direita. As seguintes instruções são usadas:

```
// acorda e sai correndo
currentImg = nekoPics[8];
repaint( );
pause(500);
nekoRun(x, size( ).width + 10);
```

Etapa 4: Conclusão

Todas as imagens da animação do Neko possuem fundo branco. Se a janela de seu applet é de outra cor, cada quadro da animação terá um contorno branco.

Para fazer com que cada quadro se harmonize na janela Applet, inclua a seguinte instrução no início do método run ( ):

```
setBackground(Color.white);
```

Há bastante código nesse applet, que utiliza vários métodos diferentes para conseguir uma animação razoavelmente simples. Observe que nada é feito para reduzir o tremido nesse applet, pois o tamanho da imagem e a área de desenho são suficientemente pequenos para minimizar o problema. Geralmente é uma boa idéia escrever sua animação primeiro do modo mais simples e depois acrescentar um comportamento para fazê-la funcionar melhor.

Para concluir esta seção, a listagem 10.4 mostra o código completo do applet Neko.

DIGITE **Listagem 10.4.** O texto completo de neko.java.

```
1: import java.awt.Graphics;
2: import java.awt.Image;
3: import java.awt.Color;
4:
5: public class Neko extends java.applet.Applet
6:     implements Runnable {
7:
8:    Image nekoPics[ ] = new Image[9];
9:   Image currentlmg;
10:    Thread runner;
11:   int x;
12: int y = 50;
13:
14:    public void init( ) {
15:      String nekoSrc[ ] = { "rightl.gif", "right2.gif"
16:        "stop.gif", "yawn.gif", "scratchl.gif",
17:        "scratch2.gif","sleepl.gif", "sleep2.gif",
18:        "awake.gif" };
19:
20:      for (int i=0; i < nekoPics.length; i++) {
21:        nekoPics[i] = getImage(getCodeBase( ),
22:          "images/" + nekoSrc[i]);
23:      }
24:    }
25:
26:    public void start( ) {
27:      if (runner == null) {
28:        runner = new Thread(this);
29:        runner.start( );
30:      }
31:    }
32:
33:    public void stop( ) {
34:      runner = null;
35:    }
36:
37:    public void run( ) {
38:        setBackground(Color.white);
39:        // corre de um lado da tela para o meio
40:        nekoRun(0, size( ).width / 2);
41:        // pára e faz uma pausa
42:        currentlmg = nekoPics[2];
43:        repaint( );
44:        pause(lOOO);
45:        // boceja
```

Listagem 10.4. Continuação

```
46:         currentlmg = nekoPics[3];
47:         repaint( );
48:         pause(1000);
49:         // coça quatro vezes
50:         nekoScratch(4);
51:         // dorme por 5 "turnos"
52:         nekoSleep(5);
53:         // acorda e sai correndo
54:         currentlmg = nekoPics[8];
55:         repaint( );
56:         pause(500);
57:         nekoRun(x, size( ).width + 10);
58:     }
59:
60:     void nekoRun(int start, int end) {
61:        for (int i = start; i < end; i += 10) {
62:          x = i;
63:          // troca de imagens
64:          if (currentlmg == nekoPics[0])
65:            currentlmg = nekoPicsfl];
66:          else currentlmg • nekoPics[0];
67:              repaint( );
68:          pause(150);
69:        }
70:     }
71:
72:     void nekoScratch(int numTimes) {
73:         for (int i = numTimes; i > 0; i-) {
74:           currentlmg • nekoPics[4];
75:           repaint( );
76:           pause(150);
77:           currentlmg = nekoPics[5];
78:           repaint( );
79:           pause(150);
80:         }
81:     }
82:
83:       void nekoSleep(int numTimes) {
84:         for (int i = numTimes; i > 0; i-) {
85:           currentlmg = nekoPics[6];
86:           repaint( );
87:           pause(250);
88:           currentlmg = nekoPics[7];
89:           repaint( );
90:           pause(250);
91:         }
92:     }
93:
```

Listagem 10.4. Continuação

```
94:     void pause(int time) {
95:         try {
96:           Thread.sleep(time);
97:         } catch (InterruptedException e) { }
98:     }
99:
100:    public void paint(Graphics screen) {
101:      if (currentImg != null)
102:        screen.drawImage(currentImg, x, y, this);
103:    }
104: }
```

Quando esse applet é compilado usando um compilador Java 2, mensagens de alerta aparecem, devido à utilização do método desaprovado size( ). Elas podem ser ignoradas e o applet terá sucesso em sua execução em navegadores compatíveis com Java 1.02 e 1.1, como o Netscape Navigator. Uma versão Java 2 desse applet está disponível no site da Web deste livro, no endereço http://www.prefect.com/java21.

Para testar esse applet, crie uma página da Web em que a janela Applet Neko possua uma largura igual a 300 pixels e altura de 200. A Figura 10.5 mostra o resultado.

Figura 10.5
O applet Neko.

Uma observação sobre o descarte de contextos GraDhics

Embora o coletor de lixo da linguagem Java se desfaça automaticamente dos objetos quando eles não são mais necessários em um programa, isso não pode ser feito no caso dos objetos Graphics, que são criados para manipular buffers fora da tela.

Como os objetos não-utilizados ocupam memória e podem prejudicar o desempenho da linguagem Java, você deve usar o método dispose( ) da classe Graphi cs para se desfazer explicitamente de um de seus objetos, quando ele não

for necessário. Um bom lugar para se colocar isso em um applet é em seu método destroy( ), que foi descrito no Dia 8. Esse método é chamado sem nenhum argumento, como se vê no exemplo a seguir:

```
public void destroy( ) {
    offscreenGraphics.dispose( );
}
```

Bufferização dupla

O próximo exemplo, o applet Checkers, utiliza uma técnica chamada bufferização dupla para melhorar o desempenho da animação.

NOVO TERMO

A *bufferização dupla* descreve o processo de desenhar um quadro inteiro de uma animação em uma área não-visível, antes de copiá-la na janela visível de um programa. A área fora da tela é chamada de *buffer*.

Com a bufferização dupla, você cria uma segunda superfície (fora da tela, por assim dizer), realiza toda a pintura nessa superfície fora da tela e, em seguida, desenha a superfície inteira de uma vez no applet (e na tela), no final — em vez de desenhar na superfície gráfica do applet. Como todo o trabalho ocorre na realidade nos bastidores, as partes intermediárias do processo de desenho não podem aparecer acidentalmente e quebrar a seqüência da animação.

Nem sempre a bufferização dupla é a melhor solução. Se seu applet tem problemas de tremido, tente anular o método update( ) e desenhar primeiro apenas partes da tela; isso pode resolver seu problema. A bufferização dupla é menos eficiente do que a bufferizaçáo normal e também ocupa mais memória e espaço; portanto, em alguns casos, essa pode não ser a melhor solução. No entanto, para praticamente eliminar o tremido da animação, a bufferização dupla funciona excepcionalmente bem.

Para criar um applet que use bufferização dupla, você precisa de dois itens: uma imagem fora da tela para desenhar e um contexto gráfico dessa imagem. Esses dois itens juntos imitam o efeito da superfície de desenho do applet: o contexto gráfico (uma instância de Graphics) para fornecer os métodos de desenho, como drawImage( ) (e drawString( )), e o objeto Image para conter os pontos que serão desenhados.

Existem quatro etapas principais para se incluir bufferização dupla em seu applet. Primeiramente, sua imagem de fora da tela e seu contexto gráfico precisam ser armazenados em variáveis de instância para que você possa passá-los para o método paint( ). Declare as seguintes variáveis de instância em sua definição de classe:

```
Image offscreenImage;
Graphics offscreen;
```

Em segundo lugar, durante a inicialização do applet, você vai criar um objeto Image e um objeto Graphics e atribuí-los a essas variáveis. (Você tem de esperar até a inicialização para saber o tamanho que eles vão ter.) O método createImage( ) fornece uma instância de Image que você pode enviar para o método getGraphics( ) para obter um novo contexto Graphics dessa imagem:

```
offscreenImage = createImage(size( ).width,
    size( ).height);
offscreen = offscreenImage.getGraphics( );
```

Agora, quando você tiver de desenhar na tela (normalmente em seu método paint( )), em vez de desenhar na imagem gráfica da pintura, desenhe na imagem gráfica fora da tela. Por exemplo, para desenhar uma imagem chamada bug na posição 10,10, utilize a seguinte linha:

```
offscreen.drawImage(bug, 10, 10, this);
```

Finalmente, no fim de seu método de pintura, depois que todo o desenho na imagem fora da tela estiver concluído, acrescente a seguinte linha para colocar o buffer de fora da tela na tela real:

```
screen.drawImage(offscreenImage, 0, 0, this);
```

E claro que, provavelmente, você vai querer anular update( ) para que ele não limpe a tela entre as pinturas:

```
public void update(Graphics g) {
    paint(g);
}
```

Agora, reveja essas quatro etapas:

1. Insira variáveis de instância para conter a imagem e os contextos gráficos para o buffer de fora da tela.
2. Crie uma imagem e um contexto gráfico quando seu applet for inicializado.
3. Faça toda a pintura de seu applet no buffer de fora da tela e não na superfície de desenho do applet.
4. No final de seu método paint( ), desenhe o buffer de fora da tela na tela real.

O applet Checkers

O applet Checkers desenha uma peça vermelha do jogo de damas e a movimenta para frente e para trás, entre um quadrado branco e um preto. Se ela fosse desenhada usando-se as mesmas técnicas que os applets Neko e ColorSwirl, haveria um tremido terrível, quando a peça passasse sobre fundos de cores diferentes. Anular update( ) para que a janela não seja limpa não removeria o problema do tremido nesse caso.

O applet Checkers usa os métodos fillRect( ) e fill0val( ) para desenhar o tabuleiro e a peça. Isso faz parte do método pai nt ( ) do applet.

Para mover a peça de um lado para outro, uma variável inteira chamada xPos controla onde ela deve aparecer em dado momento. O valor dessa variável muda continuamente no método run( ).

Isso é coerente com os applets de animação que você criou até agora. O modo como o método paint( ) manipula as atualizações na janela Applet é novo.

Em vez de se desenhar cada elemento da animação diretamente na janela Applet, eles serão desenhados primeiramente em um buffer fora da tela. Quando um quadro da animação inteiro estiver completo, esse quadro será então desenhado na janela Applet.

Usando-se esse método "não fazer nada até que tudo esteja pronto", nenhum tremido será exibido, devido à apresentação de diferentes objetos em momentos ligeiramente diferentes.

A primeira etapa na bufferização dupla é criar um objeto Image para conter o quadro fora da tela enquanto ele está sendo completado, e um objeto Graphi cs que permita desenhar nessa área de imagem de fora da tela. As instruções a seguir criam esses objetos:

```
Image offscreenlmg;
Graphics offscreen;
```

Os objetos serão atribuídos a essas variáveis no método init( ) do applet:

```
public void init( ) {
 offscreenlmg = createlmage(size( ).width, size( ).height);
 offscreen = offscreenlmg.getGraphics( );
}
```

Em seguida, o método pai nt ( ) é modificado para desenhar no buffer de fora da tela, em vez de usar o objeto Graphics principal:

```
public void paint(Graphics screen) {
 // Desenha o fundo
 offscreen.setColor(Color.black);
 offscreen.fillRect(0, 0, 100, 100);
 offscreen.setColor(Color.white);
 offscreen.fillRect(100, 0, 100, 100);
 // Desenha a peça
 offscreen.setColor(Color.red);
 offscreen.fi1l0val(xPos, 5, 90, 90);
 screen.drawlmage(offscreenlmg, 0, 0, this);
}
```

Observe a última instrução do método, que é a única que não apresenta nada na janela Applet. Essa instrução apresenta a imagem de fora da tela completa, nas coordenadas (0,0). Como offscreenlmg foi criado de forma a ter o mesmo tamanho que a área de fora da tela, ele preenche completamente a janela Applet.

Para concluir, o objeto offscreen é descartado no método destroy( ) do applet, como indicado a seguir:

```
public void destroy( ) {
 offscreen.dispose( );
}
```

A listagem 10.5 mostra o código-fonte completo do applet Checkers.

DIGITE ● **Listagem 10.5.** O texto completo de checkers.java.

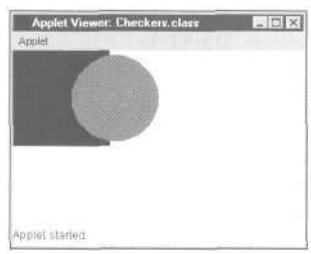
```
1: import java.awt.*;
2:
3: public class Checkers extends java.applet.Applet implements Runnable {
4:     Thread runner;
5:     int xPos = 5;
6:     int xMove = 4;
7:     Image offscreenlmg;
8:     Graphics offscreen;
9:
10:
11:     public void init( ) {
12:         offscreenlmg = createlmage(size( ).width, size( ).height);
13:         offscreen = offscreenlmg.getGraphics( );
14:    }
15:
16:     public void start( ) {
17:         if (runner == null); {
18:             runner = new Thread(this);
19:             runner.start( );
20:       }
21:    }
22:
23:     public void stop( ) {
24:         runner = null;
25:    }
26:
27:     public void run( ) {
28:         Thread thisThread = Thread.currentThread( );
29:         while (runner == thisThread) {
30:             xPos += xMove;
31:             if ((xPos > 105) | (xPos < 5))
32:               xMove *= -1;
33:             repaint( );
34:            try {
35:               Thread.sleep(100);
36:             } catch (InterruptedException e) { }
37:    }
38: }
39:
40:  public void update(Graphics screen) {
```

Listagem 10.5. O texto completo de checkers.java.

```
41:     paint(screen);
42:   }
43:
44:     public void paint(Graphics screen) {
45:          // Desenha o fundo
46:          offscreen.setColor(Color.black);
47:          offscreen.fi11Rect(0,0,100,100);
48:          offscreen.setColor(Color.white);
49:          offscreen.fillRect(100,0,100,100);
50:          // Desenha a peça
51:          offscreen.setColor(Color.red);
52:          offscreen.fill0val(xPos,5,90,90);
53:          screen.drawlmage(offscreenlmg, 0, 0, this);
54:     }
55:
56:     public void destroy( ) {
57:          offscreen.dispose( );
58:   }
59: }
```

Você pode testar esse applet em uma página da Web com uma janela Applet com atributos height=200 e width=300. A Figura 10.6 mostra o resultado.

Figura 10.6
O *applet Checkers.*



Recuperando e usando sons

A linguagem Java oferece suporte à reprodução de arquivos de som através da classe Applet, e você pode reproduzir um som apenas uma vez ou como um loop de som repetitivo.

Antes da versão 2, a linguagem Java podia manipular apenas um formato de áudio: 8 kHz mono AU com codificação de lei mi (assim chamada devido a letra grega "m", ou mi). Se você quisesse usar algo que estivesse em um formato como o WAV, tinha de transformá-lo em AU de lei mi, freqüentemente com perda de qualidade.

A linguagem Java 2 inclui um suporte muito mais completo a áudio. Você pode carregar e reproduzir arquivos de som digitalizados nos seguintes formatos: AIFF, AU e WAV. Três formatos de arquivo de som baseado em MIDÍ também são suportados: MIDI Tipo 0, MIDI Tipo 1 e KMF. O suporte a som bastante melhorado pode manipular dados de áudio de 8 ou 16 bits em mono ou estéreo, e as taxas de amostragem podem variar de 8 kHz a 48 kHz.

O modo mais simples de recuperar e reproduzir um som é pelo método play( ) da classe Applet. O método play( ), assim como o getImage( ), assume uma de duas formas:

- play( ) com um argumento — um objeto URL — carrega e reproduz o clip de áudio armazenado nesse URL.
- play( ) com dois argumentos — um URL de base e um nome de caminho de pasta — carrega e reproduz esse arquivo de áudio. O primeiro argumento freqüentemente será uma chamada a getDocumentBase( ) ou getCodeBase( ), conforme você viu no caso de getImage( ).

A instrução a seguir recupera e reproduz o som zap.au, que está armazenado no mesmo lugar que o applet:

```
play(getCodeBase( ), "zap.au");
```

O método play( ) recupera e reproduz o som dado assim que possível, depois que ele é chamado. Se o arquivo de som não puder ser encontrado, a única indicação que você receberá de um problema será o silêncio. Nenhuma mensagem de erro será apresentada.

Para reproduzir um som repetidamente, iniciar e interromper o som ou reproduzi-lo repetidamente como um loop, você deve carregá-lo em um objeto AudioClip, usando o método getAudioClip do applet. AudioClip faz parte do pacote java.applet; portanto, ele deve ser importado para ser usado em um programa.

O método getAudioClip( ) recebe um ou dois argumentos, da mesma maneira que o método play( ). O primeiro (ou único) argumento é um URL identificando o arquivo de som, e o segundo é uma referência de caminho de pasta.

A instrução a seguir carrega um arquivo de som no objeto clip:

```
AudioClip clip = getAudioClip(getCodeBase( ),
    "audio/maritnba.wav");
```

Nesse exemplo, o nome de arquivo inclui uma referência de pasta, de modo que o arquivo marimba.wav será carregado a partir da subpasta áudio.

O método getAudioClip( ) pode ser chamado apenas dentro de um applet. A partir da versão 2 da linguagem Java, os aplicativos podem carregar arquivos de som usando o método newAudioClip( ) da classe Applet. Aqui está o exemplo anterior, reescrito para uso em um aplicativo:

```
AudioClip clip = newAudioClip("audio/marimba.wav");
```

Após ter criado um objeto AudioClip, você pode chamar os métodos play( ), stop( ) e loop( ) nele. Eles fazem o que você poderia esperar — play( ) reproduz o som, stop( ) interrompe a reprodução e loop( ) o reproduz repetidamente.

Se os métodos getAudioClip( ) ou newAudioClip( ) não puderem localizar o arquivo de som indicado por seus argumentos, o valor do objeto AudioClip será null. Tentar reproduzir um objeto null resulta em um erro; portanto, teste essa condição antes de usar um objeto AudioClip.

Mais de um som pode ser reproduzido simultaneamente — eles serão misturados durante a reprodução.

Um detalhe importante a ser observado ao se usar um loop de som em um applet é que ele não parará automaticamente, quando o thread que estiver em execução no applet for interrompido. Se um usuário da Web mudar para outra página, o som continuará a ser reproduzido, o que provavelmente não vai fazer com que você ganhe amigos entre o público que navega na Web.

Você pode corrigir esse problema usando o método stop( ) no som em loop, ao mesmo tempo em que o thread do applet está sendo interrompido.

A listagem 10.6 é um applet que reproduz dois sons: um som em loop chamado loop.au e uma cometa soando, chamada beep.au, que se reproduz a cada cinco segundos.

DIGITE **Listagem 10.6.** O texto completo de audioloop.java.

```
1: import java.awt.Graphics;
2: import java.applet.AudioClip;
3:
4: public class AudioLoop extends java.applet.Applet
5:     implements Runnable {
6:
7:     AudioClip bgSound;
8:     AudioClip beep;
9:     Thread runner;
10:
11:     public void start( ) {
12:         if (runner •« null) {
13:           runner = new Thread(this);
14:           runner.start( );
15:       }
16:     }
17:
18:     public void stop( ) {
19:         if (runner != null) {
20:           if (bgSound != null)
21:             bgSound.stop( );
22:           runner = null;
23:       }
```

Listagem 10.6. O texto completo de audioloop.java.

```
24:     }
25:
26:     public void init( ) {
27:       bgSound = getAudioClip(getCodeBase( ),"loop.au");
28:       beep • getAudioClip(getCodeBase( ), "beep.au");
29:     }
30:
31:     public void run( ) {
32:       if (bgSound != null)
33:         bgSound.loop( );
34:       Thread thisThread • Thread.currentThread( );
35:       while (runner == thisThread) {
36:         try {
37:           Thread.sleep(5000);
38:         } catch (InterruptedException e) { }
39:         if (beep != null)
40:           beep.play( );
41:       }
42:    }
43:
44:     public void paint(Graphics screen) {
45:       screen.drawString("Playing Sounds ...", 10, 10);
46:     }
47: }
```

Para testar o Audi oLoop, crie uma página da Web com uma janela de applet que tenha uma altura de 100 e largura de 200. Os arquivos de áudio loop.au e beep.au devem ser copiados a partir do site da Web deste livro (http://www. prefect.com/java21) para a pasta \J21work, em seu sistema. Quando o applet for executado, a única saída visual será um string, mas você deverá ouvir dois sons tocando, enquanto o applet funciona.

O método init( ), nas linhas 26 a 29, carrega os arquivos de som loop.au e beep. au. Nesse método, nenhuma tentativa é feita no sentido de garantir que os arquivos sejam realmente carregados, o que resultaria em valores null dos objetos bgsound e beep. Isso será testado em outro lugar, antes que os arquivos de som sejam usados, como nas linhas 32 e 39, quando os métodos loop( ) e play( ) são utilizados nos objetos AudioCl ip.

As linhas 20 e 21 desligam o som em loop, se o thread também for interrompido.

Resumo

Hoje, você aprendeu diversos métodos para usar e anular — start( ), stop( ),paint( ),repaint( ),run( )eupdate( )—e conheceu os fundamentos da criação e uso de threads. Você também aprendeu sobre o uso de imagens

em seus applets — localizá-los, carregá-los e utilizar o método drawlmage( ) para apresentá-los e animá-los.

Uma técnica de animação que agora você pode usar é a buf ferização dupla, que praticamente elimina o tremido em sua animação, às custas de certa eficiência e velocidade. Usando imagens e contextos gráficos, você pode criar um buffer fora da tela para nele desenhar, cujo resultado é então apresentado na tela no último momento possível.

Você aprendeu a usar sons, que podem ser incluídos em seus applets quando forem necessários — em momentos específicos ou como sons de fundo que podem ser repetidos enquanto o applet é executado. Você aprendeu a localizar, carregar e reproduzir sons, usando os métodos play( ) e getAudio-Clip( ).

Perguntas e respostas

P **No programa Neko, você coloca o carregamento das imagens no método init( ). Parece-me que isso poderia fazer com que a linguagem Java levasse um longo tempo para carregar todas essas imagens e, como i ni t ( ) não *é* o thread principal do applet, haveria uma pausa nítida nesse momento. Por que não colocar o carregamento das imagens no início do método run( )?**

R Detalhes sutis acontecem nos bastidores. O método getlmage( ) não carrega realmente a imagem; na verdade, ele retorna um objeto Image quase que instantaneamente; portanto, ele não está ocupando um grande período de tempo de processamento na inicialização. Os dados de imagem para os quais get Image ( ) aponta não são realmente carregados até que a imagem seja necessária. Desse modo, a linguagem Java não precisa manter imagens enormes na memória, caso o programa vá utilizar apenas um pequeno trecho. Em vez disso, ele pode manter apenas uma referência para esses dados e recuperar o que precisar posteriormente.

P **Eu compilei e executei o applet Neko. Algo estranho está acontecendo; a animação começa no meio e perde quadros. É como se algumas das imagens fossem carregadas quando o applet está em execução.**

R E exatamente isso que está acontecendo. Como o carregamento de imagens não carrega realmente a imagem imediatamente, seu applet pode estar simplesmente animando telas em branco, enquanto as imagens **ainda** estão sendo carregadas. Dependendo do tempo que isso levar, seu applet pode parecer estar

iniciando no meio, perdendo quadros ou nem mesmo funcionando.

Existem três soluções possíveis para esse problema. A primeira é ter o loop de animação (isto é, começar novamente desde o início, quando ele parar). Finalmente, as imagens serão carregadas e a animação funcionará corretamente. A segunda solução (que não é muito boa) é dar um intervalo antes de iniciar a animação, para fazer uma pausa enquanto as imagens são carregadas. A terceira e melhor solução é usar observadores de imagem para certificar-se de que nenhuma parte da animação seja reproduzida antes que suas imagens tenham sido carregadas. Consulte a documentação da interface ImageObserver para saber os detalhes.

SEMANA 2

Dia 11

Criando interfaces com o usuário simples para applets

Com a popularidade dos sistemas operacionais Macintosh da Apple e Microsoft Windows, a maioria dos usuários de computador atualmente espera que seu software apresente uma interface gráfica com o usuário e itens que possam ser controlados com um mouse. No trabalho, o direito de apontar e dar um clique se equipara à vida, à liberdade e à aquisição de um bom plano de aposentadoria.

Essas amenidades de software são amigáveis para o usuário, mas não para o programador em muitas linguagens. Escrever software com janelas pode ser uma das tarefas mais desafiadoras para um projetista iniciante.

Felizmente, a linguagem Java simplificou o processo com o Abstract Windowing Toolkit, um conjunto de classes para a criação e utilização de interfaces gráficas com o usuário.

Hoje, você usará o kit de ferramentas para criar uma interface com o usuário para um applet. Você vai usar as técnicas de Java 1.02, pois essa continua sendo a versão padrão da linguagem entre o público que utiliza applets.

Amanhã, você aprenderá a organizar todos os componentes de uma interface com o usuário. No Dia 13, você concluirá uma interface com o usuário, tornando-a sensível ao controle do usuário.

Após ter aprendido a criar programas usando o Abstract Windowing Toolkit, você estará pronto para usar as técnicas da linguagem Java 2 para criar aplicativos, nos Dias 20 e 21.

O Abstract Windowina Toolkit

O Abstract Windowing Toolkit, também chamado de AWT, é um conjunto de classes que permite criar uma interface gráfica com o usuário e receber entrada do usuário a partir do mouse e do teclado.

Como a linguagem Java é independente de plataforma, o AWT oferece um modo de projetar uma interface que terá a mesma aparência e funcionalidade geral em todos os sistemas em que for executada.

Um detalhe que você aprenderá quando criar applets Java com o AWT é que algumas coisas não são totalmente coerentes entre plataformas. Os diferentes ambientes de runtime Java criados pela Netscape, Microsoft e outras empresas para seus navegadores, nem sempre estão de acordo com relação ao modo como uma interface AWT deve funcionar. É importante testar seus applets com janelas no máximo de plataformas e navegadores possíveis.

Usando-se o AWT, uma interface com o usuário é composta de três itens:

- *Componentes.* Tudo o que pode ser colocado em uma interface com o usuário, incluindo botões em que se pode dar um clique com o mouse, listas rolantes, menus instantâneos, caixas de seleção e campos de texto.
- *Containers.* Um componente que pode conter outros componentes. Você tem trabalhado com um deles o tempo todo — a janela Applet — e outros incluem painéis, caixas de diálogo e janelas independentes.
- *Gerenciadores de layout.* Um objeto que define como os componentes de um contêiner serão organizados. Você não vê o gerenciador de layout em uma interface, mas definitivamente vê os resultados de seu trabalho.

Todas as classes do AWT fazem parte do pacote java.awt. Para tornar todas as suas classes disponíveis em um programa, a seguinte instrução pode ser usada no início de um arquivo de código-fonte:

```
import java.awt.*;
```

Isso importa todos os componentes, contêineres e gerenciadores de layout que você vai usar para projetar uma interface. Você também pode usar instruções import individualmente com as classes que está utilizando em um programa.

As classes do AWT, assim como todas as partes da biblioteca de classe Java, são organizadas em uma hierarquia de herança. Quando você aprender a usar uma classe AWT, aprenderá alguns detalhes sobre como utilizar outras classes que herdam da mesma superclasse.

Componentes da interface com o usuário

Os componentes são colocados em uma interface com o usuário através de sua inclusão em um contêiner. O próprio contêiner é um componente; portanto, ele pode ser incluído em outros contêineres. Você vai usar essa funcionalidade quando começar a trabalhar com gerenciadores de layout para organizar uma interface.

O modo mais fácil de demonstrar um design de interface é utilizar o contêiner com que você trabalhou o tempo todo — a classe Applet.

Adicionando componentes em um contêiner

Um componente é adicionado a um contêiner com as duas etapas a seguir:

- Crie o componente.
- Chame o método add( ) do contêiner com o componente.

Como todos os applets são contêineres, você pode usar o método add( ) dentro de um applet para incluir um componente diretamente na janela Applet.

Cada componente de interface com o usuário AWT é uma classe; portanto, você gera um componente criando um objeto dessa classe.

A classe Button representa botões que podem receber um clique de mouse em uma interface. Você pode criar um botão especificando o rótulo em seu método construtor, como na instrução a seguir:

```
Button panic = new Button("Panic!");
```

Isso cria um objeto Button rotulado com o texto "Panic!".

Uma vez que você tenha criado um componente, o modo mais simples de incluí-lo em um contêiner é chamando o método add( ) do contêiner com o componente sendo o único argumento.

Como um applet é um contêiner, a seguinte instrução poderia ser usada em um applet para incluir o objeto panic na janela Applet:

```
add(panic);
```

A inclusão de um componente não o faz aparecer imediatamente. Em vez disso, ele aparecerá quando o método paint( ) de seu contêiner for chamado. Isso é algo que a linguagem Java manipula nos bastidores, mas você pode forçar uma chamada a paint( ) em um applet, usando seu método repaint( ).

Quando inclui um componente em um contêiner, você não especifica uma coordenada x,y que indique onde o componente deve ser colocado. A organização dos componentes é manipulada pelo gerenciador de layout que está em vigor para o contêiner.

Nota

Você aprenderá mais sobre os gerenciadores de layout amanhã. O layout padrão para um contêiner é colocar cada componente em uma linha, da esquerda para a direita, até que não haja mais espaço e, em seguida, continuar colocando componentes na próxima linha. Isso se chama *layout de fluxo* e é manipulado pela classe FlowLayout.

Em um applet, o melhor lugar para se criar componentes e incluí-los em contêineres é o método init( ). Isso está demonstrado em um applet de um botão, apresentado na listagem 11.1. O applet Slacker cria um objeto Button e o inclui na janela Applet. O botão é apresentado quando o método paint( ) do applet — herdado da classe Appl et — é chamado.

DIGITE

Listagem 11.1. O texto completo de slacker.java.

```
1: import java.awt.*;
2:
3: public class Slacker extends java.applet.Applet {
4:     String note = "I am extremely tired and would prefer not " +
5:         "to be clicked. Please interact somewhere else.";
6:     Button tired = new Button(note);
7:
8:     public void init( ) {
9:         add(tired);
10:    }
11: }
```

Teste esse applet em uma página, usando a seguinte tag <APPLET>:

```
<applet code="Slacker.class" width=550 height=75>
</applet>
```

A Figura 11.1 mostra o resultado do uso do appletviewer.

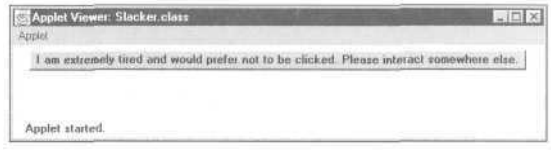


Figura 11.1
O *applet Slacker.*

Rótulos

O componente de interface com o usuário mais simples é o rótulo, criado a partir da classe Label. Os *rótulos* são utilizados freqüentemente para identificar o objetivo de outros componentes em uma interface e eles não podem ser editados diretamente por um usuário.

Usar um rótulo para texto é preferível do que usar o método drawString ( ) pelos seguintes motivos:

- Os rótulos são desenhados automaticamente após a criação e não precisam ser manipulados explicitamente no método paint( ).
- Os rótulos serão dispostos de acordo com o gerenciador de layout que estiver em uso, em vez de serem especificamente vinculados a um endereço x,y, como um string desenhado.

Para criar um rótulo, use um dos construtores a seguir:

- Label ( ) cria um rótulo vazio, com seu texto alinhado à esquerda.
- Label *{String)* cria um rótulo com o string de texto fornecido, também alinhado à esquerda.
- Label *(String, ínt)* cria um rótulo com o string de texto fornecido e o alinhamento indicado pelo argumento i nt. As seguintes variáveis de classe são usadas para definir o alinhamento: Label .RIGHT, Label .LEFT e Label.CENTER.

Você pode mudar a fonte do rótulo com o método setFont( ), sobre o qual aprendeu durante o Dia 9. Esse método pode ser chamado no contêiner do rótulo (como um applet), o que afeta todos os componentes do contêiner, ou no próprio rótulo.

O método setText *(String)* de um rótulo pode ser usado para mudar o texto do rótulo depois que ele tiver sido criado. O novo texto indicado por *String* é apresentado quando o componente é repintado. Você também pode usar o método getText( ) para indicar qual é o texto atual do rótulo.

A listagem 11.2 contém um applet simples que cria alguns rótulos em Helvetica Bold.

DIGITE

Listagem 11.2. O texto completo de labels.java.

```
1: import java.awt.*;
2:
3: public class Labels extends java.applet.Applet {
4:      Label lefty = new Label("Bleeding heart!");
5:      Label center = new Label("Centrist!", Label.CENTER);
6:      Label righty = new Label("Hardliner!", Label.RIGHT);
7:      Font lf = new Font("Helvetica", Font.BOLD, 14);
8:    GridLayout layout = new Gn'dLayout(3,l);
9:
10:      public void init( ) {
11:          setFont(lf);
12:          setLayout(layout);
13:          add(lefty);
14:          add(center);
15:          add(righty);
16:      }
17: }
```

Teste esse applet usando a seguinte tag <APPLET>:

```
<applet code="Labels.class" height=150 width=175>
</applet>
```

A Figura 11.2 mostra a saída desse applet no appietviewer. Essa é uma boa ferramenta para se usar nesse programa, pois você pode redimensionar a janela e ver como isso resulta em um realinhamento dos três rótulos. O rótulo "Hardliner!" fica preso na margem direita da janela do applet e o rótulo "Centrist!" permanece centralizado.

Figura 11.2
O *applet Labels.*

As linhas 8 e 12 desse applet são usadas para criar um objeto GridLayout e usar esse objeto para estabelecer o layout do contêiner. Isso será abordado posteriormente — teve de ser utilizado aqui para ilustrar o alinhamento, pois os rótulos não são alinhados no comportamento padrão dos contêineres, que é layout de fluxo. As linhas 8 e 12 são usadas para organizar os componentes em uma grade com uma coluna e três fileiras.

Botões

Os botões que podem receber um clique de mouse podem ser criados com a classe Button, como você viu no applet Slacker. Os botões são úteis em uma interface para disparar uma ação, como um botão Quit para sair de um programa.

Para criar um botão, use um dos seguintes construtores:

- Button ( ) cria um botão sem rótulo de texto que indique sua função.
- Button *(String)* cria um botão com o string fornecido como rótulo.

Após criar um objeto Button, você pode definir seu rótulo com o método setLabel *(String)* e obter o texto do rótulo com o método getLabel ( ).

A listagem 11.3 contém o applet VCR, que apresenta vários comandos em botões conhecidos.

DIGITE **Listagem 11.3.** O texto completo de ver. java.

```
1: import java.awt.*;
2:
3: public class VCR extends java.applet.Applet {
4:     Button rewind = new Button("Rewind");
```

Listagem 11.3. Continuação

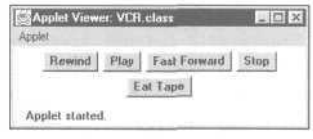
```
5:      Button play = new Button("Play");
6:      Button ff = new Button("Fast Forward");
7:      Button stop » new Button("Stop");
8:      Button eat = new Button("Eat Tape");
9:
10: public void init( ) {
11:      add(rewind);
12:      add(play);
13:      add(ff);
14:      add(stop);
15:      add(eat);
16:      }
17: }
```

Teste o applet VCR usando o seguinte código HTML:

```
<applet code="VCR.class" height=60 width=300>
</applet>
```

A Figura 11.3 mostra esse applet carregado como appletviewer. Observe que o botão Eat Tape aparece em uma nova linha, pois não havia espaço para ele na linha anterior. Se você fizesse a janela Applet com 500 pixels de largura em vez de 300, todos os cinco botões ficariam alinhados.

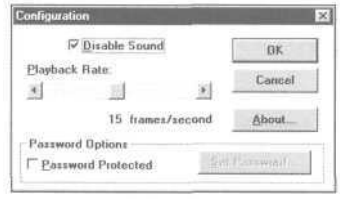
Figura 11.3
O applet VCR.



Caixas de seleção

As *caixas de seleção* são caixas rotuladas ou não que podem estar "marcadas" ou vazias. Normalmente, elas são usadas para selecionar ou anular a seleção de algum tipo de opção em um programa, como as caixas de seleção Disable Sound e Password Protected de um protetor de tela do Windows, o que está ilustrado na Figura 11.4.

Figura 11.4
Uma caixa de diálogo que utiliza caixas de seleção.



Normalmente, as caixas de seleção são *não-exclusivas*, significando que, se você possui cinco caixas de seleção em um contêiner, todas elas podem estar marcadas ou desmarcadas simultaneamente.

Esse componente pode ser organizado em grupos de caixas de seleção, os quais às vezes são chamados de *botões de rádio*. Eles recebem esse nome devido aos antigos rádios de carro, nos quais pressionar um botão desligava todos os outros que estavam pressionados.

Os dois tipos de caixas de seleção são criados usando-se a classe Checkbox. Você pode criar uma caixa de seleção não-exclusiva usando um dos seguintes construtores:

- Checkbox( ) cria uma caixa de seleção sem rótulo e desmarcada.
- Checkbox (*String)* cria uma caixa de seleção desmarcada, com o stnng fornecido como rótulo.

Após criar um objeto Checkbox, você pode usar o método set-*State(boolean)* com o valor true como argumento, para marcar a caixa, efal se para desmarcá-la. O método getState( ) retornará um valor booleano indicando o status (marcado ou desmarcado) atual da caixa de seleção.

Cinco caixas de seleção são criadas na listagem 11.4, que é um applet que permite selecionar até cinco celebridades nascidas na República Tcheca. Todos os cinco são tchecos, mas apenas um está marcado — a modelo e atriz Paulina Ponzkova.

DIGITE

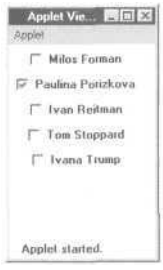
Listagem 11.4. O texto completo de checkaczech.html.

```
1: import java.awt.*;
2:
3: public class CheckACzech extends java.applet.Applet {
4:      Checkbox cl = new Checkbox("Milos Forman");
5:      Checkbox c2 = new Checkbox("Paulina Porizkova");
6:      Checkbox c3 = new Checkbox("Ivan Reitman");
7:      Checkbox c4 = new Checkbox("Tom Stoppard");
8:      Checkbox c5 = new Checkbox("Ivana Trump");
9:
10:       public void init( ) {
11:           add(cl);
12:           c2.setState(true);
13:           add(c2);
14:           add(c3);
15:           add(c4);
16:           add(c5);
17:       }
18: }
```

A Figura 11.5 mostra a saída desse applet, que pode ser testado com a seguinte tag <APPLET>:

```
<applet code="CheckACzech.class" height=200 width=150>
</applet>
```

Figura 11.5
O *applet CheckACzecb.*



Para organizar várias caixas de seleção em um grupo, para que apenas uma possa ser selecionada por vez, um objeto CheckboxGroup é criado com uma instrução como a seguinte:

```
CheckboxGroup radio = new CheckboxGroup( );
```

O objeto CheckboxGroup controla todas as caixas de seleção desse grupo. Você usa esse objeto como um argumento extra no construtor Checkbox.

Checkbox Sfrirtg, *CheckboxGroup, boolean)* cria uma caixa de seleção rotulada com o string fornecido, que pertence ao CheckboxGroup indicado pelo segundo argumento. O terceiro argumento será igual a true se a caixa estiver marcada, e false, no caso contrário.

Cuidado

A partir da versão 2 da linguagem Java, o método Checkbox *String, CheckboxGroup, boolean)* foi desaprovado, significando que um método melhor está disponível. Se você não estiver escrevendo um applet Java 1.02, o método Checkbox(Stringf, *boolean, CheckboxGroup)* deve ser usado. A utilização é a mesma, mas o segundo e o terceiro argumentos são invertidos.

O exemplo a seguir cria um grupo e duas caixas de seleção que pertencem a ele:

```
CheckboxGroup betterDarrin = new CheckboxGroup( );
Checkbox r1 = new Checkbox("Dick York", betterDarrin, true);
Checkbox r2 = new Checkbox("Dick Sargent", betterDarrin, false);
```

O objeto betterDarrin é usado para agrupar as caixas de seleção r1 e r2. O objeto ri, que possui o rótulo "Dick York", está selecionado. Apenas um membro do grupo pode estar selecionado por vez; portanto, é impossível usar true como o terceiro argumento de r1 e r2. (Além disso, como todos os fãs da série de televisão dos anos 60 A *Feiticeira* devem atestar, o ator Dick York foi um melhor Darrin Stevens do que seu sucessor, Dick Sargent.)

Se você tentar usar true em mais de uma caixa de seleção de um grupo, a última será a única selecionada. Um grupo pode aparecer sem nenhuma caixa de seleção marcada.

Um grupo de caixas de seleção está demonstrado na listagem 11.5, um applet que fornece caixas de seleção para cinco personalidades polonesas e seleciona uma — Krzysztof Kieslowski, o falecido diretor *de A liberdade é azul, A igualdade é branca* e *A fraternidade é vermelha.*

DIGITE

Listagem 11.5. O texto completo de pickapole.Java.

```
1:  import java.awt.*;
2:
3:  public class PickAPole extends java.applet.Applet {
4:       CheckboxGroup p = new CheckboxGroup( );
5:       Checkbox p1 = new Checkbox("Samuel Goldwyn", p, false);
6:       Checkbox p2 = new Checkbox("Krzysztof Kieslowski", p, true);
7:       Checkbox p3 = new Checkbox("Klaus Kinski", p, false);
8:       Checkbox p4 = new Checkbox("Joanna Pacula", p, false);
9:       Checkbox p5 - new Checkbox("Roman Polanski", p, false);
10:
11:     public void init( ) {
12:          add(pl);
13:          add(p2);
14:          add(p3);
15:          add(p4);
16:          add(pS);
17:     }
18: }
```

Use a seguinte tag <APPLET> em uma página da Web para testar esse applet, que aparece na Figura 11.6:

```
<applet code="PickAPole.class" height=200 width=150>
</applet>
```

Figura 11.6
O *applet PickAPole.*

O método setCurrent *Checkbox)* pode ser usado para definir a caixa de seleção marcada no momento no grupo. Também existe um método getCurrent( ), que retorna a caixa de seleção marcada no momento.

Listas de escolha

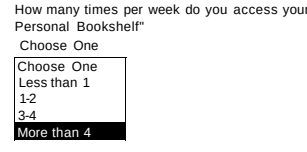
As *listas de escolha,* que são criadas a partir da classe Choice, são componentes que permitem a um único item ser escolhido em uma lista suspensa. Você encontra essas listas freqüentemente quando preenche um formulário em uma página da World Wide Web. A Figura 11.7 mostra um exemplo do site da Web da Macmillan Personal Bookshelf.

O Personal Bookshelf é um programa da Macmillan que permite aos usuários ver até cinco livros, incluindo os da Sams Publishing, em texto integral na Web. Para conhecer os detalhes, visite a seguinte página da Web:

http://www.mcp.com/personal/

Figura 11.7

Exemplo de uma listo de escolha.



O primeiro passo na criação de uma lista de escolha é gerar um objeto Choice para conter a lista, como mostra a instrução a seguir:

```
Choice gender = new Choice( );
```

Os itens são inseridos em uma lista de escolha usando-se o método add Item (String) no objeto. As instruções a seguir inserem dois itens na lista de escolha de sexo:

```
gender.addItem("Male");
gender.addItem("Female");
```

Você pode continuar a usar addltem( ) para aumentar a lista, depois que a lista de escolha tiver sido incluída em um contêiner.

O método addItem(String) foi desaprovado após a versão 1.02 da linguagem Java. Use em seu lugar o método add*(String),* quando projetar nas versões posteriores da linguagem.

Depois de criar a lista de escolha, ela é inserida em um contêiner, como qualquer outro componente — o método add( ) do contêiner é usado com a lista de escolha como argumento.

A listagem 11.6 mostra um applet que continua a tendência de se usar a linguagem Java para reconhecer personagens internacionais. O applet

SelectASpaniard constrói uma lista de escolha de celebridades nascidas na Espanha, da qual um item pode ser selecionado.

DIGITE

Listagem 11.6. O texto completo de selectaspaniard.Java.

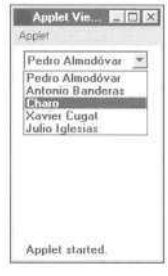
```
1: import java.awt.*;
2:
3: public class SelectASpaniard extends java.applet.Applet {
4:     Choice span = new Choice( );
5:
6:     public void init( ) {
7:         span.addItem("Pedro Almodõvar");
8:         span.addItem("Antonio Banderas");
9:         span.addItem("Charo");
10:            span.addItem("Xavier Cugat");
11:            span.addItem("Julio Iglesias");
12:            add(span);
13:    }
14: }
```

Teste esse applet com a seguinte tag HTML e o resultado será semelhante à Figura 11.8.

```
<applet code="SelectASpaniard.class" height=200 width=150>
</applet>
```

Figura 11.8

O *applet SelectASpaniard.*



The classe Choice possui vários métodos que podem ser usados para controlar uma lista de escolha:

- O método getItem(iní) retorna o texto do item da lista que está na posição do índice especificada pelo argumento inteiro. Assim como acontece nos arrays, o primeiro item de uma lista de escolha está na posição do índice 0, o segundo está na posição 1 etc.
- O método countItems( ) retorna o número de itens presentes na lista. Isso foi desaprovado a partir da Java 2 e substituído por get I temCount ( ), que faz a mesma coisa.

- O método getSelectedIndex( ) retorna a posição do índice do item selecionado no momento na lista.
- O métodogetSelectedItem( ) retorna o texto do item selecionado no momento.
- O método sel*ect(int)* seleciona o item na posição do índice indicada.
- O método *select(String)* seleciona o primeiro item da lista com o texto fornecido.

Campos de texto

Anteriormente, você usou rótulos para texto que não podiam ser modificados pelo usuário. Os campos de texto são usados para criar um componente de texto que pode ser editado. Eles são criados a partir da classe TextField.

Para criar um campo de texto, use um dos seguintes construtores:

- TextField ( ) cria um campo de texto vazio, sem largura especificada.
- TextField (*int*) cria um campo de texto vazio com largura suficiente para apresentar o número especificado de caracteres. Isso foi desaprovado a partir do Java 2 e deve ser substituído por TextField *(String, int)* no caso de applets que não sejam referentes à versão 2.
- TextField *(String)* cria um campo de texto preenchido com o texto especificado e sem nenhuma largura definida.
- TextField *(String, int)* cria um campo de texto como texto e a largura especificados.

O atributo referente à largura de um campo de texto só tem relevância em um gerenciador de layout que não redimensione componentes, como o gerenciador FlowLayout. Você ganhará mais experiência com isso quando trabalhar com gerenciadores de layout, amanhã.

A instrução a seguir cria um campo de texto vazio que possui espaço suficiente para 30 caracteres:

```
TextField name = new TextField(30);
```

A instrução a seguir poderia ser usada se você fornecesse o texto inicial "Puddin N. Tane" no campo de texto name:

```
TextField name = new TextField("Puddin N. Tane", 30);
```

Você também pode criar um campo de texto que oculte com um caractere comum os caracteres que estão sendo digitados. Isso é freqüentemente utilizado em campos Enter Password (Digite a Senha) para ocultar de olhos curiosos uma senha introduzida.

Para definir um caractere de ocultação, o método setEchoCharacter *char)* da classe TextField é usado na linguagem Java 1.02. (Nas versões subseqüentes da linguagem, deve ser usado setEchoChar(char).) Se uma literal

for usada para especificar o caractere, ela deverá ficar entre apóstrofos, como em '*'. A linguagem Java interpreta qualquer literal entre aspas como um objeto String.

O exemplo a seguir cria um campo de texto e define o símbolo de libra (#) como o caractere que será apresentado quando um texto for introduzido no campo:

```
TextField passkey = new TextField(16);
passkey.setEchoCharacter('#');
```

O applet da listagem 11.7 cria vários campos de texto. Rótulos são usados para identificar os campos — normalmente, você usará rótulos desse modo, em vez de fornecer texto dentro do campo de texto explicando seu uso. Um dos campos utiliza um caractere de ocultação para esconder o texto que está sendo digitado.

DIGITE

Listagem 11.7. O texto completo de outofsite.java.

```
1:  import java.awt.*;
2:
3:  public class OutOfSite extends java.applet.Applet {
4:       Label siteLabel = new Label("Site Name: ");
5:       TextField site = new TextField(25);
6:       Label addressLabel = new Label("Site Address: ");
7:       TextField address = new TextField(25);
8:       Label passwordLabel = new Label("Admin Password: ");
9:       TextField password = new TextField(25);
10:
11:        public void init( ) {
12:             add(siteLabel);
13:             add(site);
14:             add(addressLabel);
15:             add(address);
16:             add(passwordLabel);
17:             password.setEchoCharacter('*');
18:             add(password);
19:        }
20: }
```

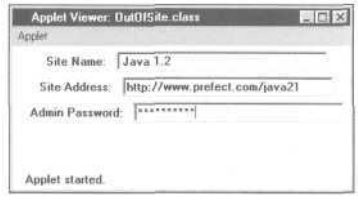
Esse applet pode ser testado usando-se a seguinte tag <APPLET>:

```
<applet code="0ut0fSite.class" width=350 height=125>
</applet>
```

Como esse applet utiliza o gerenciador de layout padrão, o único detalhe que faz os seis componentes aparecerem em três linhas diferentes é a largura da janela. Dependendo da plataforma que você estiver usando, talvez seja preciso ajustar a largura da janela Applet para produzir uma saída comparável à Figura 11.9. (Você aprenderá, na próxima seção, a usar gerenciadores de layout para evitar esse problema.)

Fiaura 11.9

O applet OufOfSite.



A classe TextField possui vários métodos que podem ser usados para controlar um campo de texto:

- O método getText( ) retorna o texto contido no campo.
- O método setText *String)* preenche o campo com o texto indicado.
- O método setEditabl *e(boolean)* determina se o campo pode ser editado. Um argumento false impede que um campo seja editado e true torna possível editar o campo (o que é o padrão).
- O método isEditable( ) retorna um valor booleano indicando se o campo pode ser editado (true) ou não (false).

Áreas de texto

As *áreas de texto,* que são criadas com a classe TextArea, são campos de texto que podem ser editados e que podem manipular mais de uma linha de entrada. As áreas de texto possuem barras de rolagem horizontais e verticais que permitem aos usuários rolar o texto contido no componente.

Para criar uma área de texto, use um dos seguintes construtores:

- TextArea ( ) cria uma área de texto vazia de altura e largura não-especificadas.
- TextArea*[int, int)* cria uma área de texto vazia com o número de linhas indicado (primeiro argumento) e a largura indicada em caracteres (segundo argumento).
- TextArea *(String)* cria uma área de texto contendo o string indicado, de altura e largura não-especificadas.
- TextArea (St*ring, int, int)* cria uma área de texto contendo o string, o número de linhas (primeiro argumento) e a largura em caracteres (segundo argumento), especificados.

O applet exibido na listagem 11.8 apresenta uma área de texto que é preenchida com um string quando o programa começa a ser executado.

DIGITE **Listagem 11.8.** O texto completo de virginia.java.

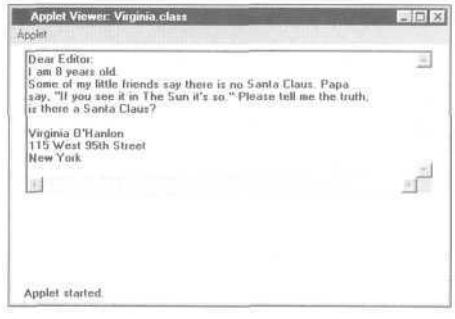
```
1: import java.awt.*;
2:
3: public class Virgínia extends java.applet.Applet {
4:      String letter • "Dear Editor:\n" +
5:          "I am 8 years old.\n" +
6:          "Some of my little friends say there is no Santa Claus." +
7:            " Papa\n" +
8:          "says, ''If you see it in The Sun it's so." Please tell" +
9:            "me the truth,\n" +
10:          "is there a Santa Claus?\n\n" +
11:          "Virginia 0'Hanlon\n" +
12:          "115 West 95th Street\n" +
13:          "New York";
14:     TextArea lt;
15:
16:     public void init( ) {
17:          lt = new TextArea(letter, 10, 50);
18:       add(lt);
19:       }
20: }
```

Teste o applet Vi rgi ni a com a seguinte tag HTML:

```
<applet code="Virginia.class" height=250 width=450>
</applet>
```

A saída do applet carregado com appletviewer aparece na Figura 11.10.

Figura 11.10
O applet *Virginia.*



As áreas de texto e os campos de texto herdam da classe TextComponent; portanto, boa parte do comportamento dos campos de texto pode ser utilizada também em áreas de texto. Você também pode usar os métodos setText( ), getText( ), setEditable( ) e isEditable( ) em áreas de texto, e as áreas de texto também podem utilizar os seguintes métodos:

- O método *insertlext(String, int)* insere o string especificado no índice de caractere indicado pelo inteiro. O índice começa em 0 para

o primeiro caractere e conta daí para cima. Esse método foi desaprovado após a versão 1.02 da linguagem Java e substituído por insert(String, int).

- O método replaceText(String, int, int) substitui o texto que está entre as posições de inteiro dadas pelo string indicado. Esse método também foi desaprovado após o Java 1.02 e substituído por replace(String, int).

Listas de rolagem

As *listas de rolagem*, que são criadas a partir da classe List, são semelhantes às listas de escolha, com duas diferenças significativas:

- Uma lista de rolagem pode ser configurada de modo que mais de um item possa ser selecionado ao mesmo tempo.
- As listas de rolagem não aparecem instantaneamente, quando selecionadas. Em vez disso, vários itens são apresentados, de maneira semelhante a uma área de texto. Se a lista contém mais itens do que pode ser apresentado, uma barra de rolagem é utilizada para a movimentação na lista inteira.

A lista de rolagem é desenvolvida pela criação de um objeto List, seguida da inclusão de itens específicos na lista. A classe List possui os seguintes construtores:

- List( ) cria uma lista de rolagem vazia que permite que apenas um item seja selecionado por vez.
- List(int, boolean) cria uma lista de rolagem com o número indicado de itens visíveis, que pode ser menor do que o número de itens total. O argumento boolean indica se vários itens podem ser selecionados (true) ou não (false).

Depois que um objeto List foi criado, seu método addItem(Stri/7sr) é usado para incluir itens na lista. (Observação de desaprovação: o método preferido para se usar na linguagem Java 2 é add (String)).

O exemplo a seguir cria uma lista e insere dois itens nela:

```
List lackeys = new List( );
lackeys.addItem("Rosencrantz");
lackeys.addItem("Guildenstern");
```

Após a criação e a inclusão de itens na lista de rolagem, ela deve ser inserida em seu contêiner com o método add( ). A listagem 11.9 ilustra a criação de uma lista de rolagem com sete itens.

DIGITE **Listagem 11.9.** O texto completo de hamlet.java.

```
1: import java.awt.*;
2:
3: public class Hamlet extends java.applet.Applet {
4:     List hm = new List(5, true);
5:
6:     public void init( ) {
7:         hm.addItem("Hamlet");
8:         hm.addItem("Claudius");
9:         hm.addItem("Gertrude");
10:        hm.addItem("Polonius");
11:        hm.addItem("Horatio");
12:        hm.addItem("Laertes");
13:        hm.addItem("Ophelia");
14:        add(hm);
15:    }
16: }
```

A saída do applet está ilustrada na Figura 11.11; a seguinte tag de applet é usada:

```
<applet code="Hamlet.class" height=150 width=200>
</applet>
```

Figura 11.11

O applet Hamlet com Claudius, Polonius e Horatio selecionados.

As listas de rolagem possuem vários métodos que funcionam exatamente como os métodos das listas de escolha: getItem(int), countItems( ), getSelectedIndex( ), getSelectedItem( ) e select(int), todos eles funcionam da mesma forma. countItems( ) também possui o mesmo substituto para programas Java 2 programs: getItemCount( ).

Como mais de um item pode ser selecionado em uma lista de rolagem, os seguintes métodos também podem ser usados:

- O método getSelectedIndexes( ) retorna um array de inteiros contendo a posição do índice de cada item selecionado.
- O método getSelectedItems( ) retorna um array de strings contendo o texto de cada item selecionado.

Barras de rolagem e controles deslizantes

As *barras de rolagem* são componentes que permitem que um valor seja selecionado deslizando-se uma caixa entre duas setas. Vários componentes possuem funcionalidade de barra de rolagem incorporada, incluindo áreas de texto e listas de rolagem. A classe Scrollbar é usada para outras barras de rolagem. Uma barra de rolagem pode ser horizontal ou vertical.

Normalmente, as barras de rolagem são criadas através da especificação dos valores mínimo e máximo que podem ser definidos usando-se o componente.

Para criar uma barra de rolagem, você pode usar os seguintes construtores:

- Scroll bar( ) cria uma barra de rolagem vertical com seus valores máximo e mínimo iguais a 0.
- Scroll bar*{int)* cria uma barra de rolagem com os valores mínimo e máximo iguais a 0 e a orientação indicada. Variáveis de classe são usadas para definir a orientação, com o único argumento do método: Scrol 1 - bar.HORIZONTAL ou Scrol 1 bar.VERTICAL.

Você também pode usar um terceiro construtor, com cinco argumentos inteiros: Scrollbar(int, *int, int, int, int).* Os argumentos desse método estão em ordem aqui:

- A orientação é Scrol 1 bar.HORIZONTAL ou Scrol 1 bar.VERTICAL.
- O valor inicial da barra de rolagem, que deve ser igual ou estar entre os valores mínimo e máximo da barra.
- A largura ou altura global da caixa usada para mudar o valor da barra de rolagem. Isso pode ser igual a 0, ao se usar o tamanho padrão.
- O valor mínimo da barra de rolagem.
- O valor máximo.

A listagem 11.10 mostra um applet simples que apresenta uma barra de rolagem. O objeto GridLayout é usado com o método setLayout( ) do applet para fornecer um layout em que uma barra de rolagem preenche seu contêiner inteiro. Você aprenderá sobre os gerenciadores de layout amanhã.

DIGITE

Listagem 11.10. O texto completo de slider.java.

```
1: import java.awt.*;
2:
3: public class Slider extends java.applet.Applet {
4:     GridLayout gl = new GridLayout(1,1);
5:     Scrollbar bar = new Scrolibar(Scrol1 bar.HORIZONTAL,
6:     50,0,1,100);
7:
```

Listagem 11.10. O texto completo de slider.java.

```
8:    public void init( ) {
9:         setLayout(gl);
10:         add(bar);
11:   }
12: }
```

Independentemente dos valores usados para a altura e a largura da janela Applet, a barra de rolagem preencherá a área inteira. A Figura 11.12 foi produzida usando-se a seguinte tag:

```
<applet code="Slider.class" height=20 width=500>
</applet>
```

A classe Scrollbar fornece vários métodos para o gerenciamento dos valores dentro das barras de rolagem:

- O método getVal ue( ) retorna o valor atual da barra de rolagem.
- O método setValue(int) define o valor atual.

Figura 11.12
O applet Slider.

Canvas

Canvas são componentes usados principalmente como um local em uma interface para a apresentação de imagens ou animação. Você pode desenhar outros componentes, como foi feito com a janela Applet em todo este livro, mas os canvas representam os objetos mais simples para esse tipo de uso.

Para usar um canvas, você deve criar uma subclasse de Canvas. Essa subclasse pode manipular qualquer desenho que precise ocorrer no canvas, em seu método pai nt( ).

Uma vez que você tenha criado uma subclasse canvas, ela pode ser usada em um programa através da chamada de seu construtor e da inclusão do novo objeto Canvas em um container.

Isso é demonstrado com o applet Crosshair, apresentado na listagem 11.11. Esse applet desenha um alvo no centro da janela Applet e pode mover o centro imediatamente, se a janela for redimensionada.

DIGITE **Listagem 11.11.** O texto completo de crosshair. java.

```
1: import java.awt.*;
2:
3: public class Crosshair extends java.applet.Applet {
```

Listagem 11.11. Continuação

```
4:      GridLayout g1 = new GridLayout(l,l);
5:      MyCanvas can = new MyCanvas( );
6:
7:      public void init( ) {
8:        setLayout(gl);
9:        add(can);
10:      }
11:
12: }
13:
14: class MyCanvas extends java.awt.Canvas {
15:      public void paint(Graphics g) {
16:           int x = size( ).width / 2;
17:           int y = size( ).height / 2;
18:           g.setColor(Color.black);
19:           g.drawLine(x-10,y,x-2,y);
20:           g.drawLine(x+10,y,x+2,y);
21:           g.drawLine(x,y-10,x,y-2);
22:           g.drawLine(x,y+10,x,y+2);
23:      }
24: }
```

O programa Crosshair pode ser testado com uma janela de qualquer tamanho para o applet. A seguinte tag <APPLET> foi usada para produzir a saída apresentada na Figura 11.13.

```
<applet code="Crosshair.class" height=100 width=100>
</aDDlet>
```

Figura **11.13**
O *applet Crosshair.*

A listagem 11.11 contém dois arquivos de classe. O primeiro, Crosshair, é o próprio applet. O segundo, listado nas linhas 14 a 24, é a classe MyCanvas, que é uma subclasse de Canvas.

A seguir está o que ocorre na classe Crosshai r:

- A linha 4 cria um objeto GridLayout que será definido como gerenciador de layout da classe da linha 8.
- A linha 5 cria um objeto MyCanvas, chamado can, usando a subclasse de Canvas que é criada nas linhas 14 a 24.

- A linha 9 adiciona can à janela Applet. Como um gerenciador de layout de grade está em vigor, o canvas se expande para preencher a janela inteira.

A maior parte do trabalho nesse projeto é realizado em MyCanvas, a classe auxiliar. O seguinte está ocorrendo na classe:

- As linhas 16 e 17 determinam o ponto central da janela Applet. Isso é feito dinamicamente, sempre que o canvas é repintado. As variáveis size( ) .width e size( ) .height contêm a largura e a altura do canvas, e elas podem ser divididas por 2 para se determinar o ponto central. Se você não estiver escrevendo um applet 1.02, as variáveis getSize( ) .width e getSize( ). height deverão, ser usadas, para evitar avisos de desaprovação, quando o programa for compilado.
- A linha 18 define a cor preta como correntemente ativa para operações de desenho. Observe que esse método é chamado no objeto Graphics e não o canvas em si. O objeto Graphics enviado para o método paint( ) manipula todas as operações de desenho que ocorrem no objeto.
- As linhas 19 a 22 utilizam as coordenadas de centro x,y para desenhar quatro linhas em forma de alvo em torno do centro. Cada linha tem 8 pixels de comprimento e terminam a 2 pixels do centro.

Resumo

Agora, você sabe pintar uma interface com o usuário na janela de um applet Java usando a paleta padrão da linguagem — os componentes do Abstract Windowing Toolkit.

O kit de ferramentas inclui classes para muitos dos botões, barras, listas e campos que você esperaria ver em um programa. Esses componentes são utilizados criando-se uma instância de sua classe e pela sua inserção em um contêiner — como uma janela Applet —, usando-se o método add( ) do contêiner.

Hoje, você aprendeu alguma coisa sobre função, desenvolvendo componentes e incluindo-os em um programa. Durante os próximos dois dias, você aprenderá mais a respeito de dois itens que são necessários para tornar uma interface gráfica útil:

Forma: como organizar os componentes para formar uma interface completa.

Feedback: como receber entrada de um usuário através desses componentes.

Perauntas e respostas

P **Com todos os métodos desaprovados que fazem parte do Abstract Windowing Toolkit a partir da versão 2 da linguagem Java, por que devo escrever applets em Java 1.02?**

R Normalmente, você não precisa aprender nada a respeito das versões passadas da linguagem Java, ao utilizar Java 2. No entanto, os principais projetistas de navegador têm sido extremamente lentos em introduzir o suporte às versões da linguagem além da 1.02, e parecia, quando da redação deste livro, que a Microsoft nunca iria oferecer suporte integral à Java 1.1 e muito menos à Java 2. Por isso, a linguagem Java 1.02 continua sendo o padrão ao se escrever applets. A Sun está trabalhando em um modo para que os projetistas de applet especifiquem seus próprios ambientes de runtime com um applet, o que tornaria possível escrever applets Java 2 e garantir que as pessoas que utilizam navegadores compatíveis com Java pudessem executar o programa.

P **Minha ferramenta de desenvolvimento Java tem um modo para se fazer o design da interface de um programa visualmente — posso arrastar e soltar botões e outros componentes e organizá-los com um mouse. Eu preciso aprender o Abstract Windowing Toolkit?**

R Se você está contente com os resultados que está obtendo e confiante em sua capacidade de usar a interface em um programa funcional, o Abstract Windowing Toolkit não é uma necessidade. Entretanto, a utilização do AWT para criar uma interface gráfica com o usuário que funcione é um dos maiores projetos deste livro. Ele traz os conhecimentos dos quais você tirará proveito em outras áreas da linguagem Java.

SEMANA 2

Dia 12

Organizando componentes em uma interface com o usuário

Se o design de uma interface gráfica com o usuário é comparável à pintura, atualmente você pode produzir apenas um tipo de arte: expressionismo abstrato. Você pode colocar componentes em uma interface, mas não possui muito controle sobre onde eles ficam.

Para impor algum tipo de forma em uma interface projetada com o Abstract Windowing Toolkit, você deve usar um conjunto de classes chamado *gerenciadores de layout.*

Hoje, você aprenderá a usar cinco gerenciadores de layout para organizar componentes em uma interface. Você tirará proveito da flexibilidade do kit de ferramentas de janelas da linguagem Java, que foi projetado para ser apresentado em diferentes plataformas que ofereçam suporte à linguagem.

No caso de uma organização não se adequar muito ao que você tinha em mente para um programa, também será visto como colocar diferentes gerenciadores de layout para funcionar na mesma interface.

Começaremos com os gerenciadores de layout básicos.

Lavout de interface básico

Como você aprendeu ontem, uma interface gráfica com o usuário projetada com o Abstract Windowing Toolkit é algo muito fluido. Redimensionar uma

janela pode estragar sua interface, quando componentes mudam para lugares em um contêiner que podem não ter sido o planejado.

Essa fluidez é uma necessidade. A linguagem Java é implementada em muitas plataformas diferentes e existem diferenças sutis no modo como cada uma apresenta itens como botões, barras de rolagem e coisas assim.

No caso de linguagens de programação, como o Microsoft Visual Basic, a posição de um componente em uma janela é definida precisamente por suas coordenadas x,y. Algumas ferramentas de desenvolvimento Java permitem um controle semelhante sobre uma interface, com o uso de suas próprias classes de janelas.

Ao usar o Abstract Windowing Toolkit, um programador ganha mais controle sobre o layout de uma interface, usando gerenciadores de layout.

Fazendo o layout de uma interface

Um gerenciador de layout determina como os componentes serão organizados ao serem inseridos em um contêmer.

O gerenciador de layout padrão é a classe FlowLayout. Essa classe permite que os componentes fluam da esquerda para a direita, na ordem em que eles são inseridos em um contêiner. Quando não houver mais espaço, uma nova linha de componentes começará imediatamente abaixo da primeira e a ordem da esquerda para a direita continuará.

O AWT inclui cinco gerenciadores de layout básicos: FlowLayout, GridLayout, BorderLayout, CardLayout e GridBagLayout. Para criar um gerenciador de layout para um contêiner, uma instância do contêiner é gerada usando-se uma instrução como a seguinte:

```
FlowLayout flo = new FlowLayout( );
```

Após criar um gerenciador de layout, você o transforma no gerenciador de layout de um contêiner usando o método setLayout( ) do contêiner. O gerenciador de layout deve ser estabelecido antes que qualquer componente seja incluído no contêiner. Se nenhum gerenciador de layout estiver especificado, será usado o layout de fluxo.

As instruções a seguir representam o ponto de partida de um applet que cria um gerenciador de layout e utiliza setLayout( ) para controlar a organização de todos os componentes que serão inseridos na janela Applet:

```
public class Starter extends java.applet.Applet {
    FlowLayout lm = new FlowLayout( );
    public void init( ) {
      setLayout(lm);
    }
}
```

Depois que o gerenciador de layout estiver definido, você pode começar a incluir componentes no container que ele gerencia. Para alguns dos geren-

ciadores de layout, como o FlowLayout, a ordem em que os componentes são inseridos é significativa. Você aprenderá mais nas seções subseqüentes de hoje, quando trabalhar com cada um dos gerenciadores.

Layout de fluxo

A classe FlowLayout representa o mais simples dos gerenciadores de layout. Ela dispõe os componentes de maneira semelhante ao modo como as palavras são dispostas em uma página — da esquerda para a direita, até que não haja mais espaço, e depois na próxima linha.

Por padrão, os componentes de cada linha serão centralizados quando você usar o construtor FlowLayout ( ) sem argumentos. Se você quer que os componentes fiquem alinhados ao longo da margem esquerda ou direita do contêiner, a variável de classe Fl owLayout. LEFT ou FlowLayout. RIGHT deve ser o único argumento do construtor, como na seguinte instrução:

```
FlowLayout righty = new FlowLayout(FlowLayout.RIGHT);
```

A variável de classe FlowLayout. CENTER é usada para especificar componentes centralizados.

O applet da listagem 12.1 apresenta seis botões, organizados pelo gerenciador de layout de fluxo. Como a variável de classe Fl owLayout. LEFT foi usada no construtor Fl owLayout ( ), os componentes ficam alinhados ao longo do lado esquerdo da janela do applet.

DIGITE

Listagem 12.1. O texto completo de alphabet.java.

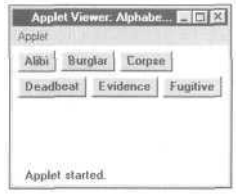
```
1: import java.awt.*;
2:
3: public class Alphabet extends java.applet.Applet {
4:    Button a = new Button("Alibi");
5:    Button b = new Button("Burglar");
6:    Button c • new Button("Corpse");
7:    Button d » new Button("Deadbeat");
8:    Button e = new Button("Evidence");
9:    Button f = new Button("Fugitive");
10:    FlowLayout 1m » new FlowLayout(FlowLayout.LEFT);
11:
12:    public void init( ) {
13:       setLayout(lm);
14:       add(a);
15:       add(b);
16:       add(c);
17:       add(d);
18:       add(e);
19:       add(f);
20:    }
21; }
```

A seguinte tag <APPLET> foi usada para produzir a saída apresentada na Figura 12.1 com o appletviewer:

```
<applet code="Alphabet.class" height=120 width=220>
</aDDlet>
```

Figura 12.1

Seis botões *organizados no layout de fluxo.*



No applet Alphabet, o gerenciador de layout de fluxo coloca um intervalo de três pixels entre cada componente de uma linha e três pixels entre cada linha. Você também pode mudar o espaço horizontal e vertical entre os componentes com alguns argumentos extras no construtor FlowLayout( ).

O construtor FlowLayout(int, *int, int)* recebe os três argumentos a seguir, em ordem:

- O alinhamento, que deve ser FlowLayout.CENTER, FlowLayout.LEFT ou FlowLayout.RIGHT.
- A distância horizontal entre os componentes, em pixels.
- A distância vertical, em pixels.

O construtor a seguir cria um gerenciador de layout de fluxo com componentes centralizados, uma distância horizontal de 30 pixels e uma distância vertical de 10:

```
FlowLayout flo = new FlowLayout(FlowLayout.CENTER, 30, 10);
```

Layout de qrade

O gerenciador de layout de grade organiza os componentes em uma grade de linhas e colunas. Os componentes são incluídos primeiramente na linha superior da grade, começando com a célula mais à esquerda e continuando à direita na grade. Quando todas as células da linha superior estiverem preenchidas, o próximo componente será inserido na célula mais à esquerda da segunda linha da grade — se houver uma segunda linha — e assim por diante.

Os layouts de grade são criados com a classe GridLayout. Dois argumentos são enviados para o construtor GridLayout — o número de linhas na grade e o número de colunas. A instrução a seguir cria um gerenciador de layout de grade com 10 linhas e 3 colunas:

```
GridLayout gr = new GridLayout(10,3);
```

Assim como acontece no layout de fluxo, você pode especificar um intervalo vertical e horizontal entre os componentes com dois argumentos

extras. A instrução a seguir cria um layout de grade com 10 linhas, 3 colunas, uma distância horizontal de 5 pixels e uma distância vertical de 8 pixels:

```
GridLayout gr2 = new GridLayout(10,3,5,8);
```

O distanciamento padrão entre os componentes no layout de grade é de 0 pixels nas direções vertical e horizontal.

A listagem 12.2 contém um applet que cria uma grade com três linhas, três colunas e um intervalo de 10 pixels entre os componentes nas direções vertical e horizontal.

DIGITE

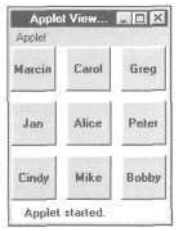
Listagem 12.2. O texto completo de bunch.java.

```
1: import java.awt.*;
2:
3: public class Bunch extends java.applet.Applet {
4:       GridLayout family = new GridLayout(3,3,10,10);
5:       Button mareia = new Button("Mareia");
6:       Button carol = new Button("Carol");
7:       Button greg = new Button("Greg");
8:       Button jan = new Button("Jan");
9:       Button alice • new Button("Alice");
10:      Button peter = new Button("Peter");
11:      Button cindy = new Button("Cindy");
12:      Button mike = new Button("Mike");
13:      Button bobby = new Button("Bobby");
14:
15:      public void init( ) {
16:            setLayout(family);
17:            add(marcia);
18:            add(carol);
19:            add(greg);
20:            add(jan);
21:            add(alice);
22:            add(peter);
23:            add(cindy);
24:            add (mike);
25:            add(bobby);
26:       }
27: }
```

A Figura 12.2 mostra esse applet em uma página com a seguinte tag <APPLET>:

```
<applet code="Bunch.class" height=160 width=160>
</applet>
```

Figura 12.2
Nove *botões organizados* no *layout de grade de 3x3.*

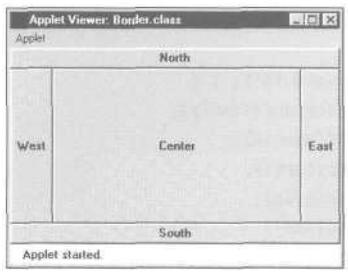


Um detalhe a ser observado em relação aos botões na Figura 12.2 é que eles se expandiram para preencher o espaço disponível em cada célula. Essa é uma diferença importante entre o layout de fluxo e alguns dos outros gerenciadores de layout. No layout de grade, um componente sempre ocupará o espaço inteiro de uma célula. Se você carregar o applet Bunch usando a ferramenta appletviewer, poderá ver que os botões mudam de tamanho, quando a janela Applet é redimensionada.

Layout de borda

Os *layouts de borda,* que são criados usando-se a classe BorderLayout, dividem um contêiner em cinco seções: North (norte), South (sul), East (leste), West (oeste) e Center (centro). As cinco áreas da Figura 12.3 mostram como essas seções são organizadas.

Figura 12.3
Organização de componentes no layout de borda.



No layout de borda, os componentes dos quatro pontos cardeais ocuparão o máximo espaço de que necessitarem — o centro ocupará o espaço restante. Normalmente, isso resultará em uma organização com um grande componente central e quatro pequenos componentes em torno dele.

Um layout de borda é criado com os construtores BorderLayout ( ) ou BorderLayout *(int, int).* O primeiro construtor cria um layout de borda sem intervalo entre os componentes. O segundo construtor especifica o intervalo horizontal e o intervalo vertical, respectivamente.

Depois que você cria um layout de borda e o define como gerenciador de layout de um contêiner, os componentes são incluídos usando-se uma chamada ao método add( ) diferente da que foi vista anteriormente:

```
add(String, component)
```

O segundo argumento desse método é o componente que deve ser incluído no contêiner.

O primeiro argumento é um string indicando a qual parte do layout de borda se deve atribuir o componente. Existem cinco valores possíveis: "North", "South", "East", "West" ou "Center".

A instrução a seguir insere um botão chamado qui tButton na parte norte de um layout de borda:

```
add("North", quitButton);
```

A listagem 12.3 contém o applet utilizado para produzir a Figura 12.3.

Listagem 12.3. O texto completo de border.java.

```
1: import java.awt.*;
2:
3: public class Border extends java.applet.Applet {
4:       BorderLayout b = new BorderLayout( );
5:       Button north = new Button("North");
6:       Button south • new Button("South");
7:       Button east = new Button("East");
8:       Button west - new Button("west");
9:       Button center - new Button("Center");
10:
11:       public void init( ) {
12:            setLayout(b);
13:            add("North", north);
14:        add("South"        south);
15:            add("East", east);
16:            add("West", west);
17:            add("Center", center);
18:       }
19: }
```

A seguinte tag <APPLET> foi usada:

```
<applet code="Border.class" height=120 width=120>
</applet>
```

Misturando aerenciadores de layout

Neste ponto, você pode estar se perguntando como os gerenciadores de layout do Abstract Windowing Toolkit funcionarão com o tipo de interface gráfica com o usuário que você deseja projetar. Escolher um gerenciador de layout é algo difícil : um é muito quadrado, outro é muito desorganizado e o outro é muito estranho.

Para encontrar o layout perfeito, freqüentemente você precisa combinar mais de um gerenciador na mesma interface.

Isso é feito inserindo-se contêineres em um contêiner principal, como uma janela Applet, e fornecendo-se a cada um desses contêineres menores seus próprios gerenciadores de layout.

O contêiner a ser usado para esses contêineres menores é o painel, que é criado a partir da classe Panei. Os *painéis* são contêineres utilizados para agrupar componentes. Existem dois detalhes a serem lembrados ao se trabalhar com painéis:

- O painel é preenchido com componentes antes de ser colocado em um contêiner maior.
- O painel possui seu próprio gerenciador de layout.

Os painéis são criados com uma simples chamada ao construtor da classe Panei, como se vê no exemplo a seguir:

```
Panei pane = new Panel( );
```

O método de layout é definido para um painel chamando-se o método setLayout( ) nesse painel. Isso funciona igual ao método setLayout( ) que você utilizou para a janela de applet — Applet e Panei são subclasses da classe Container e elas herdam o comportamento de gerenciamento de layout inalterado dessa superclasse.

As instruções a seguir criam um gerenciador de layout e o aplicam a um objeto Panei chamado pane:

```
BorderLayout bo = new Borderl_ayout( );
pane.setLayout(bo);
```

Os componentes são incluídos em um painel através da chamada ao método add( ) do painel, que funciona para os painéis exatamente como para outros contêineres, como os applets.

A instrução a seguir insere uma área de texto chamada dialogue em um objeto Panei chamado pane:

```
pane.add(dialogue);
```

Você verá diversos exemplos da utilização de painel no restante dos exemplos de programa de hoje.

Layout de interface avançado

Além dos três gerenciadores de layout sobre os quais você aprendeu —layout de fluxo, layout de grade e layout de borda —, o Abstract Windowing Toolkit inclui dois gerenciadores de layout mais sofisticados. Os gerenciadores de layout de carta e de grade de conteúdo também podem ser misturados com os outros gerenciadores, aninhando-se um contêiner dentro de outro.

Layout de carta

Os layouts de carta diferem dos outros layouts porque eles ocultam alguns componentes. Um *layout de carta* é um grupo de contêineres ou componentes que são exibidos um por vez, da mesma maneira que um jogador de vinte-e-um revela uma carta do monte por vez. Cada contêiner do grupo é chamado de *carta.*

Se você já usou software como o HyperCard no Macintosh ou uma caixa de diálogo com guias, como a parte System Properties do Control Panei do Windows 95, então já trabalhou com um programa que utiliza layout de carta.

O modo normal de usar um layout de carta é utilizar um painel para cada carta. Primeiramente, os componentes são inseridos nos painéis e, em seguida, os painéis são inseridos no contêiner que está definido para usar layout de carta.

Um layout de carta é criado a partir da classe CardLayout com uma simples chamada de construtor:

```
CardLayout cc = new CardLayout( );
```

O método setLayout( ) é usado para tornar este o gerenciador de layout do contêiner, como na seguinte instrução:

```
setLayout(cc);
```

Após definir um contêiner para usar o gerenciador de layout de carta, você deve utilizar uma chamada do método add ( ) ligeiramente diferente, para incluir cartas no layout.

O método a ser usado é *add String, container).* O segundo argumento especifica o contêiner ou componente que é a carta. Se for um contêiner, todos os componentes deverão ter sido incluídos nele, antes que a carta seja inserida.

O primeiro argumento do método add( ) é um string que representa o nome da carta. Pode ser qualquer nome que você queira para denominar a carta. Talvez você queira numerar as cartas de algum modo e utilizar o número no nome, como em "Card 1", "Card 2", "Card 3" etc.

A instrução a seguir insere um painel chamado options em um contêiner e dá a essa carta o nome "Options Card":

```
add("Options Card", options);
```

Após ter adicionado uma carta ao contêiner principal de um programa, como uma janela Applet, você pode usar o método show( ) de seu gerenciador de layout de carta para apresentar uma carta. O método show( ) recebe dois argumentos:

- O contêiner em que todas as cartas foram inseridas. Se o contêiner for o applet, você poderá usar a palavra-chave thi s dentro do applet, para esse argumento.
- O nome que foi dado à carta.

A instrução a seguir chama o método show ( ) de um gerenciador de layout de carta denominado cc:

```
cc.show(this, "Fact Card");
```

A palavra-chave this se refere à classe em que essa instrução está aparecendo e "Fact Card" é o nome da carta a ser revelada. Quando uma carta for apresentada, a carta mostrada anteriormente será ocultada. Em um layout de carta, apenas uma carta pode ser vista por vez.

Em um programa que usa o gerenciador de layout de carta, uma mudança de carta normalmente será disparada pela ação de um usuário. Por exemplo, em um programa que apresenta endereços de correio em diferentes cartas, o usuário poderia selecionar uma carta para exibição, escolhendo um item em uma lista de rolagem. Como alternativa, o applet da listagem 12.4 usa animação com threads para mudar do painel de uma carta para a seguinte.

DIGITE **Listagem 12.4.** O texto completo de burmashave.java.

```
1: import java.awt.*;
2:
3: public class BurmaShave extends java.applet.Applet
4:     implements Runnable {
5:
6:     CardLayout card = new CardLayout( );
7:     Label[ ] lab = new Label [6];
8:     int current = 0;
9:     Thread runner;
10:
11:     public void start( ) {
12:         if (runner == null) {
13:         runner = new Thread(this);
14:         runner.start( );
15:       }
16: }
17:
18:     public void stop( ) {
19:         runner = null;
20:    }
21:
22:     public void init( ) {
23:         lab[0] - new Label("Grandpa's beard");
24:         lab[l] = new Label("was stiff and coarse.");
25:         lab[2] = new Label ("And that's what caused");
26:         lab[3] = new Label("His fifth");
27:         lab[4] = new Label("Divorce.");
28:         lab[5] = new Label("Burma Shave.");
29:         setLayout(card);
30:         for (int i = 0; i < 6; i++)
31:           add("Card " + i, lab[i]);
```

Listagem 12.4. Continuação

```
32:     }
33:
34:     public void run( ) {
35:         Thread thisThread = Thread.currentThread( );
36:         while (runner == thisThread) {
37:             card.show(this, "Card " + current);
38:             current++;
39:             if (current > 5)
40:               current = 0;
41:             repaint( );
42:             try {
43:                 Thread.sleep(5000);
44:             } catch (InterruptedException e) { }
45:         }
46:     }
47: }
```

A tag <APPLET> a seguir foi usada para produzir a saída ilustrada na Figura 12.4.

```
<applet code="BurmaShave.class" height=80 width=160>
</applet>
```

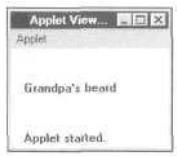
O applet BurmaShave apresenta um layout de carta com seis cartas. Cada uma delas é um componente de rótulo e a animação é obtida circulando-se pelas seis cartas.

Algumas observações sobre o applet:

- Linha 7 — O array 1 ab é criado para conter seis rótulos.
- Linha 8 — A variável current é definida. Ela é usada para controlar a carta atual a ser exibida.
- Linhas 23 a 28 — Os seis objetos Label são criados e cada um é intitulado com uma linha de um slogan do anúncio de beira de estrada de Burma Shave.
- Linha 29 — O gerenciador de layout do applet é definido como 1 ayout de carta.
- Linhas 30 e 31 — Usando-se um loop for, todos os seis rótulos do array 1 ab são incluídos como cartas na janela Applet. Cada carta recebe um nome que começa com o texto "Card", seguido de um espaço e um número de 0 a 5, como em "Card 0".
- Linha 37 — O método show( ) da classe CardLayout é usado para mostrar a carta atual. O nome da carta é o texto "Card", seguido de um espaço e do valor da variável current.

- Linha 38 — O valor de current é incrementado em 1.
- Linhas 39 e 40 — A variável current é novamente definida como 0, caso tenha ultrapassado o valor 5.

Figura 12.4
Uma caria exibida em um layout de várias cartas.



Invout de *arada* de conteúdo

O último dos gerenciadores de layout disponíveis no AWT é o layout de grade de conteúdo, que é uma extensão do gerenciador de layout de grade, diferindo deste com relação ao seguinte:

- Um componente pode ocupar mais de uma célula na grade.
- As proporções entre as diferentes linhas e colunas não precisam ser iguais.
- Os componentes dentro das células da grade podem ser organizados de diferentes maneiras.

Para criar um layout de grade de conteúdo, você usa a classe GridBag Layout e uma classe auxiliar chamada GridBagConstraints. GridBagLayout é o gerenciador de layout e GridBagConstraints é usada para definir as propriedades de cada componente a ser colocado na célula — seu posicionamento, dimensões, alinhamento etc. O relacionamento entre a grade de conteúdo, as restrições e cada componente define o layout global.

Em sua forma mais geral, a criação de um layout de grade de conteúdo envolve as seguintes etapas:

1. Criar um objeto GridBagLayout e defini-lo como o gerenciador de layout atual, como você faria com qualquer outro gerenciador de layout.
2. Criar uma nova instância de Gri dBagConstrai nts.
3. Estabelecer as restrições de um componente.
4. Informar o gerenciador de layout a respeito do componente e de suas restrições.
5. Inserir o componente no contêiner.

O exemplo a seguir insere um botão em um contêiner implementando o layout de grade de conteúdo. (Não se preocupe com os diversos valores das restrições; eles serão abordados posteriormente nesta seção.)

```
// define o layout
GridBagLayout gridbag = new GridBagLayout( );
GridBagConstraints constraints = new GridBagConstraints( );
setLayout(gridbag);

// define as restrições do botão
Button btn = new Button("Save");
constraints.gridx = 0;
constraints.gridy = 0;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.weightx = 30;
constraints.weighty = 30;
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;

// vincula as restrições ao layout, adiciona o botão
gridbag.setConstraints(btn, constraints);
add(b);
```

Como você pode ver a partir desse exemplo, é preciso definir todas as restrições de cada componente que se deseja adicionar ao painel. Dadas as numerosas restrições, é bom ter um plano e tratar de cada tipo de restrições por vez.

Etapa um: Projete a grade

O primeiro lugar para se começar o layout de grade de conteúdo é no papel. Esboçar o design de sua interface com o usuário antecipadamente — antes de escrever qualquer linha de código — ajudará muito a longo prazo, na tentativa de se descobrir onde tudo vai ficar. Coloque seu editor de lado por uns instantes, pegue uma folha de papel e um lápis e crie a grade.

A Figura 12.5 mostra o layout do painel que você vai criar neste exemplo. A Figura 12.6 mostra o mesmo layout com uma grade imposta sobre ele. Seu layout terá uma grade semelhante a essa, com linhas e colunas formando as células individuais.

Ao desenhar sua grade, lembre-se de que cada componente deve ter sua própria célula. Você não pode colocar mais de um componente na mesma célula. Entretanto, o inverso não é verdadeiro; um componente pode abranger várias células na direção x ou y (como acontece no botão OK, na linha inferior, que abrange duas colunas). Na Figura 12.6, observe que os rótulos e os campos de texto possuem suas próprias grades e que o botão abrange duas células de coluna.

figura 12.5
Um layout de grade de conteúdo.

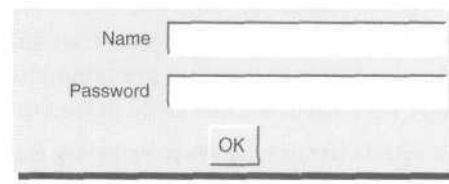
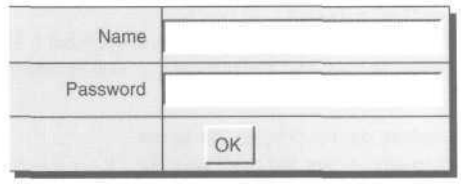


Figura 12.6

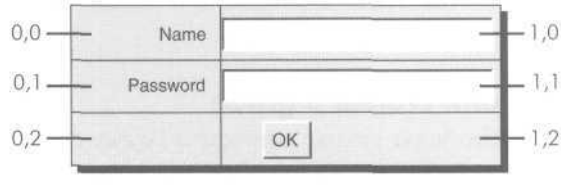
O *layout de grade de conteúdo da Figura* 12.5, com a *grade* imposfa.



Identifique as células com suas coordenadas x e y, enquanto você ainda estiver trabalhando no papel; isso o ajudará depois. Não se trata de coordenadas em pixel, mas de coordenadas de célula. A célula do canto superior esquerdo corresponde a 0,0. A célula seguinte, à direita dela, na linha superior, é 1,0. A célula à direita desta corresponde a 2,0. Passando para a linha seguinte, a célula mais à esquerda corresponde a 0,1 a célula seguinte da linha tem coordenadas iguais a 1,1 e assim por diante. Identifique suas células no papel com esses números; você precisará deles posteriormente, quando produzir o código desse exemplo. A Figura 12.7 mostra os números de cada uma das células desse exemplo.

Figura 12.7

O l*ayout de grade de conteúdo da Figura* 12.5, com coordenadas de *célula.*



Etapa dois: Crie a grade

Agora, retorne à linguagem Java e comece a implementar o layout que acabou de desenhar no papel. Inicialmente, você vai focalizar exclusivamente o layout — a obtenção da grade e das proporções corretas. Para isso, pode ser mais fácil usar botões como marcadores para os elementos reais no layout. Eles são fáceis de criar e definem claramente o espaço que um componente ocupará no gerenciador de layout (ou nos gerenciadores) que estiver sendo usado. Quando tudo estiver estabelecido corretamente, os botões poderão ser substituídos pelos elementos certos.

Para diminuir a quantidade de digitação a ser feita para definir todas essas restrições, você pode começar definindo um método auxiliar que receba diversos valores e defina as restrições para esses valores. O método buildConstraints( ) recebe sete argumentos: um objeto GridBagConstraints e seis inteiros representando as variáveis de instância GridBagConstraints gridx, gridy, gridwidth, gridheight, weightx e weighty. Posteriormente, você aprenderá o que elas fazem realmente; por enquanto, aqui está o código do método auxiliar que você vai usar mais tarde neste exemplo:

```
void buildConstraints(GriclBagConstra.ints gbc, int gx, int gy,
    int gw, int gh, int wx, int wy) {
```

```
    gbc.gridx = gx;
    gbc.gridy = gy;
    gbc.gridwidth = gw;
    gbc.gridheight » gh;
    gbc.weightx = wx;
    gbc.weighty = wy;
```

Agora, passemos ao método init( ), onde todo o layout realmente ocorre. A seguir, há a definição de método básica, onde você vai definir o Gri dBagLaout como gerenciador de layout inicial e criar um objeto constrai nts (uma instância de GridBagConstraints):

```
public void init( ) {
    GridBagLayout gridbag = new GridBagLayout( );
    GridBagConstraints constraints = new GridBagConstraints( );
    setLayout(gridbag);

    constraints.fill = GridBagConstraints.BOTH;
}
```

Mais uma pequena nota de explicação: a última linha, que define o valor de constraints.fill, será removida (e explicada) posteriormente. Ela está presente para que os componentes preencham a célula inteira em que estão contidos, o que o ajuda a ver o que está acontecendo. Insira-a por enquanto; posteriormente, você terá uma idéia mais clara de para que ela serve.

Agora, adicione ao layout os botões que definem marcadores. (Lembre-se de que, no momento, você está se concentrando na organização básica da grade; portanto, vai usar os botões como marcadores para os elementos reais da interface com o usuário, que vão ser inseridos posteriormente.) Comece com um único botão, para que você possa ter uma idéia da definição de suas restrições. Esse código ficará no método init( ), imediatamente após a linha setLayout:

```
// Rótulo de nome
  buildConstraints(constraints, 0, 0, 1, 1, 100, 100);
  Button 1 abel 1 = new Button("Name:");
  gridbag.setConstraints(label1, constraints);
  add(labell);
```

Essas quatro linhas definem as restrições de um objeto, criam um novo botão, vinculam as restrições ao botão e, em seguida, o adicionam ao painel. Observe que as restrições de um componente são armazenadas no objeto GridBagConstraints, de modo que o componente nem mesmo precisa existir para se definir suas restrições.

Agora, você pode entrar em detalhes: quais são exatamente os valores das restrições inseridas no método auxiliar bui 1 dConstrai nts ( ) ?

Os dois primeiros argumentos inteiros são os valores gridx e gridy das restrições. Eles são as coordenadas da célula que contém esse componente. Lembra-se de como você escreveu esses componentes no papel na etapa um?

Com as células provisoriamente numeradas no papel, basta inserir os valores corretos. Observe que, se você tiver um componente que abranja várias células, as coordenadas serão correspondentes às da célula do canto superior esquerdo.

Esse botão está no canto superior esquerdo; portanto, seus argumentos gridx e gridy (os dois primeiros argumentos de buildConstraints( )) são 0 e 0, respectivamente.

Os dois argumentos inteiros seguintes são gridwidth e gridheight. Eles não são larguras e alturas em pixel das células; em vez disso, representam o número de células que esse componente abrange: gridwidth para as colunas e gridheight para as linhas. Aqui, esse componente abrange apenas uma célula; portanto, os valores de ambos são 1.

Os dois últimos argumentos inteiros correspondem a weightx e weighty. Eles são usados para definir as proporções das linhas e colunas — isto é, sua largura e profundidade. Os pesos podem se tornar muito confusos, de modo que, por enquanto, defina os dois valores como 100. Os pesos serão tratados na etapa três.

Depois que as restrições estiverem definidas, você pode vinculá-las a um objeto usando o método setConstraints( ). setConstraints( ), que é um método definido em GridBagLayout, recebe dois argumentos: o componente (aqui um botão) e as restrições desse botão. Finalmente, você pode inserir o botão no painel.

Após ter definido e atribuído as restrições a um componente, você pode reutilizar esse objeto GridBagConstraints para definir as restrições do próximo objeto. Portanto, você duplica essas quatro linhas para cada componente da grade, com diferentes valores para o método bui1 dConstrai nts ( ). Para economizar espaço, serão apresentados apenas os métodos bui 1 dConstrai nts ( ) das quatro últimas células.

A segunda célula a ser incluída é a que conterá a caixa de texto para o nome. Suas coordenadas de célula são 1,0 (segunda coluna, primeira linha); ela também abrange apenas uma célula e os dois pesos (por enquanto) são 100:

```
buildConstraints(constraints, 1, 0, 1, 1, 100, 100);
```

Os dois próximos componentes, que serão um rótulo e um campo de texto, são praticamente os mesmos que os dois anteriores; a única diferença está em suas coordenadas de célula. O rótulo correspondente à senha (Password) está nas coordenadas 0,1 (primeira coluna, segunda linha) e o respectivo campo de texto está nas coordenadas 1,1 (segunda coluna, segunda linha):

```
buildConstraints(constraints, 0, 1, 1, 1, 100, 100);
buildConstraints(constraints, 1, 1, 1, 1, 100, 100);
```

Finalmente, você precisa do botão OK, que é um componente que abrange duas células na linha inferior do painel. Aqui, as coordenadas de célula representam a célula superior esquerda, onde a ocupação começa (0,2). Nesse caso, ao contrário dos componentes anteriores, você definirá gridwidth e

gridhei ght como algo diferente de 1, pois essa célula abrange várias colunas.O valor de gridweight é 2 (ele abrange duas células) e o valor de gri dheight é 1 (ele abrange apenas uma linha):

```
buildConstraints(coristraints, 0, 2, 2, 1, 100, 100);
```

Você definiu as restrições de posicionamento de todos os componentes que serão inseridos no layout de grade. Também precisa atribuir as restrições de cada componente ao gerenciador de layout e depois inserir cada componente no painel. A Figura 12.8 mostra o resultado até este ponto. Note que aqui você não está preocupado com as proporções exatas ou com o fato de tudo estar alinhado. Neste ponto, o que você deve fazer é garantir que a grade esteja funcionando, que tenha o número correto de linhas e colunas, que as abrangências estejam corretas e que nada estranho vá ocorrer (células no lugar errado, células se sobrepondo, esse tipo de coisa).

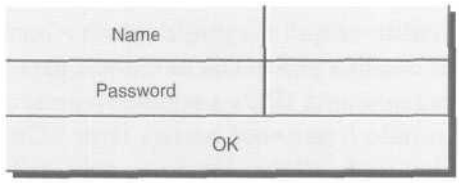


Figura 12.8
O layout de grade de conteúdo, primeiro passo.

Etapa três: Determine as proporções

A próxima etapa é determinar as proporções das linhas e colunas em relação às outras linhas e colunas. Por exemplo, nesse caso, você vai querer que os rótulos (nome e senha) ocupem menos espaço do que as caixas de texto. Talvez você queira que o botão OK, na parte inferior, tenha apenas metade da altura das duas caixas de texto que estão acima dele. Você organiza as proporções das células dentro de seu layout usando as restrições weightx e weighty.

O modo mais fácil de pensar em wei ghtx e wei ghty é que seus valores são porcentagens da largura e da altura total do painel, ou 0, se o peso ou a altura foi definido por alguma outra célula. Os valores de wei ghtx e wei ghty para todos os seus componentes, portanto, devem somar 100.

Na verdade, os valores weightx e weighty não são porcentagens; eles são simplesmente proporções — podem ter qualquer valor. Quando as proporções são calculadas, todos os valores em uma direção são somados para que cada valor específico seja proporcional a esse total. Para tornar esse processo mais fácil de entender, veja os pesos como porcentagens e certifique-se de que seu somatório seja igual a 100, para garantir que tudo funcione corretamente.

Quais células recebem valores e quais células recebem 0? As células que abrangem várias linhas ou colunas sempre devem ser 0 na direção em que se expandem. Fora isso, a decisão é uma simples questão de escolher uma célula para ter um valor e, então, todas as outras células dessa linhas ou coluna deverão ser 0.

Observe as cinco chamadas feitas a bui 1 dConstraints ( ) na etapa anterior:

```
buildConstraints(constraints, 0, 0, 1, 1, 100, 100); //nome
buildConstraints(constraints, 1, 0, 1, 1, 100, 100); //texto do nome
buildConstraints(constraints, 0, 1, 1, 1, 100, 100); //senha
buildConstraints(constraints, 1, 1, 1, 1, 100, 100); //texto da senha
buildConstraints(constraints, 0, 2, 2, 1, 100, 100); //botão OK
```

Você vai mudar esses dois últimos argumentos em cada chamada de bui 1 dConstrai nts para que sejam um valor ou 0. Comece com a direção x (as proporções das colunas), que é o antepenúltimo argumento na lista anterior.

Se você olhar novamente a Figura 12.6 (a figura do painel com a grade imposta), observe que a segunda coluna é muito maior do que a primeira. Se você fosse escolher porcentagens teóricas para essas colunas, poderia dizer que a primeira representa 10% e a segunda representa 90%. (Essa é uma estimativa e também tudo o que você precisa fazer.) Com essas duas estimativas, você pode atribuí-las às células. Você não quer atribuir nenhum valor à célula com o botão OK, pois essa célula abrange as duas colunas e as porcentagens não funcionarão nesse caso. Adicione-os nas duas primeiras células, o rótulo de nome e o campo de texto de nome:

```
buildConstraints(constraints, 0, 0, 1, 1, 10, 100); //nome
buildConstraints(constraints, 1, 0, 1, 1, 90, 100); //texto do nome
```

E quanto aos valores das duas células restantes, o rótulo correspondente à senha e o campo de texto? Como as proporções das colunas já foram definidas pelo rótulo e pelo campo de nome, você não precisa redefini-los. Atribua valores 0 a essas duas células, bem como à caixa OK:

```
buildConstraints(constraints, 0, 1, 1, 1, 0, 100); //senha
buildConstraints(constraints, 1, 1, 1, 1, 0, 100); //texto da senha
buildConstraints(constraints, 0, 2, 2, 1, 0, 100); //botão 0K
```

Observe aqui, que um valor 0 não significa que a célula possui largura 0. Esses valores são proporções e não valores em pixel. Um 0 significa simplesmente que a proporção foi definida em outro lugar; o que o valor 0 diz é "amplie até caber".

Agora que os totais de todas as restrições wei ghtx são iguais a 100, você pode passar para os argumentos de weighty. Nesse caso, você tem três linhas. Dando uma olhada na grade que você desenhou, parece que o botão ocupa cerca de 20% e os campos de texto ocupam o restante (40% cada). Assim como no caso dos valores x, você precisa definir o valor de apenas uma célula por linha (os dois rótulos e o botão), com todas as outras células possuindo um valor de wei ghtx igual a 0.

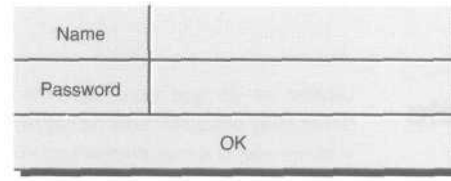
Aqui estão as cinco chamadas finais de buildConstraints( ) com os pesos no lugar:

```
buildConstraints(constraints, 0, 0, 1, 1, 10, 40); //nome
buildConstraints(constraints, 1, 0, 1, 1, 90, 0); //texto do nome
buildConstraints(constraints, 0, 1, 1, 1, 0, 40); //senha
buildConstraints(constraints, 1, 1, 1, 1, 0, 0); //texto da senha
buildConstraints(constraints, 0, 2, 2, 1, 0, 20); //botão 0K
```

A Figura 12.9 mostra o resultado com as proporções corretas.

Figura 12.9

Layout de grade de conteúdo, segundo passo.



Nesta etapa, o objetivo é tentar encontrar algumas proporções básicas de como as linhas e células serão espaçadas na tela. Você pode fazer algumas estimativas com base no tamanho esperado para os vários componentes, mas o mais provável é que você vá utilizar muita tentativa e erro nessa parte do processo.

Etapa quatro: Adicione e organize os componentes

Com o layout e as proporções definidas, você pode agora substituir os botões dos marcadores pelos rótulos e campos de texto reais. Como você já definiu tudo, todos os itens deverão funcionar perfeitamente, certo? Bem, é quase isso. A Figura 12.10 mostra o que você obterá se usar as mesmas restrições de antes e substituir os botões pelos componentes reais.

figura 12.10

Layout da grade de componente quase no fim.

Esse layout está concluído, mas é estranho. As caixas de texto são muito altas e o botão OK aumenta a largura da célula.

O que está faltando são as restrições que organizam os componentes dentro da célula. Existem duas delas: fil 1 e anchor.

A restrição fil 1 determina — para componentes que podem ser alongados nas duas direções — em que direção vai haver o alongamento (como caixas de texto e botões). f i 1 1 pode ter quatro valores, definidos como variáveis de classe na classe GridBagConstraints:

- GridBagConstraints.BOTH, que alonga os componente para preencher a célula nas duas direções.
- GridBagConstraints.NONE, que faz o componente ser apresentado em seu menor tamanho.
- GridBagConstraints. HORIZONTAL, que alonga o componente na direção horizontal.
- GridBagConstraints.VERTICAL, que alonga o componente na direção vertical.

Lembre-se de que esse layout é dinâmico. Você não vai definir as dimensões em pixel reais de todos os componentes; em vez disso, está informando a esses elementos em que direção eles podem crescer, considerando um painel que pode ser de qualquer tamanho.

Por padrão, a restrição fil 1 de todos os componentes é NONE. Por que os campos de texto e rótulos que preenchem as células são assim? Se você se lembrar do início do código deste exemplo, essa linha foi inserida no método 1nit( ):

```
constraints.fill = GridBagConstraints.BOTH;
```

Agora você sabe o que ela faz. Para a versão final desse applet, você vai querer remover essa linha e incluir valores de fi1 1 para cada componente independente.

A segunda restrição que afeta como um componente aparece na célula é anchor. Essa restrição se aplica apenas aos componentes que não estão preenchendo a célula inteira e ela diz ao AWT onde, dentro da célula, deve colocar o componente. Os valores possíveis para a restrição anchor são GridBagConstraints . CENTER, que alinha o componente tanto vertical como horizontalmente dentro da célula, ou em um de oito valores de direção:

GridBagConstraints.NORTH	GridBagConstraints.SOUTH
GridBagConstraints.NORTHEAST,	GridBagConstraints.SOUTHWEST,
GridBagConstraints.EAST	GridBagConstraints.WEST
Gri dBagConstraints.SOUTHEAST,	Gri dBagConstrai nts.NORTHWEST

O valor padrão de anchor é GridBagCons trai nts. CENTER.

Você define essas restrições da mesma maneira como fez para todas as outras: mudando variáveis de instância no objeto Gri dBagConstraints. Nesse caso, você pode mudar a definição de buildConstraints( ) para receber mais dois argumentos (eles são inteiros), ou pode simplesmente defini-los no corpo do método i ni t( ). A última opção é utilizada neste projeto.

Cuidado com os padrões. Lembre-se de que, como você está reutilizando o mesmo objeto GridBagConstraints para cada componente, podem existir alguns valores esquecidos quando se tiver concluído a utilização de um componente. Por outro lado, se um fill ou anchor de um objeto for o mesmo do anterior, você não precisará redefinir esse objeto.

Para esse exemplo, três alterações serão feitas nos valores fill e anchor dos componentes:

- Os rótulos não terão preenchimento (fill) e serão alinhados com EAST (para que colem no lado direito da célula).
- Os campos de texto serão preenchidos horizontalmente (para que eles iniciem uma linha alta, mas se alonguem de acordo com a largura da célula).
- O botão não terá preenchimento (fill) e será centralizado.

Isso é refletido no código completo, no final desta seção.

Etapa cinco: Faça ajustes

A medida que você trabalha com seus próprios programas e layouts de grade de conteúdo, o layout resultante freqüentemente exige certos acertos. Talvez seja preciso trabalhar com diversos valores das restrições para se obter uma interface mais adequada. Não há nada de errado nisso — o objetivo das etapas anteriores foi deixar os componentes bem perto de suas posições finais e não apresentar um layout perfeito logo na primeira tentativa.

A listagem 12.5 mostra o código completo do layout que você está criando nesta seção. Se você teve problemas para acompanhar a discussão até aqui, talvez ache útil percorrer esse código linha a linha para certificar-se de que entende as várias partes.

DIGITE **Listagem 12.5.** O texto completo de namepass.java.

```
1: import java.awt.*;
2:
3: public class NamePass extends java.applet.Applet {
4:
5:   void buildConstraints(GridBagConstraints gbc, int gx, int gy,
6: int gw, int gh, int wx, int wy) {
7:
8:       gbc.gridx = gx;
9:       gbc.gridy = gy;
10:      gbc.gridwidth = gw;
11:      gbc.gridheight = gh;
12:      gbc.weightx = wx;
13:      gbc.weighty = wy;
14: }
15:
```

Listagem 12.5. Continuação

```
16: public void init( ) {
17:     GridBagLayout gridbag • new GridBagLayout( );
18:     GridBagConstraints constraints = new GridBagConstraints( );
19:     setLayout(gridbag);
20:
21:     // Rótulo de nome
22:    buildConstraints(constraints, 0, 0, 1, 1, 10, 40);
23:     constraints.fill = GridBagConstraints.NONE;
24:     constraints.anchor • GridBagConstraints.EAST;
25:     Label labell - new Label("Name:", Label.LEFT);
26:     gridbag.setConstraints(labell, constraints);
27:     add(label1);
28:
29:     // Campo de texto de nome
30:    buildConstraints(constraints, 1, 0, 1, 1, 90, 0);
31:   constraints.fil1   = GridBagConstraints.HORIZONTAL;
32:     TextField tfname = new TextField( );
33:     gridbag.setConstraints(tfname, constraints);
34:     add(tfname);
35:
36:     // rótulo de senha
37:    buildConstraints(constraints, 0, 1, 1, 1, 0, 40);
38:     constraints.fill = GridBagConstraints.NONE;
39:     constraints.anchor = GridBagConstraints.EAST;
40:     Label Iabel2 - new Label("Password:", Label.LEFT);
41:     gridbag.setConstraints(1 abe12, constraints);
42:     add(label2);
43:
44:     // campo de texto de senha
45:    buildConstraints(constraints, 1, 1, 1, 1, 0, 0);
46:     constraints.fill = GridBagConstraints.HORIZONTAL;
47:     TextField tfpass = new TextField( );
48:     tfpass.setEchoCharacter('*');
49:     gridbag.setConstraints(tfpass, constraints);
50:     add(tfpass);
51:
52:     // Botão 0K
53:    buildConstraints(constraints, 0, 2, 2, 1, 0, 20);
54:     constraints.fill = GridBagConstraints.NONE;
55:     constraints.anchor • GridBagConstraints.CENTER;
56:     Button okb • new Button("0K");
57:     gridbag.setConstraints(okb, constraints);
58:     add(okb);
59:  }
60: }
```

A tag <APPLET> a seguir foi usada para testar esse applet:

```
<applet code="NamePass.class" height=180 width=240>
</applet>
```

Quando você compilar o applet, a chamada ao método setEchoCharacter( ) na linha 48 causa um aviso de desaprovação, pois esse método foi renomeado após a versão 1.02 da linguagem Java. Ele pode ser substituído por setEchoChar( ), se você estiver escrevendo um applet para a versão 2 da linguagem.

Preenchimento de célula

Antes de terminarmos com os layouts de grade de conteúdo, mais duas restrições merecem menção: ipadx e i pady. Essas duas restrições controlam o *enchimento* (o espaço extra em torno de um componente específico). Por definição, nenhum componente tem espaço extra em torno de si (o que é mais fácil ver nos componentes que preenchem suas células).

i padx adiciona espaço nos dois lados do componente e i pady adiciona em cima e embaixo.

Insets

Os espaços horizontal e vertical, gerados quando você cria um novo gerenciador de layout (ou usa i padx e ipady em layouts de grade de conteúdo), são utilizados para determinar o espaçamento entre os componentes em um painel. No entanto, as *insets* são usadas para determinar o espaço em torno do próprio painel. A classe Insets inclui valores para insets acima, abaixo, à esquerda e à direita, que são então usados quando o próprio painel é desenhado.

As insets determinam o espaço entre as margens de um painel e seus componentes.

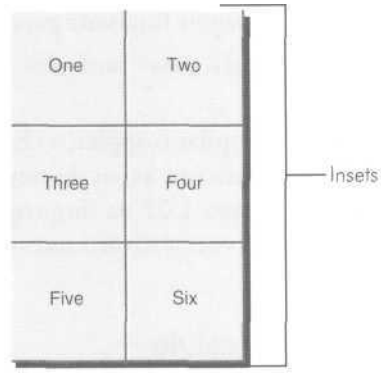
Para incluir um inset em seu layout, você anula o método insets( ) da versão 1.02 da linguagem Java ou o método getlnsets( ) da versão 2. Esses métodos fazem a mesma coisa.

Dentro do método insets( ) ou getlnsets( ), crie um novo objeto Insets, no qual o construtor da classe Insets recebe quatro valores inteiros representando os insets da parte superior, esquerda, inferior e direita do painel. O método insets( ) deve retornar, então, esse objeto Insets. Aqui está o código para incluir insets de um layout de grade: 10 nas partes superior e inferior e 30 à esquerda e à direita. A Figura 12.11 mostra o inset.

```
public Insets insets( ) {
    return new Insets(10, 30, 10, 30);
```

Figura 12.1 1

Um painel com ínsets de) 0 *pixels nas partes superior e inferior* e 30 *pixels* à *esquerda e à direita.*



Resumo

O expressionismo abstrato vai apenas até aqui, como você viu no dia que acabamos de terminar. Os gerenciadores de layout exigem certos ajustes para as pessoas que estão acostumadas a um controle mais preciso sobre o local em que os componentes aparecem em uma interface.

Agora você sabe usar os cinco diferentes gerenciadores de layout e os painéis. Quando trabalhar com o Abstract Windowing Toolkit, você verá que ele pode aproximar qualquer tipo de interface com o uso de contêineres aninhados e diferentes gerenciadores de layout.

Quando você dominar o desenvolvimento de uma interface com o usuário em Java, seus programas poderão oferecer algo que a maioria das outras linguagens de programação visuais não consegue: uma interface que funciona em várias plataformas sem modificação.

Para usar uma frase constantemente repetida: não sei se isso é arte, mas gosto dela.

Perauntas e respostas

P **Não gosto de trabalhar com gerenciadores de layout; ou eles são muito simples ou muito complicados (layout de grade de conteúdo). Mesmo com muitos ajustes, nunca consigo fazer meus applets ficarem como quero. O que desejo fazer é apenas definir o tamanho de meus componentes e colocá-los em uma posição x,y na tela. Posso fazer isso?**

R É possível, mas é muito problemático. O AWT foi projetado de tal modo que a interface gráfica com o usuário de um programa pudesse funcionar igualmente bem em diferentes plataformas e com diferentes resoluções de tela, fontes, tamanhos de tela e

coisas do tipo. Contar com coordenadas em pixel pode fazer um programa que parece bem em uma plataforma ser inútil em outras, onde os componentes se sobrepõem, são cortados pela margem de um contêiner e apresentam outros problemas de layout. Os gerenciadores de layout, colocando os elementos na tela dinamicamente, evitam esses problemas. Embora possam haver algumas diferenças entre o resultado final em diferentes plataformas, há menos probabilidade de essas diferenças serem catastróficas.

Ainda não está convencido? Use um gerenciador de layout nul 1 e o método reshape( ) para fazer um componente de um tamanho específico e colocá-lo em uma determinada posição:

```
setLayout(null);
Button myButton = new Button("0K");
myButton.reshape(10, 10, 30, 15);
```

Você pode encontrar mais dados sobre reshape( ) na classe Component.

P **Eu estava explorando as classes AWT e vi este subpacote chamado peer. Diversas referências às classes não-hierárquicas (peer) também estão espalhadas por toda a documentação da API Java. O que elas fazem?**

R As classes *não-hierárquicas* são responsáveis pelas partes específicas da plataforma do AWT. Por exemplo, ao criar uma janela AWT Java, você tem uma instância da classe Window que fornece comportamento genérico de janela e, então, você tem uma instância de implementação de classe WindowPeer que cria a janela bem específica para essa plataforma — uma janela de motivo no X Window, uma janela no estilo Macintosh no Macintosh ou uma janela Windows 95 no Windows 95. Essas classes não-hierárquicas também manipulam a comunicação entre o sistema de janelas e a própria janela Java. Separando o comportamento de componente genérico (as classes AWT) da implementação e da aparência real do sistema (as classes não-hierárquicas), você pode se ater ao fornecimento do comportamento em seu aplicativo Java e deixar que a implementação Java trate dos detalhes específicos da plataforma.

SEMANA 2

Dia 13

Respondendo à entrada do usuário em um applet

Com os conhecimentos obtidos até aqui, você pode projetar uma bela interface gráfica com o usuário, porém sem inteligência. Ela pode parecer uma interface que funciona — recebendo cliques de mouse em botões e outras interações como qualquer outro programa — mas, nada acontece em resposta a essas interações.

Para tornar uma interface funcional em Java, você precisa aprender a fazer um programa responder a eventos. *Eventos* são chamadas de método que o sistema de janelas da linguagem Java executa quando qualquer elemento de uma interface com o usuário é manipulado. Uma ampla variedade de eventos abrangem a utilização de mouse e teclado, incluindo eventos de clique de mouse, eventos de movimentação de mouse e eventos de pressionamento de tecla.

Hoje, você aprenderá a fazer um applet tratar de eventos usando as técnicas da linguagem Java 1.02, para que seus programas possam ser executados em qualquer navegador da Web que ofereça suporte à Java. No Dia 21, você aprenderá a tratar de eventos usando técnicas da linguagem Java 2.

Tratamento de eventos

Uma das coisas que você aprendeu ao criar applets pela primeira vez foi que existem forças ocultas quando o programa está em execução. O sistema de

janelas da linguagem Java chama métodos, como paint( ),init( )estart( ) automaticamente, quando eles são necessários, sem nenhum trabalho de sua parte.

Assim como acontece na programação de applets, o tratamento de eventos envolve métodos que são chamados automaticamente quando uma ação do usuário acarreta um evento.

TIPOS de eventos

Um evento é gerado em resposta a praticamente qualquer ação que um usuário possa realizar, durante o ciclo de vida de um programa Java. Cada movimento do mouse, clique em botão ou pressionamento de tecla gera um evento.

Em seus programas, você não precisa tratar de todos os eventos que possam ocorrer. Em vez disso, você trata dos eventos a que deseja que o programa responda e os restantes são ignorados. Por exemplo, se o usuário dá um clique com o mouse em algum lugar dentro da janela Applet ou pressiona uma tecla no teclado, você pode querer que o programa execute uma ação em resposta a esse evento.

Os eventos a seguir são alguns daqueles que podem ser tratados em seus próprios programas:

- *Cliques de mouse.* Botão do mouse para baixo (botão pressionado), botão do mouse para cima (botão solto) e clique no botão do mouse (pressionado e solto na mesma posição).
- *Movimentos de mouse.* O ponteiro do mouse entrando ou saindo de um componente da interface ou arrastes com o mouse (movimentos do ponteiro que ocorrem com o botão pressionado).
- *Pressionamentos de tecla.* Tecla pressionada, tecla solta e tecla digitada (pressionada e solta).
- *Eventos de interface com o usuário.* Clique no botão, barra de rolagem movimentada para cima e para baixo, menus instantâneos aparecendo etc.

O método handl eEvent( )

O tratamento de eventos é a área em que a linguagem Java mudou mais entre a versão 1.02 e a mais recente, 2. Os eventos são gerados e fluem pelo sistema praticamente da mesma maneira, independentemente da versão da linguagem que você está usando para criar um programa. A diferença reside em como os eventos são recebidos e processados.

Na linguagem Java 1.02, todos os eventos que ocorrem durante o ciclo de vida de seu programa fluem por esse programa e são tratados por um método chamado handl eEvent ( ). Esse método é definido na classe Component, que é herdada por java. applet. Applet, tornando-o disponível para todos os seus applets.

Quando um evento é enviado para o método handl eEvent ( ), esse método chama então um método de tratamento de eventos mais específico, dependendo do tipo de evento. Alguns desses métodos mais específicos são: mouseDown( ),mouseUp( ) e keyDown( ).

Para tratar de um evento em seus applets, você anula um desses métodos de tratamento de eventos específicos. Então, quando esse evento ocorre, seu método é chamado. Por exemplo, você poderia anular o método mouseDown ( ) com o comportamento para apresentar uma mensagem na janela Applet. Quando um evento de botão do mouse pressionado ocorre, essa mensagem é exibida.

Tratando de cliaues de mouse

Um dos eventos mais comuns em que você pode estar interessado é um clique de mouse. Os eventos de clique de mouse ocorrem quando um usuário dá um clique com o mouse em qualquer parte da interface do programa.

Você pode interceptar cliques de mouse para fazer coisas simples — por exemplo, alternar entre ligar e desligar o som em um applet, mover para o próximo slide em uma apresentação ou limpar a tela. Você também pode usar cliques de mouse em conjunto com movimentos de mouse, para conseguir uma interação mais complexa com o usuário.

Eventos de botão de mouse pressionado e solto

Quando um usuário dá um clique uma vez com o botão do mouse, dois eventos são gerados: um evento de botão do mouse para baixo, quando o botão é pressionado e um evento de botão do mouse para cima, quando o botão é solto. Essa divisão permite que diferentes coisas aconteçam em diferentes estágios do clique de botão do mouse.

É fácil tratar de eventos de mouse em seu applet; você anula a definição de método correta em seu applet e ela é chamada quando esse evento em particular ocorre. A seguir, há um exemplo da assinatura de método para um evento de botão do mouse para baixo:

```
public boolean mouseDown(Event evt, int x, int y) {
    // ...
}
```

O método mouseDown ( ) (e também o método mouseup( )) recebe três parâmetros: o evento em si e as coordenadas x e y em que o evento de botão do mouse para baixo ou para cima ocorreu.

O argumento evt é uma instância da classe Event. Todos os eventos geram uma instância da classe Event, que contém informações sobre onde e quando o evento ocorreu, de que tipo se trata e outros dados. Às vezes, ter uma alça para esse objeto Event é útil, como você descobrirá posteriormente nesta seção.

É particularmente interessante conhecer as coordenadas x e y do evento, quando passadas através dos argumentos x e y para o método mouseDown( ), pois você pode usá-las para determinar precisamente onde o clique de mouse ocorreu. Assim, por exemplo, se o evento de botão do mouse para baixo fosse sobre um botão gráfico, você poderia ativar esse botão. Observe que você pode obter as coordenadas x e y dentro do próprio objeto Event; nesse método, elas são passadas como variáveis separadas para tornar mais fácil lidar com elas.

A seguir há um método simples que exibe informações sobre um evento de botão do mouse para baixo, quando ele ocorre:

```
public boolean mouseDown(Event evt, int x, int y) {
    System.out.println("Mouse down at " + x + "," + y);
    return true;
}
```

Se você incluir esse método em seu applet, sempre que um usuário der um clique com o botão do mouse dentro do applet, essa mensagem será apresentada no dispositivo de saída padrão.

A utilização de System.out.println( ) em um applet causa comportamentos diversos em diferentes ambientes. O appletviewer apresenta a linha na mesma janela em que o comando appletviewer foi introduzido. O Netscape Navigator apresenta a saída em uma janela separada, chamada Java Console, que está disponível como a opção de menu suspenso Window | Java Console. O Microsoft Internet Explorer grava a saída Java em um arquivo separado. Verifique em seu ambiente para ver para onde é enviada a saída padrão de applets.

Observe que esse método, ao contrário dos outros métodos da biblioteca de classe Java que você estudou até agora, retorna um valor booleano, em vez de não retornar nada (usando-se a palavra-chave void).

O fato de um método de tratamento de eventos retornar true ou false se determina se dado componente pode interceptar um evento ou se ele precisa passar o evento para o componente que o inclui. A regra geral é que, se seu método intercepta e faz algo com o evento, ele deve retornar true. Se, por algum motivo, o método não fizer nada com esse evento, ele deve retornar false para que outros componentes no sistema de janelas global possam ter uma chance de tratar desse evento. Na maioria dos exemplos da lição de hoje, você vai interceptar eventos simples; portanto, aqui, a maioria dos métodos retornará true.

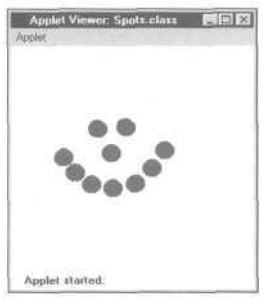
A segunda parte do clique de mouse é o método mouseUp( ), que é chamado quando o botão do mouse é solto. Para tratar de um evento de botão mouse para cima, inclua o método mouseUp( ) em seu applet. Esse método é muito parecido com o mouseDown ( ):

```
public boolean mouseUp(Event evt, int x, int y) {
    // ...
}
```

Um exemplo: Spots

Nesta seção, você vai criar um exemplo de applet que trata de eventos de botão do mouse para baixo. O applet Spots começa com uma tela em branco e depois pára e espera. Quando você dá um clique com o mouse na janela Applet, um ponto azul é desenhado. Você pode colocar até 10 pontos na tela. A Figura 13.1 mostra o applet Spots.

Figura 13.1
O *applet Spots.*



Comece desde o início e crie esse applet, partindo da definição de classe inicial:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;

public class Spots extends java.applet.Applet {
    final int MAXSPOTS = 10;
    int xspots[ ] = new int[MAXSPOTS];
    int yspots[ ] = new int[MAXSPOTS];
    int currspots = 0;
}
```

Essa classe usa três outras classes do Abstract Windowing Toolkit: Graphics, Color e Event. A última classe, Event, precisa ser importada em todos os applets que tratarem de eventos.

A classe Spots possui quatro variáveis de instância: MAXSPOTS, uma constante para determinar o número máximo de pontos que podem ser desenhados, dois arrays para armazenar as coordenadas x e y dos pontos que já foram desenhados e um inteiro para controlar o número atual de pontos.

Nota

A classe Event não inclui a cláusula implements Runnable em sua definição. Como você verá posteriormente, quando construir esse applet, ele também não possui um método run( ). Por que não? Porque, na verdade, ele não faz nada sozinho; ele apenas espera por uma entrada e depois faz algo quando a entrada aparece. O applet não precisa de threads, se não estiver fazendo algo ativamente, o tempo todo.

Em seguida, inclua o método init( ), que faz apenas uma coisa — define a cor de fundo como branca:

```
public void init( ) {
    setBackground(Color.white);
}
```

Nesse caso, você define o fundo em init( ), em vez de defini-lo em paint( ), como aconteceu nos exemplos anteriores, pois é preciso definir o fundo apenas uma vez. Como o método paint( ) é chamado repetidamente, sempre que um novo ponto é inserido, definir o fundo nesse método diminui sua velocidade desnecessariamente. Colocá-lo aqui é muito melhor.

A principal ação desse applet ocorre com o método mouseDown( ); portanto, você pode inseri-lo agora:

```
public boolean mouseDown(Event evt, int x, int y) {
    if (currspots < MAXSPOTS) {
      addspot(x,y);
      return true;
    )
    else {
      System.out.println("Too many spots.");
      return false;
     }
}
```

Quando o clique de mouse ocorre, o método mouseDown( ) faz um teste para ver se existem menos de 10 pontos. Se assim for, ele chamará o método addspot( ) (que você escreverá em breve) e retornará true (o evento de botão do mouse para baixo foi interceptado e tratado). Caso contrário, ele apenas imprime uma mensagem de erro e retorna false. O que addspot( ) faz? Ele insere as coordenadas do ponto nos arrays que armazenam as coordenadas, incrementa a variável currspots e, em seguida, chama repaint( ):

```
void addspot(int x, int y) {
    xspots[currspots] = x;
    yspots[currspots] = y;
    currspots++;
    repaint( );
}
```

Você pode estar se perguntando por que precisa controlar todos os pontos passados, além do ponto atual. Você faz isso devido ao método repaint( ): sempre que pinta a tela, você precisa pintar todos os pontos antigos, além do novo. Caso contrário, cada vez que você pintar um novo ponto, os mais antigos serão apagados.

Agora, passemos ao método paint( ):

```
public void paint(Graphics g) {
    g.setColor(Color.blue);
    for (int i = 0; i < currspots; i++) {
```

```
        g.fillOval(xspots[i] -10, yspots[i] - 10, 20, 20);
        }
    }
```

Dentro de paint( ), você apenas realiza um loop pelos pontos que armazenou nos arrays xspots e yspots, pintando cada um (na verdade, pintando-os um pouco à direita e para cima, a fim de que o ponto seja pintado em torno do ponteiro do mouse, em vez de abaixo e à direita).

Isso é tudo o que precisa ser feito para criar um applet que trata de cliques de mouse. Todo o restante é manipulado para você. Você precisa apenas incluir o comportamento apropriado em mouseDown( ) ou mouseUp( ), para interceptar e tratar desse evento.

A listagem 13.1 mostra o texto completo do applet Spots.

DIGITE

Listagem 13.1. O texto completo de spots.java.

```
1: import java.awt.Graphics;
2: import java.awt.Color;
3: import java.awt.Event;
4:
5: public class Spots extends java.applet.Applet
6:     final int MAXSPOTS = 10;
7:     int xspots[ ] = new int[MAXSPOTS];
8:     int yspots[ ] = new int[MAXSPOTS];
9:     int currspots = 0; ,
10:
11:    public void init( ) {
12:      setBackground(Color.white);
13:    }
14:
15:    public boolean mouseDown(Event evt, int x,
16:      if (currspots < MAXSPOTS) {
17:        addspot(x,y);
18:        return true;
19:      }
20:    el se {
21:        System.out.println("Too many spots.");
22:        return false;
23:      }
24:    }
25:
26:    void addspot(int x,int y) {
27:      xspots[currspots] = x;
28:      yspots[currspots] = y;
29:      currspots++;
30:      repaint( );
31:    }
32:
33:    public void paint(Graphics g) {
```

Listagem 13.1. Continuação

```
34:        g.setColor(Color.blue);
35:       for (int 1 = 0; i < currspots; i++) {
36:            g.fillOval(xspots[i] - 10, yspots[i] - 10, 20, 20);
37:       }
38:     }
39: }
```

Você pode carregar esse applet em uma página, usando o seguinte código HTML:

```
<applet code="Spots.class" height=250 width=250>
</applet>
```

Cliaues duplos

E se o evento de mouse em que você está interessado for mais do que um simples clique? E se você quiser controlar cliques duplos ou triplos? A classe Java Event fornece uma variável chamada clickCount para controlar essa informação, clickCount é um inteiro que representa o número de cliques de mouse consecutivos que ocorreram (onde "consecutivos" normalmente é determinado pelo sistema operacional ou pelo hardware de mouse). Se você estiver interessado em vários cliques de mouse em seus applets, então, pode testar esse valor no corpo de seu método mouseDown( ), como segue:

```
public boolean mouseDown(Event evt, int x, int y) {
    switch (evt.clickCount) {
      case 1: // clique simples
      case 2: // clique duplo
      case 3: // clique triplo
        // ...
    }
}
```

Um detalhe importante a ser observado quando você está procurando cliques duplos e triplos é que o método mouseDown ( ) é chamado uma vez para cada clique do botão. Considere o exemplo a seguir:

```
public boolean mouseDown(Event evt, int x, int y) {
    System.out.println("Click count: " + evt.clickCount);
    return false;
}
```

Se você colocar esse método em um applet, o seguinte será apresentado na saída padrão, quando for dado um clique triplo no botão do mouse:

```
Click count: 1
Click count: 2
Click count: 3
```

Como você aprenderá posteriormente ainda hoje, vários componentes geram um evento de ação quando recebem um clique duplo. Portanto, nem sempre é necessário usar mouseDown( ) para diferenciar entre cliques simples e duplos gerados por um componente.

Tratando de movimentos de mouse

Sempre que o mouse é movido, um evento de movimentação de mouse é gerado. Mover o mouse de um lado para outro do applet pode resultar em dezenas de eventos. Você vai descobrir dois tipos distintos de eventos de movimento do mouse no AWT: arrastes de mouse, em que o movimento ocorre com o botão do mouse pressionado, e movimentos de mouse puros, em que o botão do mouse não é pressionado.

Além disso, os eventos de entrada e saída do mouse são gerados sempre que o mouse entra ou sai de seu applet ou de qualquer componente ou contêiner do applet.

Para cada um desses eventos, métodos especiais interceptam os eventos, exatamente como os métodos mouseDown( ) e mouseUp( ) interceptam cliques de mouse.

Eventos de arraste e movimentação de mouse

Para interceptar e gerenciar eventos de movimentação de mouse, use os métodos mouseDrag( )em mouseMove( ).

O método mouseMove( ), para tratamento de movimentos puros do ponteiro do mouse, sem que o botão esteja pressionado, é muito semelhante aos métodos de clique de mouse:

```
public boolean mouseMove(Event evt, int x, int y) {
    // ...
}
```

O método mouseDrag( ) trata de movimentos de mouse feitos com o botão pressionado (uma movimentação completa de arraste consiste em um evento de botão do mouse para baixo, uma série de eventos de arraste do mouse para cada pixel em que o mouse é movido e um evento de botão do mouse para cima, quando o botão é liberado). O método mouseDrag( ) é como segue:

```
public boolean mouseDrag(Event evt, int x, int y) {
    // ...
}
```

Observe que, para os métodos mouseMove( ) emouseDrag( ),osargumentos das coordenadas x e y representam a nova posição do mouse e não sua posição inicial.

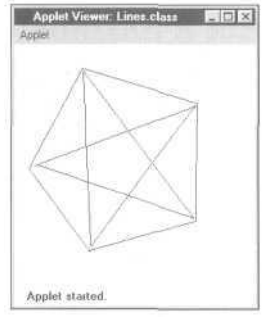
Eventos de entrada e saída de mouse

Os métodos mouseEnter( ) e mouseExit( ) são chamados quando o ponteiro do mouse entra ou sai de um applet ou de uma parte desse applet. Tanto mouseEnter( ) como mouseExit( ) possuem assinaturas semelhantes aos métodos de clique de mouse. Eles têm três argumentos: o objeto evento e as coordenadas x e y do ponto em que o mouse entrou ou saiu do applet. Os exemplos a seguir mostram as assinaturas de mouseEnter( ) e mouseExit( ):

```
public boolean mouseEnter(Event evt, int x, int y) {
    // •••
}

public boolean mouseExit(Event evt, int x, int y) {
    // •••
}
```

Um exercício: desenhando linhas

Nesta seção, você vai criar um applet que permite desenhar linhas retas na tela, arrastando-se do ponto inicial até o ponto final. A Figura 13.2 mostra o applet em funcionamento.



Assim como você fez com o applet Spots (no qual está baseado este applet), comece com a definição básica e trabalhe nela, incluindo os métodos apropriados para criar o applet. A seguir, há uma definição de classe simples para o applet Lines, com diversas variáveis de instância iniciais e um método init( ) simples:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;
import java.awt.Point;

public class Lines extends java.applet.Applet {
    final int MAXLINES = 10;
    Point starts[ ] = new Point[MAXLINES] ; // pontos iniciais
    Point ends[ ] = new Point[MAXLINES];  // pontos finais
    Point anchor;  // início da linha atual
    Point currentpoint; // final da linha atual
```

```
    int currline = 0; // número de linhas
    public void init( ) {
      setBackground(Color.white);
    }
}
```

Esse applet usa algumas variáveis a mais do que o applet Spots. Ao contrário de Spots, que controla as coordenadas inteiras individuais, o applet Lines controla objetos Point. Esses objetos representam uma coordenada x e uma coordenada y, encapsuladas em um único objeto. Para tratar de pontos, você importa a classe Point (java.awt.Point) e define diversas variáveis de instância que contêm pontos:

- O array starts contém pontos que representam os pontos iniciais das linhas já desenhadas.
- O array ends contém os pontos finais dessas mesmas linhas.
- anchor contém o ponto inicial da linha que está sendo desenhada.
- currentpoint contém o ponto final atual da linha que está sendo desenhada.
- currline contém ò número de linhas atual (para garantir que você não ultrapasse MAXLINES e para controlar a linha do array que vai acessar em seguida).

Finalmente, o método init( ), assim como no applet Spots, define o fundo do applet com a cor branca.

Os três eventos principais de que esse applet trata são mouseDown ( ), para definir o ponto-âncora da linha atual; mouseDrag( ), para animar a linha atual enquanto ela está sendo desenhada; e mouseUp( ), para definir o ponto final da nova linha. Dado que você tem variáveis de instância para conter cada um desses valores, basta apenas ligar as variáveis corretas aos métodos corretos. Aqui está mouseDown ( ), que define o ponto-âncora (mas apenas se você não tiver ultrapassado o número máximo de linhas):

```
public boolean mouseDown(Event evt, int x, int y) {
    if (currline < MAXLINES) {
      anchor = new Point(x,y);
      return true;
    }
    else {
      System.out.println("Too many lines.");
      return false;
    }
}
```

Enquanto o mouse está sendo arrastado para desenhar a linha, o applet anima a linha que está sendo desenhada. A medida que você arrasta o mouse, a nova linha se move com ele, do ponto-âncora até o ponteiro do mouse. O evento mouseDrag( ) contém o ponto atual cada vez que o mouse se move;

portanto, use esse método para controlar esse ponto (e para repintar a cada movimento, para que a linha seja "animada"). Observe que, se você ultrapassou o número máximo de linhas, não desejará realizar nenhum trabalho. A seguir, há o método mouseDrag( ) para realizar todas essas tarefas:

```
public boolean mouseDrag(Event evt, int x, int y) {
    if (currline < MAXLINES) {
      currentpoint = new Point(x,y);
      repaint( );
      return true;
    }
    else return false;
}
```

A nova linha não é incluída nos arrays de linhas antigas até que o botão do mouse seja solto. A seguir, há o método mousellp( ), que faz um teste para certificar-se de que você não ultrapassou o número máximo de linhas, antes de chamar o método addline( ) (descrito em seguida):

```
public boolean mouseUp(Event evt, int x, int y) {
    if (currline < MAXLINES) {
    addline(x,y);
    return true;
    }
    else return false;
}
```

No método addline( ), os arrays de pontos iniciais e finais são atualizados e o applet é repintado para levar a nova linha em conta:

```
void addline(int x,int y) {
    starts[currline] = anchor;
    ends[currline] = new Point(x,y);
    currline++;
    currentpoint = null;
    anchor = null;
    repaint( );
}
```

Observe que, nesse método, você também define currentpoint e anchor como null, pois a linha atual que você estava desenhando está terminada. Definindo essas variáveis como null, você pode testar esse valor no método pai nt ( ) para ver se precisa desenhar a linha atual.

Pintar o applet significa desenhar todas as linhas antigas, armazenadas nos arrays starts e ends, bem como desenhar a linha atual que está em andamento (cujos pontos finais estão em anchor e currentpoint, respectivamente). Para mostrar a animação da linha atual, desenhe-a em azul. A seguir, há o método pai nt ( ) do applet Lines:

```
public void paint(Graphics g) {
    // Desenha as linhas existentes
    for (int i = 0; i < currline; 1++) {
          g.drawLine(starts[i] .x, starts[i] .y,
```

```
            ends[i].x, ends[i].y);
        }
        // Desenha a linha atual
        g.setColor(Color.blue);
        if (currentpoint != null)
          g.drawLine(anchor.x, anchor.y,
            currentpoint.x, currentpoint.y);

    }
```

No método paint( ), quando está desenhando a linha atual, você testa primeiro para ver se currentpoint é null. Se for, então o applet não está no meio de um desenho de linha; portanto, ele não tem nenhum motivo para tentar desenhar uma linha que não existe. Testando currentpoint (e definindo currentpoint como null no método addline( )), você pode pintar apenas o que precisa. É isso; apenas 68 linhas de código e alguns métodos básicos, e você tem um aplicativo de desenho simples em seu navegador da Web. A listagem 13.2 mostra o texto completo do applet Lines para que você possa reunir as peças.

DIGITE

Listagem 13.2. O texto completo de lines.java.

```
1: import java.awt.Graphics;
2: import java.awt.Color;
3: import java.awt.Event;
4: import java.awt.Point;
5:
6: public class Lines extends java.applet.Applet {
7:     final int MAXLINES = 10;
8:     Point starts[ ] = new Point[HAXLINES]; // pontos iniciais
9:     Point ends[ ] = new Point[MAXLINES];  // pontos finais
10:    Point anctior;  // início da linha atual
11:    Point currentpoint; // final da linha atual
12:    int currline = 0; // número de linhas
13:
14:    public void init( ) {
15:        setBackground(Color.white);
16:    }
17:
18:    public boolean mouseDown(Event evt, int x, int y) {
19:        if (currline < MAXLINES) {
20:            anchor = new Point(x.y);
21:            return true;
22:        }
23:        else {
24:            System.out.println("Too many lines.");
25:            return false;
26:        }
27:    }
28:
29:    public boolean mouseIp(Event evt, int x, int y) {
30:        if (currline < MAXLINES) {
```

Listagem 13.2. Continuação

```
31:              addline(x,y);
32:              return true;
33:          }
34:      else      return false;
35:      }
36:
37:      public boolean mouseDrag(Event evt, int x, int y) {
38:          if (currline < MAXLINES) {
39:              currentpoint = new Point(x,y);
40:              repaint( );
41:              return true;
42:          }
43:          else return false;
44:      }
45:
46:      void addline(int x,int y) {
47:          starts[currline] = anchor;
48           ends[currline] = new Point(x,y);
49:          currline++;
50:          currentpoint = null;
51:          anchor = null;
52:          repaint( );
53:      }
54:
55:      public void paint(Graphics g) {
56:          // Desenha as linhas existentes
57:          for (int i = 0; i < currline; i++) {
58:              g.drawLine(starts[i].x, starts[i].y,
59:                  ends[i].x, ends[i].y);
60:          }
61:
62:          // desenha a linha atual
63:          g.setColor(Color.blue);
64:          if (currentpoint != null)
65:              g.drawLine(anchor.x,anchor.y,
66:                  currentpoint.x,currentpoint.y);
67:      }
68: }
```

Você pode testar esse applet usando o seguinte código HTML:

```
<applet code="Lines.class" height=250 width=250>
</applet>
```

Tratando de eventos de teclado

Um evento de teclado é gerado quando um usuário pressiona uma tecla. Usando eventos de teclado, você pode obter os valores das teclas que o usuário

pressionar para formar uma ação ou apenas obter entrada de caracteres dos usuários de seu applet.

Para que um evento de teclado seja recebido por um componente, esse componente deve ter o foco; em outras palavras, ele deve ser o componente da interface que está selecionado no momento para receber a entrada. Você aprenderá mais a respeito do foco posteriormente, ainda hoje, quando trabalhar com eventos de foco. O foco é mais fácil de entender quando você está considerando uma interface que contém vários campos de texto. O cursor pisca no campo de texto que possui o foco e um usuário pode introduzir texto nesse campo usando o teclado. Nenhum outro campo de texto pode receber texto até receber o foco. Todos os componentes, incluindo os contêineres, podem ser definidos para ter o foco.

Para indicar explicitamente que um componente possui o foco de entrada, o método requestFocus( ) do componente pode ser chamado sem argumentos. A instrução a seguir coloca o foco sobre um objeto Button chamado quit:

```
quit.requestFocus( );
```

Você pode colocar o foco em uma janela Applet chamando o método requestFocus( ) do applet.

Eventos de tecla pressionada e solta

Para tratar de um evento de teclado, use o método keyDown( ):

```
public boolean keyDown(Event evt, int key) {
    // •••
}
```

As teclas geradas pelos eventos de tecla pressionada (e passadas para keyDown( ) como o argumento key) são inteiros representando valores de caractere Unicode, o que inclui caracteres alfanuméricos, teclas de função, tabulações, retornos etc. Para usá-los como caracteres (por exemplo, para imprimi-los), você precisa fazer sua coerção em caracteres, como segue:

```
currentchar = (char)key;
```

A seguir, há um exemplo simples de um método keyDown ( ) que não faz nada, a não ser imprimir a tecla que você acabou de digitar, em suas representações em Unicode e de caractere (ver quais caracteres produzem quais valores pode ser divertido):

```
public boolean keyDown(Event evt, int key) {
    System.out.println("ASCII value: " + key);
    System.out.println("Character: " + (char)key);
    return true;
}
```

Assim como no caso dos cliques de mouse, cada evento de tecla pressionada também possui um evento de tecla solta correspondente. Para interceptar eventos de tecla solta, use o método keyUp( ):

```
public boolean keyUp(Event evt, int key) {
    // ...
}
```

Teclas padrão

A classe Event fornece um conjunto de variáveis de classe que faz referência a diversas teclas não-numéricas padrão, como as teclas de seta e de função. Se a interface de seu applet utiliza essas teclas, você pode fornecer um código mais legível testando esses nomes em seu método keyDown( ), em vez de testar seus valores numéricos (e também é mais provável que seu código funcione em diferentes plataformas, se você usar essas variáveis). Por exemplo, para testar se a seta para cima foi pressionada, você poderia usar o seguinte fragmento de código:

```
if (key == Event.UP) {
    // •••
}
```

Como os valores que essas variáveis de classe contêm são inteiros, você também pode usar a instrução switch para testá-las.

A Tabela 13.1 mostra as variáveis de classe Event padrão para várias teclas e as teclas que realmente representam.

Tabela 13.1 Teclas padrão definidas pela classe event.

Variável de classe	*Tecla representada*
Event.HOME	Tecla Home
Event.END	Tecla End
Event.PGUP	Tecla Page Up
Event.PGDN	Tecla Page Down
Event.UP	Seta para cima
Event.DOWN	Seta para baixo
Event.LEFT	Seta à esquerda
Event.RIGHT	Seta à direita
Event.F1	Tecla F1
Event.F2	Tecla F2
Event.F3	Tecla F3
Event.F4	Tecla F4
Event.F5	Tecla F5
Event.F6	Tecla F6
Event.F7	Tecla F7
Event.F8	Tecla F8
Event.F9	Tecla F9
Event.FIO	Tecla F10
Event.F11	Tecla F11
Event.F12	Tecla FI2

Um exemplo: inserindo, exibindo e movendo caracteres

Veja agora um applet que demonstra os eventos de teclado. Com esse applet, você digita um caractere e esse caractere é apresentado no centro da janela Applet. Em seguida, você pode mover esse caractere na tela usando as teclas de seta. Digitar outro caractere a qualquer momento muda o caractere que está sendo apresentado. A Figura 13.3 mostra um exemplo.

Figura 13.3

O *applet Keys.*

Este applet é, na verdade, menos complicado do que os applets anteriores que você usou. Ele possui apenas três métodos: init( ), keyDown( )epaint( ). As variáveis de instância também são mais simples, pois você só precisa controlar as posições x e y do caractere atual e os valores desse caractere. A seguir, há a definição de classe inicial:

```
import java.awt.Graphics;
import java.awt.Event;
import java.awt.Font;
import java.awt.Color;

public class Keys extends java.applet.Applet {

    char currkey;
    int currx;
    int curry;
}
```

Comece inserindo um método init( ).Aqui, init( ) é responsável por três tarefas: definir a cor de fundo, definir a fonte do applet (nesse caso, Helvetica, negrito, de 36 pontos) e definir a posição inicial do caractere (o meio da tela, menos alguns pontos para empurrá-lo um pouco para cima e para a direita).

```
public void init( ) {
    currx = (size( ).width / 2) - 8;
    curry = (size( ).height / 2) - 16;
    setBackground(Color.white);
    setFont(new Font("Helvetica", Font.BOLD, 36));
    requestFocus( );
}
```

A última instrução do método init ( ) coloca o foco de entrada na janela Applet. Essa instrução é necessária para garantir que a entrada do teclado seja recebida pelo componente que a está tratando — a própria janela Applet.

Nas versões anteriores da linguagem Java, chamar requestFocus ( ) não era exigido para que a janela Applet recebesse entrada de teclado, e você podia passar o foco para a janela dando um clique nela. Isso ainda vale nas versões mais recentes do Netscape Navigator e do Microsoft Internet Explorer. Entretanto, o appletviewer da linguagem Java 2 exige que requestFocus ( ) seja usado; caso contrário, a janela Applet nunca receberá o foco para entrada de teclado. Lembre-se dessa diferença, quando estiver testando applets que utilizam eventos de teclado. Provavelmente, é melhor usar requestFocus( ) para solicitar explicitamente o foco para uma janela Applet.

Como o comportamento desse applet é baseado na entrada do teclado, a maior parte do trabalho do applet ocorre no método keyDown ( ):

```
public boolean keyDown(Event evt, int key) {
    switch (key) {
      case Event.DOWN:
           curry += 5;
           break;
      case Event.UP:
           curry -= 5;
           break;
      case Event.LEFT:
           currx -= 5;
           break;
      case Event.RIGHT:
           currx += 5;
           break;
      default:
           currkey = (char)key;
    }
    repaint( );
    return true;
}
```

No centro do applet keyDown( ) está uma instrução switch que testa diferentes eventos de tecla. Se o evento é relativo a uma tecla de seta, a mudança apropriada é feita na posição do caractere. Se o evento representa qualquer outra tecla, o próprio caractere é alterado (essa é a parte padrão da instrução switch). O método termina com um repaint( ) e retorna true.

Aqui, o método paint( ) é quase trivial; ele apenas apresenta o caractere corrente na posição atual. Entretanto, observe que, quando o applet começa, ele não possui caractere inicial e nada para desenhar; portanto, você deve levar esse ponto em consideração. A variável currkey é inicializada como 0, de modo que você pinta o applet apenas se currkey possui um valor real:

```
public void paint(Graphics g) {
    if (currkey != 0) {
      g.drawString(String.valueOf(currkey),  currx,curry);
    }
)
```

A listagem 13.3 mostra o código-fonte completo do applet Keys.

DIGITE

Listagem 13.3. O texto completo de keys.java.

```
1:  import java.awt.Graphics;
2:  import java.awt.Event;
3:  import java.awt.Font;
4:  import java.awt.Color;
5:
6:  public class Keys extends java.applet.Applet {
7:
8:      char currkey;
9:      int currx;
10:     int curry;
11:
12:     public void init( ) {
13:         currx = (size( ).width / 2) -8; // padrão
14:         curry = (size( ).height / 2) -16;
15:
16:         setBackground(Color.white);
17:         setFont(new Font("Helvetica",Font.B0LD,36));
18:         requestFocus( );
19:     }
20:
21:     public boolean keyDown(Event evt, int key) {
22:         switch (key) {
23:         case Event.DOWN:
24:              curry +• 5;
25:              break;
26:         case Event.UP:
27:              curry -= 5;
28:              break;
29:         case Event.LEFT:
30:              currx -= 5;
31:              break;
32:         case Event.RIGHT:
33:              currx += 5;
34:              break;
35:         default:
36:             currkey = (char)key;
37:         }
38:
39:         repaint( );
40:         return true;
41:     }
42:
43:     public void paint(Graphics g) {
44:         if (currkey != 0) {
45:             g.drawString(String.vaiueOf(currkey), currx,curry);
46:      }
47:      }
48: }
```

Você pode testar o applet usando o seguinte código HTML:

```
<applet code="Keys.class" height=100 width=100>
</applet>
```

Testando teclas modificadoras e vários botões de mouse

Shift, Control (Ctrl) e Meta são teclas modificadoras. Elas não geram eventos de tecla, mas quando você recebe um evento de mouse ou de teclado, pode fazer um teste para ver se essas teclas modificadoras foram pressionadas quando o evento ocorreu. Às vezes, esse fato pode ser óbvio; por exemplo, as teclas alfanuméricas modificadas com Shift produzem eventos de tecla diferentes das não-modificadas. Para outros eventos, entretanto — eventos de mouse em particular —, talvez você queira tratar de um evento com uma tecla modificadora pressionada de forma diferente de uma versão normal desse evento.

A tecla Meta é normalmente usada em sistemas UNIX; em geral, ela é mapeada para a tecla Alt em teclados de PC e para Command (a maçã) em teclados Macintosh.

A classe Event fornece três métodos para testar se uma tecla modificadora está pressionada: shiftDown( ), metaDown( ) e Control Down( ). Todos eles retornam valores booleanos baseados no fato de essa tecla modificadora estar mesmo pressionada. Você pode usar esses três métodos em qualquer um dos métodos de tratamento de eventos (mouse ou teclado), chamando-as no objeto evento passado para esse método:

```
public boolean mouseDown(Event evt, int x, int y) {
    if (evt.shiftDown( ))
        // trata de clique acompanhado de shift
    else if ControlDown( )
        // trata de clique acompanhado de Control
    else // trata de clique normal
}
```

Um outro uso importante desses métodos de tecla modificadora é o teste de qual botão do mouse gerou um evento de mouse em particular em sistemas com dois ou três botões. Por padrão, os eventos de mouse (como o botão do mouse para baixo e arraste de mouse) são gerados independentemente de qual botão é utilizado. No entanto, os eventos Java mapeiam internamente as ações de botão esquerdo e central nas teclas modificadoras Meta e Control (Ctrl), respectivamente; portanto, fazendo os testes de tecla para a ação do botão do mouse. Testando as teclas modificadoras, você pode descobrir qual botão do mouse foi usado e executar para esses botões um comportamento diferente daquele que seria usado para o botão esquerdo. Use uma instrução i f para testar cada caso, como segue:

```
public boolean mouseDown(Event evt, int x, int y) {
    if (evt.metaDown( ))
      // trata de um clique com o botão direito do mouse
    else if (evt.controlDown( ))
      // trata de um clique com o botão central do mouse
    else // trata de um clique normal
}
```

Observe que, como esse mapeamento de vários botões de mouse para modificadores de teclado acontece automaticamente, você não precisa fazer muita coisa para garantir que seus applets ou aplicativos funcionem em diferentes sistemas, com diferentes tipos de dispositivos de mouse. Como os cliques com o botão esquerdo ou direito do mouse são mapeados em eventos de tecla modificadora, você pode usar as teclas modificadoras reais em sistemas com menos botões de mouse para gerar exatamente os mesmos resultados. Assim, por exemplo, a ação de manter a tecla Ctrl pressionada e dar um clique com o mouse no Windows ou manter a tecla Control pressionada no Macintosh é o mesmo que dar um clique com o botão central em um mouse de três botões; a ação de manter a tecla Command (apple) pressionada e clicar com o mouse no Mac é o mesmo que dar um clique com o botão direito em um mouse com dois ou três botões.

Considere, entretanto, que o uso de diferentes botões de mouse ou teclas modificadoras pode não ser imediatamente óbvio, se seu applet ou aplicativo for executado em um sistema com menos botões do que você utilizou para trabalhar. Considere a restrição de sua interface a um botão de mouse ou o fornecimento de ajuda ou documentação para explicar o uso de seu programa nesse caso.

A rotina de tratamento de eventos aenérica

Os métodos padrão sobre os quais você aprendeu hoje para o tratamento de eventos básicos em applets, são chamados por um método de tratamento de eventos genérico chamado handleEvent( ). Quando você usa o método handleEvent( ), o Abstract Windowing Toolkit trata genericamente dos eventos que ocorrem entre componentes de aplicativo e eventos baseados na entrada do usuário.

No método handleEvent( ) padrão, os eventos básicos são processados e os métodos que você aprendeu hoje são chamados. Para tratar de eventos além daqueles mencionados aqui (como os eventos de barras de rolagem ou de outro elementos de interface com o usuário), para mudar o comportamento do tratamento de eventos padrão ou para criar e circular seus próprios eventos, você precisa anular o método handleEvent( ) em seus próprios programas.

O método hand1eEvent( ) assemelha-se ao seguinte:

```
public boolean handleEvent(Event evt) {
    // •••
)
```

Para testar eventos específicos, examine a variável de instância id do objeto Event que é passado para handleEvent ( ). A ID do evento é um inteiro, mas, felizmente, a classe Event define todo um conjunto de IDs de evento como variáveis de classe, cujos nomes você pode testar no corpo de handl eEvent ( ). Como essas variáveis de classe são constantes inteiras, uma instrução switch funciona particularmente bem. Por exemplo, a seguir há um método handl e Event ( ) simples para imprimir informações de depuração sobre eventos de mouse:

```
public boolean handleEvent(Event evt) {
    switch (evt.id) {
      case Event.MOUSE_DOWN:
        System.out.println("MouseDown: " +
          evt.x + "," + evt.y);
        return true;
      case Event.MOUSE_up:
        System.out.println("MouseUp: " +
          evt.x + "," + evt.y);
        return true;
      case Event.MOUSE_MOVE:
        System.out.println("MouseMove: " +
          evt.x + "," + evt.y);
        return true;
      case Event.MOUSE_DRAG:
        System.out.println("MouseDrag: " +
          evt.x + "," + evt.y);
        return true;
      default:
        return false;
    }
}
```

Você pode testar os seguintes eventos de teclado:

- Event.KEY_PRESS é gerado quando uma tecla é pressionada (o mesmo que ocorre no método keyDown( )).
- Event. KEY_RELEASE é gerado quando uma tecla é solta.
- Event.KEY_ACTION e Event.KEY_ACTION_RELEASE são gerados quando uma tecla de "ação" (uma tecla de função, uma tecla de seta, Page Up, Page Down ou Home) é pressionada ou solta.

Você pode testar os seguintes eventos de mouse:

- Event.M0USE_D0WN é gerado quando o botão do mouse é pressionado (o mesmo que ocorre no método mouseOown( )).
- Event.MOUSE_UP é gerado quando o botão do mouse é solto (o mesmo que ocorre no método mousellp ( )).
- Event.M0USE_M0VE é gerado quando o mouse é movido (o mesmo que ocorre no método mouseMove( )).

- Event.MOUSE_DRAG é gerado quando o mouse é movido com o botão pressionado (o mesmo que ocorre no método mouseDrag( )).
- Event.MOUSE_ENTER é gerado quando o mouse entra no applet (ou em um componente desse applet). Você também pode usar o método mouseEnter( ).
- Event.MOUSE_EXIT é gerado quando o mouse sai do applet. Você também pode usar o método mouseExit( ).

Observe que, se você anular handleEvent( ) em sua classe, nenhum dos métodos de tratamento de eventos padrão sobre os quais você aprendeu hoje é chamado, a não ser que você os chame explicitamente no corpo de handle Event ( ). Portanto, tome cuidado se você decidir anular esse evento. A melhor maneira de evitar esse problema é testar o evento em que você está interessado e, se não se tratar desse evento, chamar super. handl eEvent( ) para que a superclasse que define handleEvent( ) possa processar as coisas. A seguir, há um exemplo:

```
public boolean handleEvent(Event evt) {
    if (evt.id == Event.MOUSE_DOWN) {
        // processa o botão do mouse para baixo
        return true;
    } else
       return super.handleEvent(evt);
}
```

Além disso, observe que, assim como os métodos específicos para eventos específicos, handl eEvent ( ) também retorna um valor booleano. O valor retornado aqui é particularmente importante; se você passar o tratamento do evento para outro método, deverá retornar false (o próprio método que você chama retorna true ou false). Se você tratar do evento no corpo desse método, retorne true. Se você passar o evento para uma superclasse, esse método retornará true ou fal se; não é necessário que você o retorne.

Tratando de eventos de componente

As técnicas de tratamento de eventos que você aprendeu até agora focalizaram a interação do usuário — dar um clique com um mouse, pressionar teclas em um teclado e coisas assim. Também existem eventos para tratar de eventos específicos que ocorrem em componentes, como botões, áreas de texto e outros elementos de interface. Por exemplo, os botões usam eventos de ação que são disparados quando o botão é pressionado. Você não precisa se preocupar com o botão do mouse para baixo ou para cima, ou determinar onde a interação de mouse ocorreu; o componente trata de tudo isso para você.

Os seguintes eventos podem ser gerados a partir da interação com componentes de interface:

- *Eventos de ação.* Os principais eventos da maioria dos componentes da interface, para indicar que esse componente foi "ativado." Os eventos de ação são gerados quando um botão é pressionado, quando uma caixa ou botão de rádio é selecionado ou perde a seleção, quando uma opção de um item de menu é feita ou quando o usuário pressiona Return ou Enter dentro de um campo de texto.
- *Eventos de seleção ou anulação de seleção de lista.* Esses eventos são gerados quando é marcada uma caixa de seleção ou uma opção de item de menu (o que também gera um evento de ação).
- *Eventos de foco obtido ou perdido.* Esses eventos podem ser gerados por qualquer componente, ou em resposta a um clique de mouse ou como parte da passagem do foco usando-se a tecla Tab. "Foco obtido" significa exatamente isso; o componente tem o foco de entrada e agora pode ser selecionado, pode-se digitar nele ou ele pode ser ativado. "Foco perdido" significa que o foco de entrada mudou para algum outro componente.

Tratando de eventos de ação

Um evento de ação é o evento de interface mais utilizado de todos e, por esse motivo, um método especial é usado para tratá-lo, exatamente como acontece com os métodos básicos de evento de mouse e teclado.

Para interceptar um evento de ação gerado por qualquer componente, defina um método acti on ( ) em seu applet ou classe, com a seguinte assinatura:

```
public boolean action(Event evt, Object arg) {
    // ...
}
```

Esse método action( ) deve ser semelhante aos métodos básicos de evento de mouse e teclado. Assim como esses métodos mencionados, este recebe o objeto evento que representa esse evento. Também é passado um objeto extra (nesse código, o parâmetro arg), que pode ser de qualquer tipo de classe.

O tipo de objeto do segundo argumento do método de ação depende do componente da interface que está gerando a ação. A definição básica é que se trata de "qualquer argumento arbitrário", determinado pelo próprio componente, para ser passado com todas as informações extras que possam ser úteis no processamento dessa ação. A Tabela 13.2 mostra os argumentos extras de cada componente de interface.

Tabela 13.2 Argumentos de ação para cada componente.

Componente	Tipo de *argumento*	*Contém*
Botões	String	O rótulo do botão
Caixas de seleção	Booleano	Sempre true
Botões de rádio	Booleano	Sempre true
Menus de escolha	String	O rótulo do item selecionado
Campos de texto	String	O texto que está dentro do campo

Dentro do método action( ), a primeira coisa a fazer é testar para ver qual componente gerou a ação (ao contrário dos eventos de mouse ou teclado, em que isso não importa realmente, pois diferentes componentes podem gerar ações). Felizmente, o objeto Event que você recebe quando action( ) é chamado contém uma variável de instância denominada target, que contém uma referência ao objeto que recebeu o evento. Você pode usar o operador instanceof para descobrir qual componente gerou o evento, como segue:

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof TextField)
      return handleText(evt.target);
    else if (evt.target instanceof Choice)
      return handleChoice(arg);
      // ...
    return false;
}
```

Nesse exemplo, action( ) poderia ter sido gerado por um TextField ou por um menu de escolha; as instruções if determinam qual deles gerou realmente o evento e chamam algum outro método (aqui, handle Texto ( ) ou handleChoice( )) para tratar dele. (Nem handl eTexto ( ) nem handleChoice( ) são métodos AWT; eles são apenas exemplos de nomes que poderiam ser usados para métodos auxiliares. Uma prática comum é criar métodos auxiliares para que action( ) não fique congestionado com código demais.)

Assim como acontece com os outros métodos de evento, action( ) retorna um valor booleano. Assim como em todos os métodos de evento, você deve retornar true, se action( ) em si tratar do método ou false, se ele passar o método para outro lugar (ou ignorá-lo). Nesse exemplo, você passou o controle para os métodos handleText( ) ou hand1eChoice( ), e eles devem retornar true ou false; portanto, você pode retornar false (lembre-se: você retorna true apenas se esse método processou o evento).

Mais complicações ocorrem quando você possui muitos componentes e todos eles têm a mesma classe — por exemplo, vários botões. Todos eles geram ações e todos eles são instâncias de Button. Esse argumento extra entra em ação aqui: você pode usar os rótulos, itens ou conteúdo do componente para

determinar qual gerou o evento e utilizar comparações de string simples para fazer uma escolha dentre eles. (Não se esqueça de fazer a coerção do argumento no objeto correto.)

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
      String labl = (String)arg;
      if  (labl.equals("0K"))
        // trata do botão 0K
      else  if  (labl.equals("Cancel"))
            // trata do botão Cancel
      else  if  (labl.equals("Browse"))
            // trata do botão Browse
      //  ...
    }
}
```

E quanto as caixas de seleção e os botões de rádio? O argumento extra deles é sempre `true`, o que não é muito útil para se fazer um teste. Geralmente, você não deve reagir a uma caixa de seleção ou um botão de rádio quando o componente está marcado. Normalmente, as caixas de seleção e os botões de rádio podem ser selecionados ou ter sua seleção desfeita à vontade pelo usuário e, então, seus valores são verificados em algum outro ponto (por exemplo, quando um botão é pressionado).

Se você quiser realmente que seu programa reaja a uma caixa de seleção ou a um botão de rádio quando estiver marcado, pode usar o método `getLabel ( )` para extrair o rótulo da caixa de seleção de dentro de `action( )`, em vez de utilizar o argumento extra. (Na verdade, todos os componentes possuem alguma espécie de método desse tipo; apenas é mais fácil usar quando ele é passado como argumento extra.)

Tratando de eventos de foco

Como foi mencionado anteriormente, os eventos de ação são seguramente os eventos de interface mais utilizados, dos quais você tratará com relação aos componentes que aprendeu nesta lição. Contudo, você pode usar outros quatro eventos em seus programas: seleção de lista, anulação de seleção de lista, foco obtido e foco perdido.

No caso dos eventos de foco obtido e foco perdido, você pode usar os métodos gotFocus( ) e lostFocus( ), que são utilizados da mesma maneira que acti on ( ). Aqui estão suas assinaturas:

```
public boolean gotFocus(Event evt, Object arg) {
    // ...
i
```

```
public boolean lostFocus(Event evt, Object arg) {
    // ...
}
```

No caso dos eventos de seleção de lista e anulação de seleção de lista, nenhum método que se possa anular com facilidade está disponível. Você precisa usar o método handleEvent ( ) para esses eventos, como segue:

```
public boolean handleEvent(Event evt) {
    if (evt.id == Event.LIST_SELECT)
      handleSelect(Event);
    else if (evt.id == Event.LIST_DESELECT)
      handleDeselect(Event);
    else return super.handleEvent(evt);
}
```

Nesse fragmento de código, Event.LIST_SELECT e Event.LIST_DESELECT são as IDs de evento oficiais para os eventos de seleção de lista e anulação de seleção de lista, e aqui o controle foi passado para dois métodos auxiliares (handl eSelect( ) e handl eDeselect( )), que, teoricamente, estão definidos em outro lugar. Observe também a chamada a super. handl eEvent ( ) no final; essa chamada permite que outros eventos voltem sem problemas para o método handl eEvent ( ) original.

Eventos de área de texto

As áreas de texto possuem os mesmos eventos que os campos de texto. Você pode usar os métodos gotFocus( ) e lostFocus( ) para capturar eventos de foco. Por exemplo:

```
public boolean gotFocus(Event evt, Object arg) {
    // ...
}
public boolean lostFocus(Event evt, Object arg) {
    // ...
}
```

Eventos de lista de rolagem

As listas de rolagem geram três tipos diferentes de eventos: a seleção ou anulação de seleção de um item específico da lista resulta em um evento de seleção de lista ou anulação de seleção de lista, e dar um clique duplo em um item da lista resulta em um evento de ação.

Você pode anular o evento action ( ) para tratar de um item de uma lista que está recebendo um clique duplo. No caso da seleção de lista e da anulação da seleção de lista, você precisa anular handleEvent ( ) e testar as IDs de evento LIST_SELECT e LIST_DESELECT.

Eventos de barra de rolagem

Se você gosta de mexer com eventos, vai adorar as barras de rolagem. Um conjunto inteiro de eventos é gerado e tratado apenas por diferentes movimentações de barra de rolagem. Você precisa usar handleEvent ( ) para todos esses eventos. A Tabela 13.3 mostra as IDs de evento a serem procuradas e os movimentos que as disparam.

Tabela 13.3 Eventos de barra de rolagem.

ID do evento	*O que ele representa*
SCROLL_ABSOLUTE	Gerado quando a barra de deslocamento de uma caixa de rolagem é movido.
SCROLL_LINE_DOWN	Gerado quando a parte inferior ou o ponto final da esquerda (botão) de uma barra de rolagem é selecionado.
SCROLL_LINE_UP	Gerado quando a parte superior ou o ponto final da direita (botão) de uma barra de rolagem é selecionado.
SCROLL_PAGE_DOWN	Gerado quando o campo da barra de rolagem abaixo (ou à esquerda) do elevador é selecionado.
SCROLL_PAGE_UP	Gerado quando o campo da barra de rolagem acima (ou à direita) do elevador é selecionado.

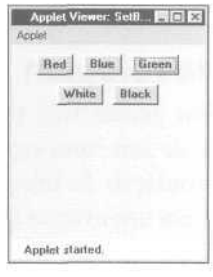
Um exemplo: alternador de cor de fundo

Se você possui apenas fragmentos de código a partir dos quais vai trabalhar, é difícil ter uma idéia de como todas as partes se encaixam. Para corrigir esse problema agora, você pode criar um applet AWT simples.

O applet que você vai construir nesta seção, ilustrado na Figura 13.4, usa cinco botões, organizados na parte superior da tela, cada um dos quais rotulado com uma cor. Cada botão muda a cor do fundo do applet, de acordo com o seu rótulo.

Figura 13.4

O *applet SetBack.*



Para o primeiro passo desta seção, você criará o código da interface com o usuário do applet. Normalmente, essa é a melhor maneira de abordar qualquer applet baseado no AWT: crie os componentes e o layout, e certifique-se de que tudo esteja certo, antes de ligar os eventos para realmente fazer o applet funcionar.

Para esse applet, os componentes e o layout não poderiam ser mais simples. O applet contém cinco botões simples, organizados em uma linha na parte superior da tela. Um layout de fluxo funciona melhor para essa organização e exige pouco trabalho.

A seguir está o código da estrutura de classe e o método init( ) criado para esse applet. O Fl owLayout é centralizado e cada botão terá 10 pontos entre ele. Depois disso, basta apenas criar e incluir cada um dos botões.

```
import java.awt.*;

public class SetBack extends java.applet.Applet {

    Button redButton.blueButton.greenButton,
        whiteButton.blackButton;

    public void init( ) {
        setBackground(Color.white);
        setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
        redButton = new Button("Red");
        add(redButton);
        blueButton = new Button("Blue");
        add(blueButton);
        greenButton = new Button("Green");
        add(greenButton);
        whiteButton = new Button("White");
        add(whiteButton);
        blackButton = new Button("Black");
        add(blackButton);
        }
```

Adicionando o código do evento

Os botões, quando estão pressionados, resultam em eventos de ação. E, como foi aprendido anteriormente, para tratar de um evento de ação, você usa o método action( ). Aqui, o método action( ) faz o seguinte:

- Testa para garantir que o alvo do evento é mesmo um botão.
- Testa de forma mais efetiva para descobrir exatamente o botão que foi pressionado.
- Muda o fundo para a cor nomeada pelo botão.
- Chama repaint( ) (a ação de mudar o fundo não é suficiente).

Antes de realmente escrever o método action( ), vamos ajudá-lo a tomar mais uma decisão de design. Os três últimos passos são basicamente idênticos para cada botão, com pequenas diferenças; portanto, faz sentido colocá-los em seu próprio método, que você pode chamar de changeColor( ). Fazer isso simplifica a lógica do próprio método acti on ( ).

Com essa decisão tomada, criar o método action( ) em si é fácil:

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
```

```
            changeColor((Button)evt.target);
            return true;
        } else return false;
    }
```

Não há muita diferença entre esse método acti on ( ) em relação aos mais simples, criados na seção sobre ações. O primeiro passo é usar evt. target para certificar-se de que o componente é um botão, no momento em que você passa o controle para o método changeColor( ) (que ainda vai ser escrito) e retorna true. Se o evento não é um botão, você retorna false.

Observe o argumento de changeColor( ). Com esse argumento, você passa o objeto botão que recebeu o evento para o método changeColor ( ). (O objeto em evt.target é uma instância da classe Object; portanto, é necessário fazer sua coerção para um Button para que você possa usá-lo como um botão.) O método changeColor( ) tratará disso.

Vá em frente e defina o método changeColor( ) agora. O principal objetivo de changeColor( ) é decidir o botão que recebeu o clique. Lembre-se de que o argumento extra de action( ) era o rótulo do botão. Embora você possa utilizar uma comparação de strings em changeColor( ) para descobrir qual botão foi pressionado, essa solução não é a mais elegante e ela vincula demais o seu código de evento à interface com o usuário. Se você decidir mudar um rótulo de botão, terá de voltar e trabalhar também em seu código de evento. Assim, neste applet, você pode ignorar completamente o argumento extra.

Então, como você identifica o botão que foi pressionado? Neste ponto, as variáveis de instância de botão entram em ação. O objeto contido na variável de instância de destino do evento — a que você passou para changeCol or ( ) — é uma instância de Button, e uma dessas variáveis de instância contém uma referência a esse mesmo objeto. Em changeColor( ), você precisa apenas comparar os dois para ver se eles são o mesmo objeto, definir o fundo e repintar, como segue:

```
void changeColor(Button b) {
    if (b == redButton) setBackground(Color.red);
    else if (b == blueButton) setBackground(Color.blue);
    else if (b == greenButton) setBackground(Color.green);
    else if (b == whiteButton) setBackground(Color.white);
    else setBackground(Color.black);
    repaint( );
}
```

A partir da interface com o usuário, pressionar um botão chama action( ),action( ) chama changeColor( ) e changeCol or( ) define o fundo apropriado. Fácil! A listagem 13.4 mostra o applet final.

DIGITE

Listagem 13.4. O texto completo de setback.java.

```
1: import java.awt.*;
2:
3: public class SetBack extends java.applet.Applet {
4:
5:      Button redButton,blueButton,greenButton,whi teButton,blackButton;
6:
7:      public void init( ) {
8:           setBackground(Color.white);
9:           setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
10:
11:          redButton = new Button("Red");
12:          add(redButton);
13:          blueButton = new Button("Blue");
14:          add(blueButton);
15:          greenButton = new Button("Green");
16:          add(greenButton);
17:          whiteButton = new Button("White");
18:          add(whiteButton);
19:          blackButton = new Button("Black");
20:          add(blackButton);
21:     }
22:
23:     public boolean action(Event evt, Object arg) {
24:          if (evt.target instanceof Button) {
25:               changeColor((Button)evt.target);
26:               return true;
27:          } else return false;
28:     }
29:
30:     void changeColor(Button b) {
31:          if (b •• redButton) setBackground(Color.red);
32:          else if (b -« blueButton) setBackground(Color.blue);
33:          else if (b == greenButton) setBackground(Color.green);
34:          else if (b == whiteButton) setBackground(Color.white);
35:          else setBackground(Color.black);
36:
37:          repaint( );
38:     }
39: }
```

Você pode testar o applet usando o seguinte código HTML:

```
<applet code="SetBack.java" width=200 height=200>
</applet>
```

Resumo

O término deste dia de trabalho é um grande evento em sua carreira de programação em Java. A capacidade de tratar de eventos torna possível escrever applets Java completos, com interfaces gráficas que podem ser usadas para a interação com o usuário.

Amanhã, você aprimorará seu conhecimento do Abstract Windowing Toolkit com um projeto mais sofisticado e uma abordagem de recursos como janelas independentes.

Na Semana 3, você terá a chance de criar um aplicativo funcional que utiliza o Swing, o novo pacote de janelas introduzido com a linguagem Java 2.

Perguntas e respostas

P **Tenho uma nova classe de botão definida para parecer diferente dos objetos botão padrão do AWT na 1.02. Gostaria de implementar callbacks nesse botão (isto é, executar uma função arbitrária quando o botão for pressionado), mas não consigo descobrir como fazer a linguagem Java executar um método arbitrário. Em C++, tenho simplesmente um ponteiro para uma função. Em Smalltalk, utilizo perform:. Como posso fazer isso em Java?**

R Você não pode fazer isso usando Java 1.02; as ações de botão são executadas a partir de um evento action( ), que deve estar contido na mesma classe que o botão. Você precisa colocar seu botão em uma subclasse, sempre que quiser criar um comportamento diferente para ele. Esse aspecto da linguagem é um dos motivos pelos quais o modelo de tratamento de eventos foi alterado após a Java 1.02. A criação de seus próprios componentes é muito mais fácil e eficiente quando o código do evento não está demasiadamente vinculado ao código da interface com o usuário.

SEMANA 2

Dia 14

Desenvolvendo interfaces com o usuário avançadas com o AWT

Este é o último dia em que você aprenderá sobre o Abstract Windowing Toolkit. Se você acha isso bom ou ruim depende de quanto se sentiu à vontade com suas classes.

Se você acha que isso é bom, então deverá se sentir melhor ainda com o AWT, depois de aprender a respeito de alguns de seus recursos avançados hoje.

Você vai aprimorar tudo o que aprendeu nos dias anteriores a respeito de componentes, gerenciadores de layout e eventos de interface com o usuário, e vários conceitos novos são apresentados:

- Como os componentes funcionam e as diversas coisas que você pode fazer com eles
- Janelas, quadros e caixas de diálogo
- Menus
- Criação de aplicativos AWT independentes

Janelas, auadros e caixas de diálogo

Além do que foi abordado até aqui, o AWT fornece recursos para a criação de elementos de interface com o usuário fora do applet e da estrutura do navegador, incluindo janelas, quadros e caixas de diálogo. Esses recursos permitem

que você crie aplicativos completos, como parte de seu applet ou independentemente, para aplicativos Tava auto-suficientes.

As classes Window

As classes AW1 para produzir janelas e caixas de dialogo herdam de uma unica classe: Window. A classe Window herda de Container, como acontece com os painéis e applets, e ela fornece comportamento genérico para todos os elementos relacionados às janelas.

Geralmente, você não utiliza instâncias de Wi ndow. Em vez disso, você usa duas de suas subclasses: Frame e Dialog.

A classe Frame fornece uma janela com uma barra de título, caixas de fechamento e outros recursos de janela específicos da plataforma. Os quadros também permitem que você inclua barras de menu. A subclasse Dialog é uma forma mais limitada de Frame, que normalmente não possui um título. File-Dialog, uma subclasse de Dialog, fornece uma caixa de diálogo de seleção de arquivos padrão (normalmente, útil apenas dentro de aplicativos Java, devido às restrições de segurança dos applets).

Quando quer inserir uma nova janela ou caixa de diálogo em seu applet ou aplicativo, você cria subclasses das classes Frame e Dialog.

Quadros

NOVO TERMO *Quadros* são janelas que são independentes de um applet e do navegador que o contém; são janelas separadas, com seus próprios títulos, alças de redimensionamento, caixas de fechamento e barras de menu. Você pode criar quadros para seus próprios applets para produzir janelas ou pode usá-los em aplicativos Java para manter o conteúdo desse aplicativo.

Um quadro (frame) é uma janela específica da plataforma, com um título, uma barra de menu, caixas de fechamento, alças de redimensionamento e outros recursos de janela.

Use um dos seguintes construtores para criar um quadro:

- new Frame ( ) cria um quadro básico, sem título.
- new Frame *(String)* cria um quadro básico com o título fornecido.

Como os quadros herdam de Window, que herda de Container, que herda de Component, eles são criados e utilizados de forma muito parecida com os outros componentes do AWT. Os quadros são contêineres, exatamente como acontece com os painéis; portanto, você pode inserir outros componentes neles, como faria com os painéis normais, usando o método add( ). O layout padrão dos quadros é BorderLayout. A seguir, há um exemplo simples que cria um quadro, define seu layout e insere dois botões:

```
win = new Frame("My Cool Window");
win.setLayoutfnew BorderLayout(10, 20));
win.add("North", new Button("Start"));
win.add("Center", new Button("Move"));
```


Para definir um tamanho para o novo quadro, use o método resize( ) com a largura e a altura do novo quadro. Por exemplo, a linha de código a seguir redimensiona a janela com 100 pixels de largura e 200 pixels de altura:

```
win.resize(100, 200);
```

Como os diferentes sistemas possuem diferentes concepções do que é um pixel e resoluções distintas para esses pixels, é difícil criar uma janela que seja do tamanho "certo" para todas as plataformas. As janelas que funcionam bem para uma podem ficar grandes demais ou pequenas demais para outra.

Um modo de contornar esse problema é usar o método pack ( ) em lugar de resize( ). O método pack( ), que não possui argumentos, cria uma janela do menor tamanho possível, dados os tamanhos atuais de todos os componentes internos da janela, o gerenciador de layout e os insets em uso. O exemplo a seguir cria dois botões e os adiciona uma janela. A janela é, então, redimensionada para o menor tamanho possível que ainda pode conter esses botões.

```
FlowLayout flo = new FlowLayout( );
Button ok = new Button("0K");
Button cancel = new Button("Cancel");
win = new Frame("My Other Cool Window");
win.setLayout(flo);
win.add(ok);
win.add(cancel);
win.pack( );
```

Quando você cria uma janela, ela fica invisível. Você precisa usar o método show ( ) para fazer a janela aparecer na tela. Para ocultá-la novamente, você pode usar hide( ):

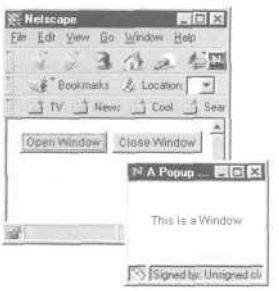
```
win.show( );
```

Observe que, quando você apresenta uma janela dentro de applets, o navegador pode indicar de algum modo que ela não é uma janela de navegador normal — normalmente, com um alerta na própria janela.

No Netscape, uma mensagem na parte inferior de cada janela informa: Unsigned Java Applet Window. Esse alerta se destina a permitir que os usuários saibam que a janela é proveniente do applet e não do próprio navegador. (Lembre-se de que a classe Frame produz janelas muito parecidas com as janelas de sistema normais.) O alerta serve para evitar que um programador mal-intencionado crie um applet que imite outros programas para adquirir senhas de usuário e outras informações. A não ser que seu applet passe pelas etapas de solicitar — e receber — certificação do usuário, ele parece ser um programa confiável. Isso é descrito no Dia 16.

As listagens 14.1 e 14.2 mostram as classes que compõem um applet simples com um quadro de janela instantânea. O applet e a janela estão ilustrados na Figura 14.1. O applet possui dois botões: um para apresentar a janela e outro para ocultá-la. O quadro da janela em si, criado a partir de uma subclasse chamada BaseFramel, contém um único rótulo: This is a Window. Essa janela e esse applet básicos são referidos por toda esta seção, de modo que, quanto mais você entender o que está ocorrendo aqui, mais fácil será posteriormente.

Figura 14.1
Janelas



DIGITE **Listagem 14.1.** O texto completo de popupwindow.java.

```
1: import java.awt.*;
2:
3: public class PopUpWindow extends java.applet.Applet {
4:     Frame window;
5:     Button open, close;
6:
7:     public void init( ) {
8:         open = new Button("Open Window");
9:         add(open);
10:        close = new Button("Close Window");
11:        add(close);
12:
13:        window = new BaseFramel("A Pop Up Window");
14:        window.resize(150,150);
15:    }
16:
17:    public boolean action(Event evt, Object arg) {
18:        if (evt.target instanceof Button) {
19:            String label = (String)arg;
20:            if (label.equals("Open Window")) {
21:            if (!window.isShowing( ))
22:                 window.show( );
23:        } else {
24:            if (window.isShowing( ))
25:                window.hide( );
26:        }
```

Listagem 14.1. Continuação

```
27:             return true;
28:         } else
29:             return false;
30:     }
31: }
```


DIGITE

Listagem 14.2. O texto completo de baseframel.java.

```
1: import java.awt.*;
2:
3: class BaseFramel extends Frame {
4:       String message = "This is a Window";
5:       Label 1;
6:
7:    BaseFrame1(String    title) {
8:           super(title);
9:           setLayout(new BorderLayout( ));
10:
11:          1 = new Label(message, Label.CENTER);
12:          l.setFont(new Font("Helvetica", Font.PLAIN, 12));
13:          add("Center", 1);
14:      }
15:
16:      public Insets getInsets( ) {
17:      return new Insets(20,0,25,0);
18:      }
19: }
```

Depois que essas duas classes tiverem sido compiladas, o applet poderá ser testado com o seguinte código HTML:

```
<applet code="PopUpWindow.class" height=200 width=200>
</applet>
```

Duas classes constituem esse exemplo: a primeira, PopUpWindow, é a classe de applet que cria e controla a janela instantânea. No método init( ) dessa classe e, em particular, nas linhas 7 a 15 da listagem 14.1, você insere dois botões de controle no applet, para gerenciar a janela; em seguida, você cria, redimensiona e apresenta a janela em si.

O controle nesse applet ocorre quando um dos botões é pressionado. O método action( ), nas linhas 17 a 30 da listagem 14.1, manipula esses cliques de botão, que geram eventos de ação. Nesse método, o botão Open Window simplesmente mostra a janela, se ela estiver oculta (linhas 20 a 22 da listagem 14.1), e a ocultam se estiver sendo apresentada (linhas 23 a 25).

A própria janela instantânea é um tipo especial de quadro chamado BaseFramel. Nesse exemplo, o quadro é bastante simples; ele usa um BorderLayout e apresenta um rótulo no centro. Observe que a inicialização do quadro ocorre em um construtor e não em um método `init( )`. Como os quadros são objetos normais e não applets, você precisa inicializá-los de uma maneira mais convencional.

No construtor de BaseFramel, observe que a primeira linha (linha 8) é uma chamada ao construtor de sua superclasse. Como você aprendeu no Dia 6, o primeiro passo para inicializar uma nova classe é fazer essa chamada. Não se esqueça desse passo em suas próprias classes; você nunca sabe as coisas importantes que sua superclasse pode estar fazendo nesse construtor.

Caixas de diálogo

As caixas de diálogo são funcionalmente semelhantes aos quadros, no sentido de que apresentam novas janelas na tela. Entretanto, as caixas de diálogo são destinadas a serem usadas para janelas transientes — janelas que permitem receber alertas, que solicitam informações específicas etc.

NOVO TERMO Normalmente, as caixas de diálogo não possuem barras de título nem muitos dos recursos mais gerais que as janelas possuem (embora você possa criar uma caixa de diálogo com uma barra de título). Elas podem ser feitas sem possibilidade de redimensionamento, caso em que podem ser chamadas de *modais*. (As caixas de diálogo modais impedem a entrada de quaisquer outras janelas na tela até que sejam dispensadas.)

Os *diálogos* são janelas transientes destinadas a alertar o usuário sobre algum evento ou a obter entrada do usuário. Ao contrário dos quadros, os diálogos geralmente não possuem barras de título ou caixas de fechamento.

Um diálogo modal impede a entrada de qualquer uma das outras janelas na tela até que essa caixa de diálogo seja dispensada. Você não poderá apresentar outras janelas ou transformar uma janela de caixa de diálogo modal em ícone; você precisa realmente dispensar a caixa de diálogo modal, antes de poder fazer qualquer outra coisa no sistema. Normalmente, os avisos e alertas são caixas de diálogo modais.

O AWT fornece dois tipos de caixas de diálogo: a classe Dialog, que fornece uma caixa de diálogo genérica, e FileDialog, que produz a caixa de diálogo de localização de arquivos específica da plataforma.

- Dialog(Frame, *boolean)* cria uma caixa de diálogo invisível ligada ao quadro atual, que é modal (true) ou não (false).
- Dialog(Frome, *String, boolean)* cria uma caixa de diálogo invisível com o título fornecido, que é modal (true) ou não (false).

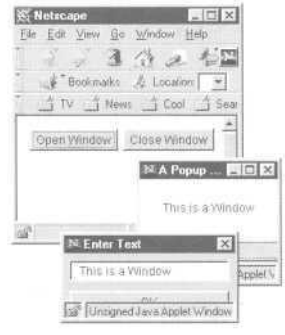
NOVO TERMO

A *janela de diálogo,* assim como a janela de quadro, é um painel em que você pode dispor e desenhar componentes da interface com o usuário e efetuar operações gráficas, exatamente como faria com qualquer outro painel. Assim como acontece com as outras janelas, o diálogo é inicialmente invisível, mas você pode apresentá-lo com show( ) e ocultá-lo com hide( ).

Adicione um diálogo ao exemplo da janela instantânea. Das três classes desse applet, BaseFrame2 é a única que precisa de alteração. Aqui, você modifica a classe para incluir um botão Set Text e inserir uma nova classe, TextDialog, que produz um diálogo de entrada de texto semelhante ao que aparece na Figura 14.2.

Figura 14.2

A caixa de diálogo Enter Text.



Nota

Esse projeto é uma expansão do anterior. Para evitar a sobrescrita dos arquivos-fonte do último projeto, faça uma cópia de BaseFramel.java chamada BaseFrame2.java e uma cópia de PopUpWindow.java, chamada PopUpWindowDiaiog.java. Use essas cópias no próximo projeto.

As alterações são pequenas ao se inserir o diálogo na classe BaseFrame2. Primeiramente, o nome da classe deve ser alterado de BaseFramel para BaseFrame2 e o construtor deve ser renomeado para BaseFrame2 (String title). Em seguida, você precisa de uma variável de instância para conter o diálogo, pois faz referência a ele em toda essa classe:

```
TextDiaiog dl;
```

No método construtor de BaseFrame2, você pode criar o diálogo (uma instância da nova classe TextDialog que você vai criar daqui a pouco), atribuí-lo à variável de instância dl e redimensioná-lo; o redimensionamento é mostrado

nas próximas duas linhas de código. Você não quer apresentá-lo ainda, pois ele só deve aparecer quando o botão Set Text receber um clique de mouse.

```
dl = new TextDialog(this, "Enter Text", true);
dl.resize(150,150);
```

Agora, crie o botão Set Text para funcionar de modo semelhante aos outros botões e inclua-o no BorderLayout na posição "South" (o que o coloca imediatamente abaixo do rótulo).

```
Button b = new Button("Set Text");
add("South", b);
```

Após ter inserido o TextDialoge um botão Set Text na classe BaseFrame2, você precisa incluir o seguinte método de tratamento de eventos:

```
public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
        dl.show( );
        return true;
    } else
        return false;
 }
```

Isso apresenta o objeto dl de TextDialog, quando qualquer botão do quadro receber um clique de mouse. Nesse exemplo, existe apenas um botão — Set Text.

Esse é o final do comportamento que você precisa incluir na janela instantânea para criar um diálogo. Apenas duas alterações são necessárias em PopUpWindowDialog. Primeiramente, o nome da classe deve ser mudado de PopUpWindow para PopUpWindowDiaiog. Em seguida, a referência deve ser feita à classe BaseFrame2, em vez de a BaseFramel, como ilustrado na instrução a seguir:

```
window = new BaseFrame2("A Pop Up Window");
```

O restante do novo comportamento fica na classe TextDialog, cujo código aparece na listagem 14.3.

DIGITE

Listagem 14.3. O texto completo de textdialog.java.

```
1: import java.awt.*;
2:
3: class TextDialog extends Dialog {
4:     TextField tf;
5:     BaseFrame2 theFrame;
6:
7:     TextDialog(Frame parent, String title, boolean modal) {
8:         super(parent, title, modal);
9:
10:         theFrame = (BaseFrame2)parent;
11:         setLayout(new BorderLayout(10,10));
12:         setBackground(Color.white);
```

Listagem 14.3. Continuação

```
13:             tf = new TextField(theFrame.message,20);
14:             add("Center", tf);
15:
16:            Button b = new Button("0K");
17:             add("South", b);
18:         }
19:
20:         public Insets insets( ) {
21:             return new Insets(30,10,10,10);
22:         }
23:
24:          public boolean action(Event evt, Object arg) {
25:               if (evt.target instanceof Button) {
26:                   String label = (String)arg;
27:                   if (label == "0K") {
28:                        hide( );
29:                        theFrame.l.setText(tf.getText( ));
30:                   }
31:                   return true;
32:               } else
33:                   return false;
34:         }
35: }
```


Você deve observar alguns detalhes a respeito desse código. Primeiramente, diferente das outras duas janelas desse applet, o tratamento de eventos ocorre dentro da classe para que o diálogo sirva como sua própria rotina de tratamento.

A despeito desse fato, esse diálogo possui muitos dos mesmos elementos que a classe BaseFrame2. Observe que o construtor de TextDialog é idêntico ao dos construtores de sua superclasseDialog, pois, a despeito do fato de TextDi a-log estar vinculado a um objeto cuja classe é BaseFrame2, os diálogos devem estar vinculados a um objeto Frame real. Você pode facilmente tornar o construtor mais genérico e depois especializá-lo, após o construtor da superclasse ter sido chamado — o que é precisamente o que se faz nas linhas 8 e 10 da listagem 14.3. A linha 8 é a chamada ao construtor da superclasse para ligar o diálogo ao quadro, e a linha 10 estabelece realmente a variável de instância como a instância específica da classe Frame definida na classe BaseFrame2.

O restante do construtor TextDialog simplesmente define o resto do layout: um campo de texto e um botão em um layout de borda. O método getlnsets( ) acrescenta algumas insets e o método action( ), que manipula a ação do botão OK do diálogo. O método action( ) faz duas coisas: na linha 28, ele oculta o diálogo para dispensá-lo, e na linha 29, ele muda o valor do rótulo no quadro de origem para ser o novo valor do texto.

Todas essas classes apenas para um simples applet! As diferentes janelas e classes de evento associadas compõem o applet complicado. Nesse ponto, contudo, você deve se sentir à vontade a respeito de como cada parte de um applet possui seus próprios componentes e ações, como todas as partes dele se encaixam.

Ligando diálogos a applets

Os diálogos podem ser ligados apenas a quadros. Para criar um diálogo, você precisa passar uma instância da classe Frame para um dos métodos construtores do diálogo. Isso implica que você não pode criar caixas de diálogo que estejam ligadas a applets. Como os applets não possuem quadros explícitos, você não pode fornecer à classe Dialog um argumento de quadro. No entanto, através de um trecho de código inteligente, você pode obter o objeto quadro que contém esse applet (freqüentemente, a própria janela de visualização do navegador ou applet) e depois utilizá-lo como quadro do diálogo.

Esse código inteligente faz uso do método getParent( ), definido para todos os componentes do AWT. O método getParent ( ) retorna o objeto que contém esse objeto. A origem de todos os aplicativos AWT, então, deve ser um quadro. Os applets se comportam da mesma maneira. Chamando getParent ( ) repetidamente, finalmente você deve obter uma instância de Frame. Aqui está o código inteligente que você pode colocar dentro de seu applet:

```
Object anchorpoint = getParent( )
while (! (anchorpoint instanceof Frame))
    anchorpoint = ( (Component) anchorpoint ).getParent( );
```

Na primeira linha desse código, você cria uma variável local, chamada anchorpoint, para conter o quadro final desse applet. O objeto atribuído a anchorpoint pode ser uma de muitas classes; portanto, declare seu tipo como Object.

As duas linhas seguintes desse código são um loop while que chama getParent ( ) em cada diferente objeto do encadeamento, até chegar a um objeto Frame real. Observe aqui que, como o método getParent ( ) é definido apenas em objetos que herdam de Component, você sempre precisa fazer a coerção do valor de anchorpoint para Component para que o método getParent( ) funcione.

Depois que o loop terminar, o objeto contido na variável anchorpoi nt será uma instância da classe Frame (ou uma de suas subclasses). Você pode então criar um objeto Dialog ligado a esse quadro, fazendo mais uma vez a coerção de anchorpoint para garantir que possui um objeto Frame:

```
TextDialog dl = new TextDialog((Frame)anchorpoint,
    "Enter Text", true);
```

Objetos de diálogo de arquivo

A classe FileDialog fornece uma caixa de diálogo File Open/Save básica que permite acessar o sistema de arquivos local. A classe Fi1eDialog é independente de sistema, mas dependente da plataforma, o diálogo Open File ou Save File padrão é apresentado.

O fato de você poder usar instâncias de FileDialog para applets é dependente do navegador. Devido às restrições de segurança padrão vigentes para os applets, a maior parte dos navegadores produz uma exceção de segurança quando você experimenta. FileDialog tem muito mais utilidade em aplicativos independentes.

Use os seguintes construtores para criar um diálogo de arquivo:

- FileDialog(Frame, *String)* cria um diálogo de arquivo, ligado ao quadro dado, com o título fornecido. Essa forma cria um diálogo para carregar um arquivo.
- Fi1eDialog(Frame, *String, int)* também cria um diálogo de arquivo, mas o argumento de inteiro é usado para determinar se o diálogo serve para carregar ou salvar um arquivo. (A única diferença são os rótulos nos botões; o diálogo de arquivo não abre ou salva nada realmente.) As opções do argumento de modo são FileDialog.LOAD e FileDialog. SAVE.

Após criar uma instância de Fi1eDialog, use show( ) para apresentá-la:

```
FileDialog fd = new Fi1eDialog(this, "FileDialog");
fd.show( );
```

Quando o leitor escolhe um arquivo no diálogo File e o dispensa, você pode então acessar o nome de arquivo que o leitor escolheu, usando os métodos getDirectory( ) egetFile( ). Esses dois métodos retornam strings indicando os valores que o leitor escolheu. Você pode, então, abrir o arquivo, usando os métodos de tratamento de fluxo e arquivo (sobre os quais você aprenderá na próxima semana) e depois ler ou escrever nesse arquivo.

Eventos de janela

Você está no último conjunto de eventos dos quais pode tratar no AWT: os eventos para janelas e diálogos. (Em termos de eventos, um diálogo é considerado apenas outro tipo de janela.) Os eventos de janela são o resultado de quando o estado de uma janela muda de alguma maneira: quando a janela é movida, redimensionada, transformada em ícone, quando sai do estado de ícone, quando é trazida para a frente ou quando é fechada. Em um aplicativo bem-comportado, você desejará tratar pelo menos de alguns desses eventos —

por exemplo, para interromper a execução de threads quando uma janela é transformada em ícone ou para fazer a limpeza quando ela é fechada.

Você pode usar handleEvent( ) para testar cada um dos eventos apresentados na Tabela 14.1, usando a instrução switch padrão com a variável de instância id.

Tabela 14.1 Eventos de janela.

Nome do evento	*Quando ele ocorre*
WINDOW_DESTROY	Gerado quando uma janela é destruída usando-se a caixa Close ou o item de menu Close.
WINDOW_EXPOSE	Gerado quando a janela é trazida para frente das outras janelas.
WINDOW_ICONIFY	Gerado quando a janela é transformada em ícone.
WINDOW_DEICONIFY	Gerado quando a janela é restaurada do estado de ícone.
WINDOW_MOVED	Gerado quando a janela é movida.

Menus

Resta falarmos apenas de um elemento da interface com o usuário do AWT: os menus.

Uma barra de menus é um conjunto de menus. Um menu, por sua vez, é composto de um conjunto de itens de menu, que possuem nomes e, às vezes, atalhos opcionais. O AWT fornece classes para todos esses elementos de menu, incluindo MenuBar, Menu e MenuItem.

Menus e barras de menu

NOVO TERMO

Uma *barra de menus* é um conjunto de menus que aparece na parte superior de uma janela. Como sua raiz reside em janelas, você não pode criar barras de menu em applets (mas se esse applet apresentar uma janela independente, ela poderá ter uma barra de menus).

Para criar uma barra de menus para determinada janela, você gera uma nova instância da classe MenuBar:

```
MenuBar mbar = new MenuBar( );
```

Você usa o método setMenuBar( ) (definido na classe Frame) para definir essa barra de menus como o menu padrão da janela:

```
window.setMenuBar(mbar);
```

Você pode incluir menus específicos (File, Edit etc) na barra de menus, criando-os e depois inserindo-os na barra, usando add( ). O argumento do construtor Menu é o nome do menu, conforme ele aparece na barra.

```
Menu myMenu = new Menu("File");
mbar.add(myMenu);
```

Alguns sistemas fornecem um menu Help especial, que é desenhado no lado direito da barra de menus (e não no lado esquerdo). Você pode indicar que um menu específico é de Help usando o método setHel pMenu ( ). O menu dado já deve estar inserido no próprio menu, antes de se tornar um menu Help.


```
Menu heipmenu = new Menu("Help");
mbar.add(helpmenu);
mbar.setHelpMenu(helpmenu);
```

Se, por qualquer motivo, você desejar impedir que um usuário selecione um menu, então, poderá usar o comando disable( ) nesse menu (e o comando enable( ) para torná-lo novamente disponível):

```
myMenu.disabieí );
```

Itens de menu

Você pode adicionar quatro tipos de itens aos menus individuais:

- Instâncias da classe Menu Item, para itens de menu normais.
- Instâncias da classe CheckBoxMenuItem, para itens de menu de alternância.
- Outros menus, com seus próprios itens.
- Separadores, para linhas que separam grupos de itens em menus.

Criando itens de menu

Os itens de menu normais são criados e inseridos em um menu usando-se a classe Menultem. Primeiramente, crie uma nova instância de Menultem e depois insira-a no componente Menu usando o método add( ):

```
Menu myMenu = new Menu("Tools");
myMenu.add(new  Menultem("lnfo"));
myMenu.add(new  MenuItem("Colors"));
```

Submenus podem ser incluídos simplesmente criando-se uma nova instância de Menu e inserindo-o no primeiro menu. Você pode, então, inserir itens nesse menu:

```
Menu submenu = new Menu("Sizes");
myMenu.add(submenu);
submenu.add(new  MenuItem("Smal1"));
submenu.add(new  MenuItem("Medium"));
submenu.add(new  MenuItem("Large"));
```

A classe CheckBoxMenuItem cria um item de menu contendo uma caixa de seleção, permitindo que o estado do menu alterne entre ativo e desativado. (Selecioná-la uma vez faz a caixa de seleção aparecer marcada; selecioná-la

novamente anula a seleção da caixa.) Crie e insira um item de menu de caixa de seleção, da mesma maneira que você cria e inclui itens de menu normais:

```
CheckboxMenuItem coords =
new CheckboxMenuItem("Show Coordinates");
myMenu.add(coords);
```

Finalmente, para adicionar um separador em um menu (uma linha utilizada para separar grupos de itens em um menu), crie e insira um item de menu com um traço simples (-) como rótulo. Esse item de menu especial será desenhado com uma linha separadora. As duas linhas de código Java a seguir criam um item de menu separador e o inserem no menu myMenu:

```
MenuItem msep = new MenuItem("-");
myMenu.add(msep);
```

Todo item de menu pode ser desativado usando-se o método disab1e( ) e novamente ativado usando-se enable( ). Os itens de menu desativados não podem ser selecionados.

```
MenuItem item = new MenuItem("Fill");
myMenu.addItem(item);
item.disable( );
```

Eventos de menu

A ação de selecionar um item de menu com o mouse ou escolher o atalho de teclado do item faz um evento de ação ser gerado. Você pode tratar desse evento usando o método action( ), exatamente como fez nos dois últimos dias.

Além dos eventos de ação, CheckBoxMenuItems gera eventos de seleção de lista e anulação de seleção de lista, que podem ser tratados através de handleEvent( ).

Quando você processar eventos gerados por itens de menu e itens de menu de caixa de seleção, lembre-se de que, como CheckboxMenuItem é uma subclasse de MenuItem, não é preciso tratar desse item como um caso especial. Você pode tratar dessa ação da mesma maneira que trata outros métodos de ação.

Criando oplicativos AWT independentes

Embora você aprenda a criar interfaces gráficas com o usuário para aplicativos na próxima semana, usando as novas classes Swing, já possui a maior parte dos conhecimentos necessários para gerar um aplicativo Java 1.02.

Na verdade, não há muita diferença entre um applet Java e um aplicativo gráfico Java. Tudo o que foi aprendido até agora a respeito do AWT, incluindo os métodos gráficos, técnicas de animação, eventos, componentes de interface com o usuário, janelas e diálogos, pode ser usado em aplicativos Java da mesma maneira que nos applets.

Como você faz para criar um aplicativo gráfico Java? O código para criá-lo é quase trivial. Sua classe de aplicativo principal deve herdar de Frame. Se ele usar threads (para animação ou outro processamento), então também deverá implementar Runnabl e:

```
class MyAWTApplication extends Frame implements Runnable {
    // ...
}
```

Você cria uma nova instância de sua classe dentro do método mai n( ) de seu aplicativo — como sua classe estende Frame, isso fornecerá uma nova janela AWT que você pode então redimensionar e apresentar como faria com qualquer janela AWT.

Configure os recursos normais do AWT para uma janela, como você normalmente poderia fazer em um método init( ) para um applet dentro do método construtor de sua classe: defina o título, adicione um gerenciador de layout, crie e adicione componentes, como uma barra de menus ou outros elementos de interface com o usuário, inicie um thread etc.

A seguir há um exemplo de um aplicativo bastante simples:

```
import java.awt.*;
class MyAWTApplication extends Frame {
    MyAWTApplication(String title) {
      super(title);
      setLayout(new FlowLayout( ));
      addfnew Button("OK"));
      add(new Button("Reset"));
      add(new Button("Cancel"));
    }
    public static void main(String args[ ]) {
      MyAWTApplication app = new MyAWTApplication("Hi! I'm an application");
      app.resize(300,300);
      app.show( );
    }
}
```

De modo geral, você pode usar qualquer um dos métodos sobre os quais aprendeu nesta semana, para controlar e gerenciar seu aplicativo. Os únicos métodos que você não pode usar são aqueles específicos dos applets (ou seja, aqueles definidos em java. appl et. Appl et, que incluem métodos para recuperar informações de URL e reproduzir clipes de áudio).

Você deve saber de uma outra diferença entre aplicativos e applets: quando você lidar com um evento de fechamento de janela, além de ocultar ou destruir a janela, também deve chamar System.exit(O) para sinalizar para o sistema que seu aplicativo saiu.

```
public void windowClosing(WindowEvent e) {
    win.hide( );
    win.destroy( );
    System.exit(0);
}
```

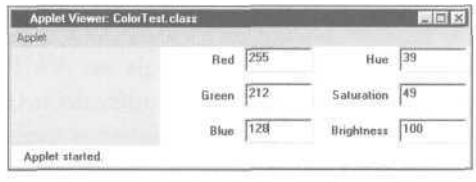
Um exemplo completo: conversor RGB para HSB

Como uma oportunidade de utilizar bem o material dos dias anteriores, o seguinte exemplo de applet demonstra a criação de layout, painéis aninhados, criação de interface e tratamento de eventos.

A Figura 14.3 mostra o applet que você cria nesse exemplo. O applet ColorTest permite que você escolha cores baseadas nos espaços de cor sRGB e HSB — que descreve as cores com base em seu conteúdo de vermelho, verde e azul ou valores de matiz, saturação e brilho, respectivamente.

Figura 14.3

O *applet ColorTest.*



O applet ColorTest possui três partes principais: uma caixa colorida no lado esquerdo e dois grupos de campos de texto à direita. O primeiro grupo indica valores RGB; o segundo grupo, valores HSB. Se você mudar qualquer um dos valores em qualquer uma das caixas de texto, a caixa colorida será atualizada para a nova cor, assim como os valores do outro grupo de caixas de texto.

Esse applet utiliza duas classes:

- ColorTest, que herda de Applet. Essa é a classe de controle para o próprio applet.
- ColorControls, que herda de Panei. Você cria essa classe para representar um grupo de três campos de texto e para tratar de suas ações. Duas instâncias dessa classe, uma para os valores sRGB e outra para os valores HSB, são criadas e incluídas no applet.

Como ele é muito complicado e pode ser confuso, vamos trabalhar nesse exemplo passo a passo. Todo o código desse applet está apresentado no final desta seção.

Proietando e criando o lavout do applet

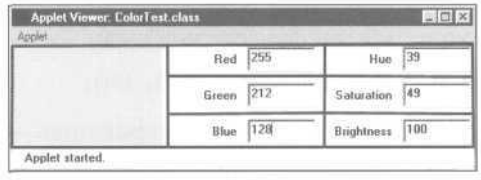
A melhor maneira de começar a criar um applet que utiliza componentes AWT é primeiro se preocupar com o layout e depois com a funcionalidade. Ao tratar do layout, você deve começar com o painel mais externo e daí para dentro.

Fazer um rascunho de seu design de interface com o usuário pode ajudá-lo a organizar os painéis dentro de seu applet ou janela, para melhor tirar proveito do layout e do espaço. Os designs em papel são úteis mesmo quando você não está usando layouts de grade de conteúdo, mas duplamente úteis quando está. (Você vai usar um layout de grade simples para esse applet.)

A Figura 14.4 mostra o applet ColorTest com uma grade desenhada sobre ele para que você possa ter uma idéia de como os painéis e painéis incorporados funcionam.

Figura 14.4

Os *painéis* e *componentes do applet ColorTest.*



Comece com o painel mais externo — o próprio applet. Esse painel possui três partes: a caixa de cor à esquerda, os campos de texto RGB no meio e os campos HSB à direita.

Como o painel mais externo é o próprio applet, a classe ColorTest será a classe do applet e herdará de Applet. Você também pode importar as classes AWT aqui. (Observe que, como você utiliza muitas delas neste programa, importar o pacote inteiro é mais fácil.)

```
import java.awt.*;

public class ColorTest extends java.applet.Applet {
    // ...
}
```

Esse applet tem três elementos principais para controlar a caixa de cor e os dois subpainéis. Cada um dos dois subpainéis faz referência a itens diferentes, mas basicamente eles são o mesmo painel e se comportam da mesma maneira. Em vez de duplicar código nessa classe, você pode aproveitar essa oportunidade para criar outra classe estritamente para os subpainéis, usar instâncias dessa classe no applet e fazer a comunicação entre tudo o que está usando métodos. A nova classe, chamada ColorControl s, será definida daqui a pouco.

Por enquanto, entretanto, você sabe que precisa manter uma alça para as três partes do applet, para que possa atualizá-las quando elas mudarem. Crie três variáveis de instância: uma do tipo Canvas para a caixa de cor e as outras duas de tipo ColorControls, para os painéis de controle:

```
ColorControls RGBcontrols, HSBcontrols;
Canvas swatch;
```

Agora, você pode passar para o método i n i t ( ), onde toda a inicialização básica e o layout do applet ocorrem. Siga estas três etapas para inicializar o applet:

1. Crie o layout das partes grandes do painel. Embora um layout de fluxo possa funcionar, criar um layout de grade com uma linha e três colunas é uma idéia muito melhor.

2. Crie e inicialize os três componentes desse applet: um canvas para a caixa de cor e dois subpainéis para os campos de texto.
3. Adicione esses componentes ao applet.

A etapa 1 é o layout. Use um layout de grade e um intervalo de 10 pontos para separar cada um dos componentes:

```
setLayout(new GridLayout(l, 3, 5, 15));
```

A etapa 2 é a criação dos componentes — primeiro o canvas. Você tem uma variável de instância para contê-lo. Aqui, você cria o canvas e inicializa seu fundo com a cor preta:

```
swatch = new Canvas( );
swatch.setBackground(Color.black);
```

Você também precisa criar duas instâncias dos painéis ColorControl s, ainda não existentes aqui. Como você ainda não criou a classe, não sabe como serão os construtores dessa classe. Nesse caso, coloque alguns construtores marcadores aqui; você pode completar os detalhes posteriormente.

```
RGBcontrols = new ColorControls(...);
HSBcontrols = new ColorControls(...);
```

A etapa 3 é a adição dos três componentes no painel do applet, como segue:

```
add(swatch);
add(RGBcontrols);
add(HSBcontrols);
```

Acrescente insets ao applet, enquanto estiver trabalhando no layout (10 pontos ao longo de todas as margens):

```
public Insets getInsets( ) {
    return new Insets(10, 10, 10, 10);
}
```

Tudo bem até aqui? Nesse ponto, você deve ter três variáveis de instância, um método `init( )` com dois construtores incompletos e um método `getInsets( )` em sua classe ColorTest. Passemos agora para a criação do layout do subpainel na classe ColorControls, para que você possa completar esses construtores e concluir o layout.

Definindo os suboainéis

A classe ColorControls terá o comportamento de organizar e manipular os subpainéis que representam os valores RGB e HSB da cor. Col orControl s não precisa ser uma subclasse de Appl et, pois não se trata na verdade de um applet; ele é apenas um painel. Defina-o para herdar de Panel:

```
import java.awt.*;

class ColorControls extends Panei {
    // •••
}
```


A classe Col orControl s precisa de diversas variáveis de instância para que as informações do painel possam voltar ao applet. A primeira dessas variáveis de instância é um gancho para a classe do applet que contém esse painel. Como a classe mais externa do applet controla a atualização de cada painel, esse painel precisa de um modo de informar ao applet de que algo mudou. Para chamar um método nesse applet, você precisa de uma referência ao objeto; a variável de instância número um é uma referência a uma instância da classe Col orTest:

```
ColorTest applet;
```

Se você descobrir que a classe applet está atualizando tudo, essa classe estará interessada nos campos de texto específicos desse subpainel. Você cria variáveis de instância para esses campos de texto:

```
TextField tfieldl, tfield2, tfield3;
```

Agora, você pode passar para o construtor dessa classe. Como essa classe não é um applet, você não usará `init( )` para ínicializá-la; em vez disso, usará um método construtor. Dentro do construtor, você realiza grande parte do que realizou dentro de `init( )`: cria o layout do subpainel, cria os campos de texto e os insere no painel.

O objetivo aqui é tornar a classe Col orControl s suficientemente genérica para que você possa utilizá-la para campos RGB e campos HSB. Esses dois painéis diferem apenas em um aspecto: os rótulos do texto — esses são três valores a serem obtidos, antes que você possa criar o objeto. Você pode passar esses três valores através dos construtores em Col orTest. Você também precisa de mais um: a referência ao applet subjacente, que você também pode obter a partir do construtor.

Agora, você tem quatro argumentos para o construtor básico da classe Col orControl s. A seguir, há a assinatura do construtor:

```
ColorControls(ColorTest parent,
    String 11, String 12, String 13) {
}
```

Inicie esse construtor primeiramente definindo o valor de parent para variável de instância appl et:

```
applet = parent;
```

Em seguida, crie o layout desse painel. Você também pode usar um layout de grade para esses subpainéis, como foi feito no painel do applet, mas desta vez a grade terá três linhas uma para cada campo de texto e pares de rótulo) e duas colunas (uma para os rótulos e uma para os campos). Além disso, defina um intervalo de 10 pontos entre os componentes da grade:

```
setLayout(new GridLayout(3,2,10,10));
```

Agora, você pode criar e incluir os componentes no painel. Primeiramente, crie os objetos campo de texto (inicializados com o stnng "0") e os atribua às variáveis de instância apropriadas:

```
tfieldl = new TextField("0");
tfield2 = new TextField("0");
tfield3 = new TextField("0");
```

Agora, insira esses campos e os rótulos apropriados no painel, usando os três parâmetros restantes do construtor como texto dos rótulos:

```
addfnew Label(ll, Label .RIGHT));
add(tfieldl);
add(new Label(12, Label.RIGHT));
add(tfield2);
add(new Label(13, Label.RIGHT));
add(tfield3);
```

Você terminou o construtor da classe de subpainel ColorControls. O layout está terminado? Não totalmente. Você também acrescenta uma inclusão no subpainel — apenas nas margens superior e inferior — para aprimorar o layout. Acrescente a inclusão aqui, como fez na classe ColorTest, usando o método getlnsets( ):

```
public Insets getInsets( ) {
    return new Insets(10, 10, 0, 0);
}
```

Está quase acabando. Você tem 98% da estrutura básica pronta, mas resta uma etapa: voltar a ColorTest e corrigir os marcadores para construtores do subpainel, para que eles se adeqüem aos construtores reais de ColorControls.

O construtor de ColorControls que você acabou de criar possui agora quatro argumentos: o objeto ColorTest e três rótulos (strings). Lembre-se de quando você criou o método init( ) para ColorTest. Você inseriu dois marcadores para a criação de novos objetos ColorControls. Substitua agora esses marcadores pelas versões corretas. Certifique-se de incluir os quatro argumentos de que o construtor precisa para funcionar: o objeto Col orTest e três strings. Você pode usar a palavra-chave thi s para passar o objeto Col orTest para esses construtores:

```
RGBcontrols = new ColorControls(this, "Red",
 "Green", "Blue");
HSBcontrols = new ColorControls(this, "Hue",
 "Saturation", "Brightness");
```

Nota

O número 0 (na verdade, o string "0" é usado para os valores iniciais de todos os campos de texto deste exemplo. Para a cor preta, os valores RGB e HSB são 0 e esse é o motivo de essa suposição poder ser feita. Se você quiser inicializar o applet com alguma outra cor, talvez queira reescrever a classe ColorControls para usar valores inicializadores, bem como para inicializar rótulos. O modo como isso foi feito constitui um exemplo mais curto.

Tratando dos eventos

Após criar o layout, você define ações com os componentes da interface com o usuário para que o applet possa responder quando o usuário interagir com ele.

A ação desse applet ocorre quando o usuário muda um valor em qualquer um dos campos de texto. Causando uma ação em um campo de texto, a cor muda, a caixa de cor é atualizada para a nova cor e o valor dos campos do subpainel oposto muda para refletir a nova cor.

A classe ColorTest é responsável por fazer realmente a atualização, pois ela controla todos os subpainéis. Entretanto, você deve controlar e interceptar eventos no subpainel em que eles ocorrem. Como a ação do applet é uma ação de texto real, você pode usar um método acti on ( ) para interceptá-la na classe ColorControl s:

```
public boolean action(Event evt, Object arg) {

   if (evt.target instanceof TextField) {
      applet.update(this);
      return true;
    }
    else return false;
1
```

No método action( ), você faz um teste para certificar-se de que a ação foi realmente gerada por um campo de texto (como existem apenas campos de texto disponíveis, essa é a única ação que você obterá; de qualquer modo, é uma boa idéia testar isso). Se assim for, chame o método update( ) definido em Col orTest, para atualizar o applet para refletir os novos valores. Como o applet mais externo é responsável por realizar toda a atualização, é exatamente por isso que você precisa desse gancho de volta para o applet — para que você possa chamar o método correto no momento certo.

Atualizando o resultado

Agora vem a parte difícil: a atualização baseada nos novos valores de qualquer campo de texto foi realmente mudada. Para esta etapa, você define o método update( ) na classe ColorTest. Esse método update( ) recebe um único argumento: a instância Col orControl s que contém o valor alterado. (Você obtém o argumento dos métodos de evento no objeto ColorControls.)

Nota

Esse método update( ) não vai interferir no método update( ) do sistema? Não. Lembre-se de que os métodos podem ter o mesmo nome, mas diferentes assinaturas e definições. Como esse método update( ) possui um único argumento do tipo ColorControls, ele não interfere na outra versão de update( ). Normalmente, todos os métodos chamados update( ) devem significar basicamente a mesma coisa. Não é o caso, aqui, mas esse é apenas um exemplo.

O método update( ) é responsável por atualizar todos os painéis no applet. Para saber qual painel deve atualizar, você precisa saber qual deles mudou. Você pode descobrir isso fazendo um teste para ver se o argumento que passou do painel é o mesmo dos subpainéis que armazenou nas variáveis de instância RGBcontrol s e HSBcontrol s:

```
void update(ColorControls Control Panel) {
    if (controlPanel == RGBcontrols) { // RGB mudou, atualiza HSB
      // ...
    } else { // HSB mudou, atualiza RGB
      // ...
    }
}
```

Esse teste é a parte central do método update ( ). Comece com o primeiro caso — um número foi alterado nos campos de texto RGB. Agora, com base nesses novos valores RGB, você precisa gerar um novo objeto Col or e atualizar os valores no painel HSB. Você pode criar algumas variáveis locais para conter alguns valores básicos, para reduzir a digitação. Em particular, os valores dos campos de texto são strings, cujos valores você pode obter usando o método getText( ), definido nos objetos TextField do objeto Col orControls. Como na maioria das vezes você desejará tratar desses valores como inteiros nesse método, pode obter esses valores de string, convertê-los para inteiros e armazená-los em variáveis locais (value1, value2, value3). Aqui está o código para cuidar dessa tarefa (ele parece mais complicado do que realmente é):

```
int valuel = Integer.parseInt(controlPanel.tfieldl.getText( ));
int value2 = Integer.parseInt(controlPanel.tfield2.getText( ));
int value3 = Integer.parseInt(controlPanel.tfield3.getText( ));
```

Enquanto está definindo variáveis locais, você também precisa de uma para o novo objeto Color:

```
Color c;
```

Suponha agora que um dos campos de texto do lado RGB do applet tenha mudado e insira o código da parte if do método update( ). Você precisa criar um novo objeto Col or e atualizar o lado HSB do painel. Essa primeira parte é fácil. Dados os três valores RGB, você pode criar um novo objeto Col or usando esses valores como argumentos do construtor:

```
c = new Color(value1, value2, value3);
```

Nota

Essa parte do exemplo não é muito forte. Ela pressupõe que o usuário realmente introduziu inteiros de 0 a 255 nos campos de texto. Uma versão melhor faria um teste para certificar-se de que nenhum erro de entrada de dados ocorreu, mas esse exemplo foi mantido pequeno.

Agora, você converte os valores RGB em HSB. Os algoritmos padrão podem converter uma cor baseada em RGB em uma cor HSB, mas você não precisa pesquisá-los. A classe Color possui um método de classe chamado RGBtoHSB( ), que você pode usar. Esse método realiza o trabalho para você — a maior parte dele, pelo menos. Contudo, o método RGBtoHSB ( ) apresenta dois problemas:

- O método RGBtoHSB ( ) retorna um array dos três valores HSB; portanto, você precisa extrair esses valores do array.
- Os valores HSB são medidos em ponto flutuante, de 0.0 a 1.0. Eu prefiro pensar nos valores HSB como inteiros, onde o matiz é um valor graduado em uma roda de cores (de 0 a 360) e a saturação e o brilho são porcentagens de 0 a 100.

Nenhum desses problemas é insuperável; você precisa apenas inserir algumas linhas extras de código. Comece chamando RGBtoHSB ( ) com os novos valores RGB que possui. O tipo de retorno desse método é um array de f 1 oats, de modo que você cria uma variável local (HSB) para armazenar os resultados do método RBGtoHSB ( ). (Observe que você também precisa criar e passar um array de floats vazio como o quarto argumento para RGBtoHSB( ).)

```
float[ ] HSB = Color.RGBtoHSB(valuel, value2,
    value3, (new float[3]));
```

Agora, converta esses valores em ponto flutuante, que variam de 0.0 a 1.0, para valores que variam de 0 a 100 (para saturação e brilho) e de 0 a 360 para o matiz, multiplicando os números apropriados e reatribuindo o valor de volta para o array:

```
HSB[O] *= 360;
HSB[1] *= 100;
HSB[2] *= 100;
```

Agora você tem os números desejados. A última parte da atualização coloca esses valores de volta nos campos de texto. É claro que esses valores ainda são números em ponto flutuante; portanto, você precisa fazer a coerção deles para ints, antes de transformá-los em strings e armazená-los:

```
HSBcontrols.tfieldl.setText(String.valueOf((int)HSB[0]));
HSBcontrols.tfi eid2.setText(Stri ng.vaiueOf((i nt)HSB[1]));
HSBcontrols.tfield3.setText(String.valueOf((int)HSB[2]));
```

Você está na metade do caminho. A próxima parte do applet é a que atualiza os valores RGB quando um campo de texto no lado HSB mudou. Esse

é o else no enorme if...else que define esse método e determina o que vai ser atualizado se houver uma mudança.

Gerar os valores RGB a partir de valores HSB é na verdade mais fácil do que fazer o processo inverso. Um método de classe na classe Color, (getHSBCol or ( )), cria um novo objeto Col or a partir de três valores HSB. Após ter um objeto Color, você pode extrair os valores RGB facilmente dele. O problema, é claro, é que getHSBCol or recebe três argumentos em ponto flutuante e os valores que você tem são os inteiros que prefiro usar. Na chamada de getHSBCol or, você terá de fazer a coerção dos valores inteiros dos campos de texto para floats e dividi-los pelo fator de conversão correto. O resultado de getHSBCol or é um objeto Color. Portanto, você pode simplesmente atribuir o objeto à variável local c, para depois poder utilizá-lo novamente:

```
c = Color.getHSBColor((float)valuel / 360,
    (float)value2 / 100, (f1oat)value3 / 100);
```

Com o objeto Color totalmente definido, a atualização dos valores RGB envolve a extração desses valores do objeto Color. Os métodos getRed( ), getGreen( ) egetBlue( ),definidos nadasse Color, realizarão exatamente essa tarefa:

```
RGBcontrol s.tfi ei dl.setText(Stri ng.vaiueOf(c.getRed( )));
RGBcontrol s.tfiel d2.setText(Stri ng.vaiueOf(c.getGreen( )));
GBcontrols.tfield3.setText(String.value0f(c.getBlue( )));
```

Finalmente, independentemente do valor RGB ou HSB ter mudado, você precisa atualizar a caixa de cor da esquerda para refletir a nova cor. Como você possui um novo objeto Color armazenado na variável c, pode usar o método setBackground para mudar a cor. Observe também que setBackground não repinta automaticamente a tela; portanto, ative também o método repai nt ( ):

```
swatch.setBackground(c);
swatch.repaint( );
```

É isso! Você terminou. Agora, compile as classes ColorTest e ColorControl s, crie um arquivo HTML para carregar o applet ColorTest e verifique o resultado.

O códiqo-fonte comoleto

A listagem 14.4 mostra o código-fonte completo da classe de applet Col orTest e a listagem 14.5 mostra o código-fonte da classe auxiliar ColorControls. Descobrir o que está ocorrendo em um applet freqüentemente é mais fácil quando o código está todo em um só lugar e você pode acompanhar as chamadas de método e como os valores são transmitidos. Comece com o método init( ) do applet ColorTest e parta de lá.

DIGITE

Listagem 14.4. O texto completo de colortest.java.

```
1: import java.awt.*;
2:
3: public class ColorTest extends java.applet.Applet {
4:      ColorControls RGBcontrols, HSBcontrols;
5:      Canvas swatch;
6:
7:      public void init( ) {
8:           setLayout(new GridLayout(l, 3, 5, 15));
9:           swatch = new Canvas( );
10:          swatch.setBackground(Color.black);
11:          RGBcontrols = new ColorControls(this, "Red",
12:               "Green", "Blue");
13:          HSBcontrols = new ColorControls(this, "Hue",
14:               "Saturation", "Brightness");
15:          add(swatch);
16:          add(RGBcontrols);
17:          add(HSBcontrols);
18:     }
19:
20:     public Insets getInsets( ) {
21:          return new Insets(10, 10, 10, 10);
22:     }
23:
24:     void update(ColorControls controlPanel) {
25:          int valuel = Integer.parseInt(controlPanei.tfieldl.getText( ));
26:          int value2 = Integer.parseInt(controlPanei.tfield2.getText( ));
27:          int value3 = Integer.parseInt(controlPanei.tfield3.getText( ));
28:          Color c;
29:          if (controlPanel == RGBcontrols) { // RGB mudou, atualiza HSB
30:               c = new Color(valuel, value2, value3);
31:               float[ ] HSB = Color.RGBtoHSB(valuel, value2,
32:               value3, (new float[3]));
33:               HSB[O] *- 360;
34:               HSB[1] *- 100;
35:               HSB[2] *= 100;
36:               HSBcontrols.tfieldl.setText(String.valueOf((int)HSB[O]));
37:               HSBcontrols.tfield2.setText(String.value0f((int)HSB[l]));
38:               HSBcontrols.tfield3.setText(String.value0f((int)HSB[2]));
39:          } else { // HSB mudou, atualiza RGB
40:               c = Color.getHSBColor((float)valuel / 360,
41:                    (float)value2 / 100, (float)value3 / 100);
42:          RGBcontrols.tfieldl.setText(Stri      ng.vaiueOf(c.getRed( )));
43:               RGBcontrols.tfield2.setText(String.value0f(c.getGreen( )));
44:               RGBcontrols.tfield3.setText(String.value0f(c.getBlue( )));
45:          }
46:          swatch.setBackground(c);
47:          swatch.repaint( );
48:     }
49: }
```

DIGITE

Listagem 14.5. O texto completo de colorcontrols.java.

```
1: import java.awt.*;
2:
3: class ColorControls extends Panei {
4:       ColorTest applet;
5:       TextField tfieldl, tfield2, tfield3;
6:
7:       ColorControls(ColorTest parent,
8:           String 11, String 12, String 13) {
9:
10:          applet = parent;
11:          setLayout(new GridLayout(3,2,10,10));
12:          tfieldl = new TextField("0");
13:          tfield2 = new TextField("0");
14:          tfield3 = new TextField("0");
15:          add(new Label(ll, Label.RIGHT));
16:          add(tfieldl);
17:          add(new Label(12, Label.RIGHT));
18:          add(tfield2);
19:          add(new Label(13, Label.RIGHT));
20:          add(tfield3);
21:
22:      }
23:
24:      public Insets getInsets( ) {
25:          return new Insets(10, 10, 0, 0);
26:      }
27:
28:      public boolean action(Event evt, Object arg) {
29:           if (evt.target instanceof TextField) {
30:               applet.update(this);
31:               return true;
32:           }
33:           else return false;
34:      }
35: }
```

Depois que esses dois arquivos de classe tiverem sido compilados, o applet ColorTest poderá ser carregado em uma página com o seguinte código HTML:

```
<applet code="ColorTest.class" width=475 height=100>
</applet>
```

Resumo

Quatro dias é um longo tempo para focalizar um elemento específico da linguagem Java, mas o Abstract Windowing Toolkit representa uma parte essencial dos conhecimentos de qualquer programador Java.

Agora você pode criar uma interface gráfica com o usuário para um applet ou mesmo um aplicativo, usando o AWT e as técnicas de Java 1.02. Durante a última semana deste livro, você aprenderá a executar algumas das mesmas tarefas usando as classes de janelas Swing.

Seja sua resposta um triste adeus ou um alegre "já vai tarde", a partir de amanhã você passará do AWT para novos assuntos.

Isso é bem merecido: bom trabalho!

Perguntas e respostas

P **Em nossa discussão sobre aplicativos independentes, tive a impressão de que não há absolutamente nenhuma diferença entre um applet e um aplicativo. Por que não?**

R Os applets e os aplicativos usam os mesmos procedimentos dentro do AWT para construir componentes, apresentá-los e tratar de eventos. Fora as restrições de segurança, as únicas diferenças são que os aplicativos são inicializados a partir de mai n ( ) e apresentam suas próprias janelas, e os applets são inicializados e iniciados a partir de init( ) e start( ), respectivamente. Dado o enorme número de semelhanças entre applets e aplicativos, 99% do que você aprende a respeito de applets pode ser usado com aplicativos. Na verdade, como os applets ignoram o método mai n ( ), caso ele exista em uma classe, não há motivo para que você não possa criar um programa único que seja executado igualmente bem como um applet e como um aplicativo.

P **Criei um aplicativo independente, mas nada acontece quando dou um clique na caixa Close. O que preciso fazer para que meu aplicativo realmente se feche?**

R Capture o evento de fechamento de janela com WINDOW_CLOSE, no modelo de eventos 1.02. Em resposta a esse evento, chame hi d**e**( ), caso ele possa voltar posteriormente, ou chame destroy( ) para desfazer-se dele totalmente. Se o evento de fechamento de janela resultar na saída de seu programa inteiro, chame também System.exi t ( ).

SEMANA 3

Aprimorando seus conhecimentos de Java

15 Papéis da classe: pacotes, interfaces e outros recursos

16 Circunstâncias excepcionais: tratamento de er e segurança

17 Manipulando dados através de fluxos Java

18 Comunicando-se via Internet

19 JavaBeans e outros recursos avançados

20 Criando uma interface com o usuário com o Swing

21 Tratando de eventos de usuário com o Swing

SEMANA 3

Papéis da classe: pacotes, interfaces e outros recursos

A terceira semana deste curso amplia o que você já sabe. Você poderia sair neste ponto e desenvolver programas funcionais, mas estariam faltando alguns recursos avançados que exprimem o real poder da linguagem.

Hoje, você ampliará seu conhecimento de classes e como elas interagem com outras classes em um programa Java. Os seguintes assuntos serão abordados:

- Controle do acesso aos métodos e variáveis fora de uma classe
- Finalização de classes, métodos e variáveis para que seus valores ou definições não possam ser subclasses ou não possam ser anulados
- Criação de classes e métodos abstratos para conseguir comportamento comum em superclasses
- Agrupamento de classes em pacotes
- Uso de interfaces para ligar lacunas em uma hierarquia de classe

Modificadores

As técnicas de programação que você aprenderá hoje envolvem diferentes estratégias e maneiras de se pensar sobre como uma classe é organizada. Mas uma coisa que todas essas técnicas têm em comum é que elas utilizam palavras-chave modificadoras especiais na linguagem Java.

Na Semana 1, você aprendeu a definir classes, métodos e variáveis em Java. Os modificadores são palavras-chave que você insere nessas definições para mudar seu significado.

A linguagem Java possui uma ampla variedade de modificadores, incluindo:

- Modificadores para controlar o acesso a uma classe, método ou variável: public, protected e private.
- O modificador stati c, para a criação de métodos e variáveis de classe.
- O modificador final, para finalizar as implementações de classes, métodos e variáveis.
- O modificador abstract, para a criação de classes e métodos abstratos.
- Os modificadores synchroni zed e vol ati 1 e, que são usados para threads.

Para usar um modificador, você inclui sua palavra-chave na definição da classe, método ou variável que está sendo modificada. O modificador precede o restante da instrução, como nos exemplos a seguir:

```
public class MyApplet extends java.applet.Applet { ... }

private boolean killJabberwock;

static final double weeks = 9.5;

protected static final int MEANINGOFLIFE = 42;

public static void main(String argumentsf ]) { ...}
```

Se você estiver usando mais de um modificador em uma instrução, pode colocá-los em qualquer ordem, desde que todos os modificadores precedam o elemento que estão modificando. Evite tratar do tipo de retorno de um método — como void —como se ele fosse um dos modificadores.

Os modificadores são opcionais — o que você já deve ter percebido, após utilizar alguns deles nas duas semanas anteriores. Contudo, você pode ter muitos bons motivos para usá-los, conforme verá.

Controle de acesso para métodos e variáveis

Os modificadores que você vai usar mais freqüentemente em seus programas são aqueles que controlam o acesso a métodos e variáveis: public, private e protected. Esses modificadores determinam quais variáveis e métodos de uma classe são visíveis em outras classes.

Usando controle de acesso, você determina como sua classe será usada por outras classes. Algumas variáveis e métodos em uma classe serão utilizados apenas dentro da própria classe e devem ser ocultados de outras classes que poderiam interagir com a classe. Esse processo é chamado de encapsulamento: um objeto controla o que o mundo exterior pode saber a respeito dele e como pode interagir com ele.

Encapsulamento é o processo de impedir que as variáveis de uma classe sejam lidas ou modificadas por outras classes. A única maneira de usar essas variáveis é chamando métodos da classe, se eles estiverem disponíveis.

A linguagem Java fornece quatro níveis de controle de acesso: public, private, protected e um nível padrão que é especificado pela utilização de nenhum modificador.

Acesso padrão

Para a maioria dos exemplos deste livro, você não especificou nenhum tipo de controle de acesso. As variáveis e métodos foram declarados com instrução como as seguintes:

```
String singer = "Phil Harris";
boolean digThatCrazyBeat( ) {
    return true;
}
```

Uma variável ou método declarado sem nenhum modificador de controle de acesso está disponível para qualquer outra classe no mesmo pacote. Anteriormente, você viu como as classes da biblioteca de classe Java são organizadas em pacotes. O pacote java.awt é um deles — um conjunto de classes relacionadas para comportamento relacionado ao Abstract Windowing Toolkit da linguagem Java.

Qualquer variável declarada sem um modificador pode ser lida ou alterada por qualquer outra classe no mesmo pacote. Qualquer método declarado da mesma maneira pode ser chamado por qualquer outra classe no mesmo pacote. Nenhuma outra classe pode acessar esses elementos de maneira alguma.

Esse nível de controle de acesso não exerce realmente muito controle. Quando você começar a pensar mais a respeito de como sua classe será usada por outras classes, utilizará um dos três modificadores com mais freqüência do que aceitará o controle padrão.

A discussão precedente levanta a dúvida sobre em que pacote suas classes estiveram até agora. Como você verá posteriormente ainda hoje, é possível tornar sua classe um membro de um pacote, usando a instrução package. Se você não usar essa estratégia, a classe será colocada em um pacote com todas as outras classes que não pertencem a nenhum outro pacote.

Acesso privativo

Para impedir completamente um método ou variável de ser usado por quaisquer outras classes, você utiliza o modificador private. O único lugar em que esses métodos ou variáveis podem ser vistos é dentro de sua própria classe.

Uma variável de instância privativa, por exemplo, pode ser usada pelos métodos de sua própria classe, mas não por objetos de outra classe. Da mesma maneira, os métodos privativos podem ser chamados por outros métodos de sua própria classe, mas por nenhum outro. Essa restrição também afeta a herança: nem as variáveis privativas nem os métodos privativos são herdados pelas subclasses.

As variáveis privativas são extremamente úteis em duas circunstâncias:

- Quando outras classes não têm motivo para usar essa variável.
- Quando outra classe poderia ser estragada pela mudança da variável de um modo inadequado.

Por exemplo, considere uma classe Java chamada BingoBrain que gera números de bingo para um site de jogos na Internet. Uma variável dessa classe, chamada winRatio, poderia controlar o número de ganhadores e perdedores que são gerados. Como você pode imaginar, essa variável tem um grande impacto sobre o lucro resultante do site. Se a variável fosse alterada por outras classes, o desempenho de BingoBrain mudaria bastante. Para prevenir-se contra esse cenário, você pode declarar a variável winRatio como private.

A classe a seguir usa controle de acesso privativo:

```
class Writer {
    private boolean writersBlock = true;
    private String mood;
    private int income = 0;

    private void getIdea(Inspiratior) in) {
      // ...
    }

    Manuscript createManuscript(int numDays, long numPages) {
      // ...
    }
}
```

Nesse exemplo de código, os dados internos da classe Wri ter (as variáveis writersBlock, mood, income e o método getIdea( )) são todos privativos. O único método acessível fora da classe Writer é createManuscript( ). create-Manuscript( ) é a única tarefa que os outros objetos podem solicitar para que o objeto Wri ter execute. Os objetos Edi tor e Publ i sher poderiam preferir um meio mais direto de extrair um objeto Manuscri pt da classe Wri ter, mas eles não possuem o acesso para fazer isso.

A utilização do modificador pri vate é a principal maneira pela qual um objeto se encapsula. Você não pode limitar as maneiras pelas quais uma classe é usada, sem usar private em muitos lugares para ocultar variáveis e métodos. Outra classe estará livre para mudar as variáveis dentro de uma classe e chamar seus métodos da forma desejada, se você não controlar o acesso.

Acesso público

Em alguns casos, talvez você queira que um método ou uma variável de uma classe esteja completamente disponível para qualquer outra classe que queira utilizá-los. Considere a variável de classe bl ack da classe Col or. Essa variável é usada quando uma classe quer utilizar a cor preta; portanto, bl ack não deve ter nenhum controle de acesso.

As variáveis de classe são freqüentemente declaradas como public. Um exemplo seria um conjunto de variáveis em uma classe Footbal 1 que representa o número de pontos utilizados no placar. A variável TOUCHDOWN poderia ser igual a 7, a variável FIELDGOAL poderia ser igual a 3 etc. Essas variáveis precisariam ser públicas para que outras classes pudessem utilizá-las em instruções como as seguintes:


```
if (position < 0) {
    System.out.pri ntln("Touchdown!");
    score = score + Football.TOUCHDOWN;
}
```

O modificador public torna um método ou variável completamente disponível para todas as classes. Você já o utilizou em todos os aplicativos que escreveu até agora, com uma instrução como a seguinte:

```
public static void main(String[ ] arguments) {
    // ...
}
```

O método mai n( ) de um aplicativo precisa ser público. Caso contrário, ele não poderia ser chamado pelo interpretador java para executar a classe.

Devido à herança de classe, todos os métodos e variáveis públicos de uma classe são herdados por suas subclasses.

Acesso protegido

O terceiro nível de controle de acesso é limitar um método e uma variável para uso pelos dois grupos a seguir:

- Subclasses de uma classe.
- Outras classes no mesmo pacote.

Você faz isso usando o modificador protected, como na instrução a seguir:

```
protected boolean weNeedMoreCalgon = true;
```

Você pode estar se perguntando como esses dois grupos diferem. Afinal, as subclasses não fazem parte do mesmo pacote que suas superclasses? Nem sempre. Um exemplo é a classe Appl et. Ela é uma subclasse de java.awt. Panei, mas está em seu próprio pacote, java.appl et. O acesso protegido difere do acesso padrão desse modo; as variáveis protegidas estão disponíveis para as subclasses, mesmo que não estejam no mesmo pacote.

Esse nível de controle de acesso é útil se você quer tornar mais fácil para uma subclasse implementar a si mesma. Sua classe poderia usar um método ou variável para ajudar a classe a realizar seu trabalho. Como uma subclasse herda grande parte dos mesmos comportamentos e atributos, ela poderia ter a mesma tarefa a realizar. O acesso protegido dá à subclasse uma chance de usar o método ou variável auxiliar, enquanto impede que uma classe não-relacionada tente usá-lo.

Considere o exemplo de uma classe chamada AudioPlayer que reproduz um arquivo de áudio digital. AudioPlayer possui um método chamado openSpeaker( ), que é um método interno que interage com o hardware para preparar o alto-falante para a reprodução. openSpeaker( ) não é importante para ninguém fora da classe AudioPlayer; portanto, inicialmente, talvez você queira torná-la private. Um fragmento de AudioPlayer poderia ser semelhante ao seguinte:

```
class AudioPlayer {

    private boolean openSpeaker(Speaker sp_ {
      // detalhes da implementação
    }
}
```

Esse código funciona bem se AudioPlayer não se transformar em uma subclasse. Mas, e se você criasse uma classe chamada StreamingAudioPlayer que é uma subclasse de AudioPlayer? Essa classe iria querer acessar o método openSpeaker( ) para que pudesse anulá-lo e fornecer inicialização de alto-falante específica do fluxo de áudio. Você ainda não quer o método disponível genericamente para objetos aleatórios (e, portanto, ele não deve ser public), mas quer que a subclasse tenha acesso a ele.

Comparando níveis de controle de acesso

As diferenças entre os diversos tipos de proteção podem se tornar muito confusas, particularmente no caso de métodos e variáveis protected. A Tabela 15.1, que resume exatamente o que é permitido e onde, ajuda a esclarecer as diferenças das formas menos restritivas (public) às mais restritivas (private) de proteção.

Tabela 15.1 Os diferentes níveis de controle de acesso.

Visibilidade	*public*	*protected*	*default*	*private*
A partir da mesma classe	sim	sim	sim	sim
A partir de qualquer classe no mesmo pacote	sim	sim	sim	não

Tabela 15.1 Continuação

Visibilidade	*public*	*protected*	*default*	*private*
A partir de qualquer classe fora do pacote	sim	não	não	não
A partir de uma subclasse no mesmo pacote	sim	sim	sim	não
A partir de uma subclasse fora do mesmo pacote	sim	sim	não	não

Controle de acesso e herança

Uma última questão a respeito ao controle de acesso para métodos envolve as subclasses. Quando cria uma subclasse e anula um método, você deve considerar o controle de acesso que está em vigor no método original.

Você deve se lembrar de que esses métodos Applet, como `init( )` e `pai nt( )`, devem ser `publ i c` em seus applets.

Como regra geral, você não pode anular um método na linguagem Java e tornar o novo método mais controlado do que o original. Você pode, entretanto, torná-lo mais público. As seguintes regras são impostas para métodos herdados:

- Métodos declarados como `publ i c` em uma superclasse também devem ser public em todas as subclasses (por esse motivo, a maioria dos métodos de applet é `publ ic`).
- Métodos declarados como protected em uma superclasse devem ser protected ou `publ i c` nas subclasses; eles não podem ser `pri vate`.
- Métodos declarados sem controle de acesso (nenhum modificador foi usado) podem ser declarados como mais privativos nas subclasses.

Os métodos declarados como private não são herdados; portanto, as regras não se aplicam.

Métodos de acesso

Em muitos casos, você pode ter uma variável de instância em uma classe que possui regras restritas para os valores que contém. Um exemplo seria uma variável zipCode. Um código postal (zip code) nos Estados Unidos deve ser um número de cinco dígitos de extensão: 10000 a 99999 são valores válidos, mas outros inteiros fora desse intervalo não podem ser códigos postais.

Para impedir que uma classe externa defina a variável zipCode incorretamente, você pode declará-la `private` com uma instrução como a seguinte:

```
private int zipCode;
```

No entanto, e se outras classes precisarem definir a variável zipCode para que a classe seja útil? Nessa circunstância, você pode dar a outras classes acesso a uma variável privativa, usando um método de acesso dentro da mesma classe como zipCode.

Os métodos de acesso recebem esse nome porque fornecem acesso a algo que, de outro modo, estaria fora dos limites. Usando um método para fornecer acesso a uma variável privativa, você pode controlar como essa variável é usada. No exemplo do código postal, a classe poderia impedir que se definisse zipCode com um valor incorreto.

Freqüentemente, estão disponíveis métodos de acesso para ler e escrever uma variável. Os métodos de leitura têm um nome que começa com get e os métodos de escrita possuem um nome que começa com set, como em setZi p-*Code(int)* egetZipCode(iní).

Essa convenção está se tornando mais padronizada a cada nova versão da linguagem Java. Você pode se lembrar de como o método size( ) da classe Dimension foi alterado para getSize( ) a partir da linguagem Java 2. Talvez você queira usar a mesma convenção de atribuição de nomes para seus próprios métodos de acesso, como um meio de tornar a classe mais inteligível.

Usar métodos para acessar variáveis de instância é uma técnica freqüentemente utilizada na programação orientada a objetos. Essa estratégia torna as classes mais reutilizáveis, pois evita que uma classe seja usada incorretamente.

Variáveis e métodos estáticos

Um modificador que você já usou em programas é stati c, que foi apresentado no Dia 6. O modificador stati c é usado para criar métodos e variáveis de classe, como no exemplo a seguir:

```
public class Circle {
    public static float pi = 3.14159265F;

    public float area(float r) {
      return pi * r * r;
    }
}
```

As variáveis e métodos de classe podem ser acessados usando-se o nome da classe seguido de um ponto final e do nome da variável ou método, como em Color.black ou Circle.pi. Você também pode usar o nome de um objeto da classe, mas no caso das variáveis e métodos de classe usar o nome da classe é melhor. Essa estratégia torna mais claro o tipo de variável ou método com que você está trabalhando; nunca se pode fazer referência às variáveis e aos métodos de instância pelo nome da classe.

As instruções a seguir usam variáveis e métodos de classe:

```
float circunference - Z * Circle.pi * getRadius( );
float randomNumer = Math.random( );
```

Dica

Pelas mesmas razões que continuam válidas para variáveis de instância, as variáveis de classe podem tirar proveito de serem privativas e limitar seu uso apenas aos métodos de acesso.

A listagem 15.1 mostra uma classe chamada CountInstances que usa variáveis de classe e de instância para controlar quantas instâncias dessa classe foram criadas.

DIGITE

Listagem 15.1. O texto completo de countinstances.java.

```
1: public class CountInstances {
2:      private static int numInstances = 0;
3:
4:      protected static int getNumInstances( ) {
5:        return numInstances;
6:       }
7:
8:      private static void addInstance( ) {
9:           numInstances++;
10:      }
11:
12:     CountInstances( ) {
13:          CountInstances.addInstance( );
14:      }
15:
16:     public static void main(String arguments[ ]) {
17:          System.out.println("Starting with " +
18:            CountInstances.getNumInstances( ) + " instances");
19:          for (int i = 0; i < 10; ++i)
20:            new CountInstances( );
21:          System.out.println("Created " +
22:               CountInstances.getNumInstances( ) + " instances");
23:      }
24: }
```

A saída desse programa é a seguinte:

SAÍDA

```
Started with 0 instances
Creates 10 instances
```

Esse exemplo possui vários recursos; portanto, analise-o linha a linha. Na linha 2, você declara uma variável de classe private para conter o número de

instâncias (chamada numlnstances). Trata-se de uma variável de classe (declarada como static), pois o número de instâncias é relevante para a classe como um todo e não para uma instância. E é privativa para que siga as mesmas regras dos métodos de acesso das variáveis de instância.

Observe a inicialização de numlnstances como 0 nessa mesma linha. Assim como uma variável de instância é inicializada quando sua instância é criada, uma variável de classe é inicializada quando sua classe é criada. Essa inicialização de classe acontece basicamente antes que qualquer coisa possa acontecer com essa classe, ou suas instâncias, para que a classe do exemplo funcione como o planejado.

Nas linhas 4 a 6, você cria um método get para que a variável de instância privativa receba seu valor (getNumInstances( )). Esse método também é declarado como um método de classe, pois ele se aplica diretamente à variável de classe. O método getNumInstances( ) é declarado como protected, e não como public, pois apenas essa classe e, talvez, as subclasses, estarão interessadas nesse valor; portanto, existe uma restrição para que outras classes aleatórias a vejam.

Observe que você não possui um método de acesso para definir o valor. A razão é que o valor da variável deve ser incrementado apenas quando uma nova instância for criada; ele não deve ser definido com nenhum valor aleatório. Portanto, em vez de criar um método de acesso, você cria um método privativo especial chamado addlnstance( ), nas linhas 8 a 10, que incrementa o valor de numlnstances em 1.

As linhas 12 a 14 criam o método construtor dessa classe. Lembre-se de que os construtores são chamados quando um novo objeto é criado, o que torna esse o local mais lógico para se chamar addInstance( ) e incrementar a variável.

Finalmente, o método main ( ) indica que você pode executar esse aplicativo Java e testar todos os outros métodos. No método main( ), você cria 10 instâncias da classe CountInstances, relatando, após a conclusão, o valor da variável de classe numlnstances (que, de forma previsível, imprime 10).

Classes, métodos e variáveis finais

O modificador final é usado com classes, métodos e variáveis para indicar que eles não serão alterados. Ele tem significados diferentes para cada item que pode ser finalizado, como segue:

- Uma classe final não pode se tornar uma subclasse.
- Um método final não pode ser anulado por nenhuma subclasse.
- Uma variável f i nal não pode mudar de valor.

Variáveis

Você teve uma chance de trabalhar com variáveis finais no Dia 6. Freqüentemente, elas são chamadas de variáveis constantes (ou apenas constantes), pois não mudam de valor em nenhum momento.

No caso das variáveis, o modificador final é usado muitas vezes com stati c para transformar a constante em uma variável de classe. Se o valor nunca muda, você não tem muitos motivos para dar a cada objeto da mesma classe sua própria cópia do valor. Todos eles podem utilizar a variável de classe com a mesma funcionalidade.

As instruções a seguir são exemplos de declaração de constantes:

```
public static final int touchdown = 7;
static final title = "Captain";
```


A partir da Java 2, qualquer tipo de variável pode ser uma variável final: variáveis de classe, instância ou locais. Uma variável local não podia ser final no Java 1.02, mas isso foi mudado como parte da adição de classes internas na linguagem.

Métodos

Os métodos finais são aqueles que nunca podem ser anulados por uma subclasse. Você os declara usando o modificador final na declaração de classe, como no exemplo a seguir:

```
public final void getMaxwellSmart( ) {
    // ...
}
```

A única razão para se declarar um método final é fazer a classe ser executada mais eficientemente. Normalmente, quando um ambiente de runtime Java, como o interpretador j ava, executa um método, primeiro ele verifica a classe atual para localizar o método, em seguida ele verifica sua superclasse e sobe na hierarquia de classe até que o método seja encontrado. Esse processo sacrifica certa velocidade em nome da flexibilidade e facilidade de desenvolvimento.

Se um método é final, o compilador Java pode colocar seu código de byte executável do método diretamente em qualquer programa que o chame. Afinal, o método nunca mudará, devido à subclasse que o anula.

Quando você estiver desenvolvendo uma classe pela primeira vez, não terá muitas razões para usar final. Entretanto, se precisa fazer essa classe executar mais rapidamente, você pode mudar alguns métodos para final, para acelerar o processo. Fazer isso elimina a possibilidade de o método se tornar uma subclasse posteriormente; portanto, considere essa alteração com cuidado, antes de continuar.

A biblioteca de classe Java declara muitos dos métodos f i nal Comumente usados, de modo que eles podem ser executados mais rapidamente quando utilizados em programas que os chamam.

Os métodos private são finais sem serem declarados desse modo, pois eles não podem se tornar subclasses em nenhuma circunstância.

Classes

Você finaliza classes usando o modificador final na declaração da classe, como no seguinte:

```
public final class AnotherFineMess {

}
```

Uma classe final não pode se tornar uma subclasse por outra classe. Assim como acontece com os métodos finais, esse processo apresenta algumas vantagens na velocidade da linguagem Java, às custas da flexibilidade.

Se você estiver se perguntando o que está perdendo quando usa classes final, ainda não deve ter tentado transformar em subclasse alguma coisa da biblioteca de classe Java. Muitas das classes conhecidas são finais, como java.lang.String, java.lang.Math e java.net. InetAddress. Se quiser criar uma classe que se comporte como strings, mas com algumas alterações novas, você não pode transformar String em uma subclasse e definir apenas o comportamento que é diferente. Você deve começar do início.

Todos os métodos de uma classe final são automaticamente finais; portanto, você não precisa usar um modificador em suas declarações.

Você não terá muitos motivos para tornar suas próprias classes finais, pois as classes que podem herdar seu comportamento e atributos para subclasses são muito mais úteis.

Classes e métodos abstratos

Em uma hierarquia de classe, quanto mais alta a classe, mais abstrata é sua definição. Uma classe no topo da hierarquia de outras classes pode definir apenas o comportamento e os atributos que são comuns a todas as classes. Comportamento e atributos mais específicos recairão em algum lugar mais abaixo na hierarquia.

Quando você está considerando comportamento e atributos comuns, durante o processo de definição de uma hierarquia de classes, às vezes pode encontrar uma classe que nem mesmo precisa ser instanciada diretamente. Em vez disso, essas classes servem como um local para conter comportamento e atributos comuns, compartilhados por suas subclasses.

Essas classes são chamadas de classes abstratas e são criadas usando-se o modificador abstract. A seguir está um exemplo:

```
public abstract class BroadwayBoogieWoogie {
    // •••
}
```

Um exemplo de classe abstrata é java.awt.Component, a superclasse de todos os componentes de Abstract Windowing Toolkit. Todos os componentes herdam dessa classe, de modo que ela contém métodos e variáveis úteis para cada um deles. Entretanto, não existe um componente genérico que possa ser incluído em uma interface; portanto, você nunca precisaria criar um objeto Component em um programa.

As classes abstratas podem conter qualquer coisa que uma classe normal também pode, incluindo métodos construtores, pois suas subclasses podem precisar herdar os métodos. As classes abstratas também podem conter métodos abstratos que são assinaturas de método sem nenhuma implementação. Esses métodos são implementados em subclasses da classe abstrata. Os métodos abstratos são declarados com o modificador abstract. Você não pode declarar um método abstrato em uma classe não-abstrata. Se uma classe abstrata possui apenas métodos abstratos, é melhor usar uma interface, como você verá posteriormente hoje.

Pacotes

Usar pacotes, como foi mencionado anteriormente, é um modo de organizar grupos de classes. Um pacote contém qualquer número de classes que se relacionam pelo objetivo, pelo escopo ou pela herança.

Se seus programas são pequenos e usam um número limitado de classes, você pode achar que não precisa explorar os pacotes. Mas, quanto mais programas Java você cria, mais classes descobrirá que possui. E, embora essas classes possam ser bem projetadas individualmente, reutilizáveis, encapsuladas e com interfaces específicas para outras classes, você pode encontrar a necessidade de uma entidade organizacional maior que permita agrupar seus pacotes.

Os pacotes são úteis por vários motivos:

- Eles permitem que você organize suas classes em unidades. Assim como você tem pastas ou diretórios em seu disco rígido para organizar seus arquivos e aplicativos, os pacotes permitem organizar suas classes em grupos para que você use apenas o que precisa para cada programa.
- Eles reduzem problemas com conflitos de nomes. A medida que o número de classes Java cresce, também aumenta a probabilidade de que você utilize o mesmo nome de classe que outra pessoa, abrindo a possibilidade de conflitos de atribuição de nomes e erros, se você tentar integrar grupos de classes em um único programa. Os pacotes permitem "ocultar" classes para que esses conflitos possam ser evitados.
- Eles permitem que você proteja classes, variáveis e métodos de modo mais amplo do que acontece quando se faz isso classe por classe, como aprendeu hoje. Você aprenderá mais a respeito de proteções com pacotes posteriormente.

- Eles podem ser usados para identificar suas classes. Por exemplo, se você implementasse um conjunto de classes para realizar alguma tarefa, poderia atribuir um nome a um pacote dessas classes com um identificador exclusivo identificando você ou sua organização.

Embora um pacote normalmente seja um conjunto de classes, eles também podem conter outros pacotes, formando um outro nível de organização, um tanto análoga à hierarquia de herança. Normalmente, cada "nível" representa um agrupamento menor e mais específico de classes. A própria biblioteca de classe Java é organizada nesses termos. O nível superior é chamado java; o nível seguinte inclui nomes como io, net, útil e awt. O último deles tem até um nível inferior, que inclui o pacote image.

Por convenção, o primeiro nível da hierarquia especifica o nome globalmente exclusivo para identificar o autor ou proprietário desses pacotes. Por exemplo, as classes da Sun Microsystems, que não fazem parte do ambiente Java padrão, começam todas com o prefixo sun. As classes que a Netscape inclui com sua implementação estão contidas no pacote netscape. O pacote padrão, java, é uma exceção a essa regra por ser fundamental e porque algum dia poderá ser implementado por diversas empresas.

Usando pacotes

Você tem usado pacotes o tempo todo neste livro. Sempre que usa o comando import e que faz referência a uma classe por seu nome de pacote completo (java.awt.Color, por exemplo), você usa pacotes.

Para usar uma classe contida em um pacote, você pode utilizar três mecanismos distintos:

- Se a classe que você quer usar está no pacote java. 1 ang (por exemplo, System ou Date), então pode simplesmente utilizar o nome da classe para fazer referência a ela. As classes java. 1 ang estão automaticamente disponíveis para você, em todos os seus programas.
- Se a classe que você deseja usar está em algum outro pacote, então você pode fazer referência a essa classe por seu nome completo, incluindo quaisquer nomes de pacote (por exemplo, java.awt.Font).
- No caso das classes de outros pacotes utilizadas freqüentemente, você pode importar classes específicas ou um pacote inteiro de classes. Depois que uma classe ou pacote foi importado, você pode fazer referência a essa classe por seu nome.

Se você não declarar que sua classe pertence a um pacote, ela será colocada em um pacote padrão sem nome. Você pode fazer referência a essas classes simplesmente pelo nome, em qualquer lugar de seu código.

Nomes de pacote e de classe completos

Para fazer referência a uma classe em algum outro pacote, você pode usar seu nome completo: o nome da classe precedido de qualquer nome de pacote. Você não precisa importar a classe ou o pacote para usá-lo assim:

```
java.awt.Font f = new java.awt.Font( )
```

No caso das classes que você utiliza apenas uma ou duas vezes em seu programa, usar o nome completo faz mais sentido. Entretanto, se você usa essa classe várias vezes ou se o nome do pacote é muito longo, com diversos subpacotes, então, você deve importar essa classe, para evitar muita digitação.

O comando import

Para importar classes de um pacote, use o comando import, como fez em todos os exemplos deste livro. Você pode importar uma classe específica, como segue:

```
import java.util.Vector;
```

Ou você pode importar um pacote inteiro de classes, usando um asterisco (*) para substituir os nomes de classe específicos, como segue:

```
import java.awt.*
```

Nota Na verdade, para sermos tecnicamente corretos, esse comando não importa todas as classes de um pacote; ele importa apenas as classes que foram declaradas como `public`, mesmo assim, importa apenas as classes a que o próprio código faz referência. Você aprenderá mais a respeito desse assunto na seção "Pacotes e controle de acesso a classe".

Observe que o asterisco (*) nesse exemplo não é como o que você poderia utilizar em um prompt de comando para especificar o conteúdo de uma pasta ou para indicar vários arquivos. Por exemplo, se você pedir para listar o conteúdo do diretório `class/java/awt/*`, essa lista incluirá todos os arquivos e subdiretórios de .class, como image e peer. Escrever import java.awt.* importa todas as classes públicas desse pacote, mas não importa subpacotes, como image e peer. Para importar todas as classes de uma hierarquia de pacote complexa, você deve importar explicitamente cada nível da hierarquia manualmente. Além disso, você não pode indicar nomes de classe parciais (por exemplo, L* para importar todas as classes que começam com L). As únicas opções ao se usar uma instrução import são carregar todas as classes de um pacote ou carregar apenas uma classe.

As instruções import em sua definição de classe ficam no início do arquivo, antes de todas as definições de classe (mas depois da definição de pacote, como você verá na próxima seção).

Você deve importar classes individualmente ou apenas importá-las como um grupo? A resposta depende do quanto você quer ser específico. Importar um grupo de classes não diminui a velocidade de seu programa nem o torna maior; apenas as classes que você realmente utiliza em seu código são carregadas, à medida que forem necessárias. Mas importar um pacote torna um pouco mais confuso para os leitores de seu código descobrir de onde suas classes estão vindo. Usar instruções import individuais ou importar pacotes é principalmente uma questão de seu próprio estilo de codificação.

Se você está vindo para a linguagem Java de C ou C++, talvez possa esperar que a instrução import funcione como include, que resulta em um programa muito grande, incluindo código-fonte de outro arquivo. Não é o caso aqui. A instrução import indica apenas onde a classe pode ser encontrada; ela não faz nada para aumentar o tamanho de uma classe.

Conflitos de nome

Após ter importado uma classe ou um pacote de classes, normalmente você pode fazer referência a um nome de classe simplesmente por seu nome, sem o identificador de pacote. Em um caso, talvez você tenha de ser mais explícito: quando tem várias classes com o mesmo nome, de diferentes pacotes.

A seguir há um exemplo. Digamos que você importe as classes de dois pacotes de dois programadores diferentes (Jonathan e Bourne):

```
import jonathanclasses.*;
import bourneclasses.*;
```

Dentro do pacote de Jonathan existe uma classe chamada Defenestrate. Infelizmente, dentro do pacote de Bourne, você também encontra uma classe chamada Defenestrate que possui um significado e uma implementação completamente diferentes. Você poderia se perguntar qual versão de Defenestrate será usada se fizer referência à classe Defenestrate em seu programa, como segue:

```
Defenestrate outWindow = new Defenestrate("Phil");
```

A resposta é: nenhuma das duas. O compilador Java reclama de um conflito de atribuição de nomes e se recusa a compilar seu programa. Nesse caso, a despeito do fato de você ter importado as duas classes, ainda terá de fazer referência à classe Defenestrate apropriada através do nome completo do pacote, como segue:

```
jonathanclasses.Defenestrate outWindow = new
    jonathanclasses.Defenestrate("Phil");
```

Uma observação sobre CLASSPATH e onde as classes são localizadas

Para que a linguagem Java possa utilizar uma classe, ela precisa encontrar essa classe no sistema de arquivos. Caso contrário, você obterá um erro dizendo que a classe não existe. A linguagem Java usa dois elementos para localizar classes: o próprio nome do pacote e os diretórios listados em sua variável CLASSPATH (se você estiver em um sistema Windows ou Solaris).

Primeiramente, os nomes de pacote. Tais nomes são mapeados para nomes de diretório no sistema de arquivos; portanto, a classe java.applet. Applet é encontrada no diretório applet que, por sua vez, está dentro do diretório java (em outras palavras, java\applet\Applet.class).

A linguagem Java, por sua vez, procura esses diretórios dentro dos diretórios listados em sua variável CLASSPATH, caso ela tenha sido fornecida em sua configuração. Lembre-se do Dia 1, quando você instalou o JDK, que poderia ser necessário ter uma variável CLASSPATH para apontar para os diversos lugares em que suas classes Java se encontram. Se nenhuma variável CLASSPATH for fornecida, o JDK procurará o padrão — o diretório java\lib em sua versão de JDK e na pasta atual. Quando a linguagem Java procura uma classe a que você fez referência em seu código-fonte, ela procura o pacote e o nome da classe em cada um desses diretórios e retorna um erro, se não puder localizar o arquivo de classe. A maioria dos erros do tipo class not found resulta de variáveis CLASSPATH malconfiguradas.

Criando seus próprios pacotes

Criar um pacote para algumas de suas classes em Java não é muito mais complicado do que criar uma classe. Você deve seguir as três etapas básicas delineadas a seguir.

Escolher um nome de pacote

A primeira etapa é decidir-se quanto ao nome. O nome escolhido para seu pacote depende de como você vai usar essas classes. Talvez você atribua o nome de seu pacote de acordo com o seu ou talvez de acordo com a parte do sistema Java em que está trabalhando (como graphi cs ou hardware_i nterf aces). Se você pretende distribuir seu pacote na Rede em geral ou como parte de um produto comercial, então deve usar um nome que identifique exclusivamente o autor.

Uma convenção para a atribuição de nomes de pacotes, recomendada pela Sun, é usar seu nome de domínio na Internet com os elementos invertidos. Por exemplo, se a Sun seguisse essa recomendação, a referência a seus pacotes seria de acordo com o nome com.sun. java, em vez de apenas java. Se seu nome de domínio na Internet fosse prefect.com, o nome de seu pacote poderia ser com.prefect. Talvez você queira estender o nome com algo que descreva as classes do pacote, como em com.prefect.canasta.

A idéia é garantir que. o nome de seu pacote seja exclusivo. Embora os pacotes possam ocultar nomes de classe conflitantes, a proteção pára aí. Você não pode garantir que seu pacote não vai estar em conflito com o pacote de outra pessoa, se ambos usarem o mesmo nome de pacote.

Por convenção, os nomes de pacote tendem a começar com uma letra minúscula para distingui-los dos nomes de classe. Assim, por exemplo, no nome completo da classe interna String, java.lang.String, você pode separar mais facilmente, de forma visual, o nome do pacote do nome da classe. Essa convenção ajuda a reduzir os conflitos de nome.

Criar a estrutura de pasta

A segunda etapa na criação de pacotes é gerar uma estrutura de pasta em sua unidade de disco rígido, que combine com o nome de pacote. Se seu pacote tem apenas um nome (mypackage), você deve criar uma pasta apenas para esse nome. Se o nome do pacote possui várias partes, você precisa criar pastas dentro de pastas. No caso do nome de pacote com. prefect. canasta, por exemplo, você precisa criar uma pasta com, uma pasta pref ect dentro de com e uma pasta canasta dentro de prefect. Suas classes e arquivos-fonte poderão, então, ficar dentro do diretório prefect.

Adicionar uma classe a um pacote

A última etapa para colocar sua classe dentro de pacotes é adicionar uma instrução ao arquivo de classe, acima de todas as instruções import que estejam sendo usadas. A instrução package é utilizada junto com o nome do pacote, como no seguinte:

```
package com.prefect.canasta;
```

O único comando package, se houver, deve ser a primeira linha de código em seu arquivo-fonte, após todos os comentários ou linhas em branco e antes de todos os comandos import.

Depois de começar a usar pacotes, você deve certificar-se de que todas as suas classes pertençam a algum pacote para reduzir a chance de confusão em relação a onde suas classes pertencem.

Pacotes e controle de acesso à classe

Anteriormente, você aprendeu a respeito de modificadores de controle de acesso para métodos e variáveis. Você também pode controlar o acesso às classes, como pode ter observado quando o modificador public foi usado em algumas declarações de classe em objetos passados.

As classes possuem o controle de acesso padrão, se nenhum modificador for especificado, o que significa que a classe está disponível para todas as outras classes no mesmo pacote, mas não é visível nem disponível fora desse pacote

— nem mesmo para subpacotes. Ela não pode ser importada ou referida pelo nome; as classes com proteção de pacote ficam ocultas dentro do pacote em que estão contidas.

A proteção de pacote entra em ação quando você define uma classe, como foi feito em todo este livro, como segue:

```
class TheHiddenClass extends AnotherHiddenClass {
    // ...
}
```


Para permitir que uma classe seja visível e possa ser importada fora de seu pacote, você pode dar-lhe proteção pública, adicionando o modificador public em sua definição:

```
public class TheVisibleClass {
    // ...
}
```

As classes declaradas como public podem ser importadas por outras classes fora do pacote.

Observe que, quando usa uma instrução import com um asterisco, você importa apenas as classes públicas de dentro desse pacote. As classes ocultas permanecem nesse estado e só podem ser usadas por outras classes desse pacote.

Por que você iria querer ocultar uma classe dentro de um pacote? Pela mesma razão que desejaria ocultar variáveis e métodos dentro de uma classe: para poder ter comportamento e classes utilitárias que são úteis apenas para sua implementação ou para poder limitar a interface de seu programa a fim de minimizar o efeito de alterações maiores. Ao projetar suas classes, você deve levar todo o pacote em consideração e decidir quais classes deseja declarar como publ i c e quais deseja que fiquem ocultas.

A listagem 15.2 mostra duas classes que ilustram esse ponto. A primeira é uma classe pública que implementa uma lista encadeada; a segunda é um nó privativo dessa lista.

DIGITE

Listagem 15.2. O texto completo de 1 inkedlist.java.

```
1: package collections;
2:
3: public class LinkedList {
4:     private Node root;
5:
6:     public void add(Object o) {
7:         root = new Node(o, root);
8:     }
9:   // ...
10: }
11:
```

Listagem 15.2. Continuação

```
12: class Node {  // não pública
13:      private Object contents;
14:      private Node  next;
15:
16:      Node(Object o, Node n) {
17:           contents = o;
18:           next   = n;
19:      }
20:      // ...
21: }
```

Nota

Você pode incluir, por arquivo, quantas definições de classe quiser, mas apenas uma delas pode ser declarada como public e esse nome de arquivo deve ser igual ao da classe pública. Quando a linguagem Java compilar o arquivo, ela criará arquivos **.class** separados para cada definição de classe nele existente.

A classe pública LinkedList fornece um conjunto de métodos públicos úteis (como add ( )) para todas as outras classes que possam querer utilizá-los. Essas outras classes não precisam conhecer quaisquer classes de suporte de que LinkedList precisa para fazer seu trabalho. Portanto, Node, que é uma dessas classes de suporte, é declarada sem um modificador publ i c e não aparecerá como parte da interface pública para o pacote col1ections.

Apenas porque Node não é pública não significa que LinkedList não terá acesso a ela, depois que tiver sido importada para alguma outra classe. Considere a proteção não como uma ocultação total das classes, mas como uma verificação das permissões de determinada classe para usar outras classes, variáveis e métodos. Quando você importar e usar LinkedList, a classe Node também será carregada no sistema, mas apenas as instâncias de Li nkedLi st terão permissão de usá-la.

A criação de um bom pacote consiste na definição de um pequeno conjunto de classes e métodos públicos simples para outras classes usarem, seguida da sua implementação usando qualquer número de classes de suporte ocultas. Você verá outro uso de classes ocultas posteriormente, ainda hoje.

Interfaces

As interfaces, assim como as classes e métodos abstratos, fornecem modelos de comportamento esperados na implementação de outras classes. Entretanto, as interfaces fornecem muito mais funcionalidade para a linguagem Java e para o design de classes e objetos do que as classes e os métodos abstratos.

O problema da herança simples

No entanto, após alguma reflexão mais profunda ou uma experiência de design mais complexa, você pode descobrir que essa simplicidade pura da hierarquia de classe é restritiva, particularmente quando se tem algum comportamento que precisa ser utilizado por classes em diferentes ramos da mesma árvore.

Veja um exemplo que tornará os problemas mais claros. Digamos que você tenha uma hierarquia biológica com Animal no topo e as classes Mammal (mamífero) e Bird (pássaro) sob ela. Os itens que definem um mamífero incluem dar à luz a filhotes e ter pele. Os comportamentos ou características dos pássaros incluem ter um bico e pôr ovos. Ate aqui, tudo bem, certo? Então, como você vai criar uma classe para o ornitorrinco, que tem pele, bico e põe ovos? Você precisaria combinar o comportamento de duas classes para formar a classe Platypus (ornitorrinco). E, como as classes podem ter apenas uma superclasse imediata em Java, esse tipo de problema simplesmente não pode ser resolvido de forma elegante.

Outras linguagens de OOP incluem o conceito de herança múltipla, o que resolve o problema. Com a herança múltipla, uma classe pode herdar de mais de uma superclasse e obter comportamento e atributos de todas as suas superclasses simultaneamente. Um problema da herança múltipla é que ela torna uma linguagem de programação bem mais complexa para se aprender, usar e implementar. As questões sobre chamada de método e como a hierarquia de classe é organizada se tornam bem mais complicadas na herança múltipla e mais predispostas à confusão e ambigüidade. E, como um dos objetivos da linguagem Java era ser simples, a herança múltipla foi rejeitada em favor da herança simples (única), bem menos complexa.

Então, como você resolve o problema da necessidade de comportamento comum que não se encaixa na hierarquia de classe estrita? A linguagem Java possui outra hierarquia completamente separada da hierarquia de classe principal, uma hierarquia de classes de comportamento misto. Assim, quando você cria uma nova classe, essa classe possui apenas uma superclasse principal, mas pode selecionar e escolher diferentes comportamentos comuns da outra hierarquia. Essa outra hierarquia é a hierarquia de interface. Uma interface Java é um conjunto de comportamentos abstratos que podem ser misturados em qualquer classe para acrescentar a essa classe um comportamento que não é fornecido por suas superclasses. Especificamente, uma interface Java contém apenas definições de método abstratas e constantes — nada de variáveis de instância e nada de implementações de método.

As interfaces são implementadas e utilizadas em toda a biblioteca de classe Java, quando se espera que um comportamento seja implementado por diversas classes distintas. Por exemplo, a hierarquia de classe Java define e utiliza as interfaces java.lang.Runnable, java.útil.Enumeration, java.útil .Observable, java.awt.image.ImageConsumere java.awt.image.ImageProducer. Algumas dessas interfaces você já viu antes; outras você verá posteriormente neste livro.

Interfaces e classes

A despeito de suas diferentes definições, as classes e interfaces têm muito em comum. As interfaces, assim como as classes, são declaradas em arquivos-fonte, uma interface para um arquivo. Assim como as classes, elas também são compiladas em arquivos .class usando-se o compilador Java. E, na maioria dos casos, em qualquer lugar que você possa usar uma classe (como um tipo de dados para uma variável, como resultado de uma coerção etc), também pode usar uma interface.

Em quase todos os lugares em que este livro tem um nome de classe, em qualquer um de seus exemplos ou discussões, você pode substituir por um nome de interface. Os programadores Java freqüentemente dizem "classe" quando, na verdade, querem dizer "classe ou interface". As interfaces complementam e ampliam o poder das classes e as duas podem ser tratadas quase da mesma forma. Uma das poucas diferenças entre elas é que uma interface não pode ser instanciada: new só pode criar uma instância de uma classe.

Implementando e usando interfaces

Você pode fazer duas coisas com as interfaces: usá-las em suas próprias classes e definir as suas próprias interfaces. Por enquanto, comece com a primeira.

Para usar uma interface, você inclui a palavra-chave implements como parte de sua definição de classe. Você fez isso no Dia 10, quando aprendeu sobre threads e incluiu a interface Runnable na definição de seu applet:

```
public class Neko extends java.applet.Applet
    implements Runnable {
    //...
}
```

Nesse exemplo, java.applet.Applet é a superclasse, mas a interface Runnable estende o comportamento que ela implementa.

Como as interfaces fornecem apenas definições de método abstratas, você precisa, então, implementar esses métodos em suas próprias classes, usando as mesmas assinaturas de método da interface. Observe que, quando você inclui uma interface, precisa implementar todos os métodos dessa interface; você não pode selecionar e escolher os métodos de que necessita. Implementando uma interface, você está dizendo aos usuários de sua classe que oferece suporte a tudo dessa interface. (Note que essa é outra diferença entre interfaces e classes abstratas; as subclasses destas últimas podem selecionar quais métodos vão implementar ou anular e podem ignorar os outros.)

Depois que sua classe implementa uma interface, as subclasses de sua classe herdam aqueles métodos novos (e podem anulá-los ou sobrecarregá-los), exatamente como se sua superclasse as tivesse definido realmente. Se sua classe herda de uma superclasse que implementa determinada interface, você não precisa incluir a palavra-chave implements em sua própria definição de classe.

Agora, examine um exemplo simples — a criação da nova classe Orange. Suponha que você já tenha uma boa implementação da classe Fruit e uma interface, Fruitiike, que representa o que se espera que Fruit consiga fazer. Você quer que uma laranja (orange) seja uma fruta (fruit), mas também quer que ela seja um objeto esférico que pode ser agitado, rodado etc. Aqui está como expressar tudo isso (não se preocupe com as definições dessas interfaces por enquanto; você aprenderá mais sobre elas posteriormente, ainda hoje):

```
interface Fruitiike {
    void decay( );
    void squish( );
    // ...
}

class Fruit implements Fruitiike {
    private Color myColor;
    private int daysTilIRot;
    // ...
}

interface Spherelike {
    void toss( );
    void rotate( );
    // ...
}

class Orange extends Fruit implements Spherelike {
    // toss( )ing may squish( ) me (unique to me)
}
```

Observe que a classe Orange não precisa dizer implements Fruitlike, pois, ampliando Frui t, ela já faz isso! Um dos detalhes interessantes a respeito dessa estrutura é que você pode mudar de idéia sobre o que a classe Orange amplia (se, por exemplo, uma classe Sphere fantástica for implementada de repente) e ela ainda entenderá as mesmas duas interfaces:

```
class Sphere implements Spherelike {  // estende Object
    private float radius;
    // ...
}

class Orange extends Sphere implements Fruitiike {
    // ... os usuários de Orange nunca precisam saber da mudança!
}
```

Implementando várias interfaces

Ao contrario do que acontece com a hierarquia de classe de herança simples, você pode incluir quantas interfaces precisar em suas classes, e sua classe implementará o comportamento combinado de todas as interfaces incluídas. Para incluir várias interfaces em uma classe, basta separar seus nomes com vírgulas, como segue:

```
public class Neko extends java.applet.Applet
    implements Runnable, Eatable, Sortable, Observable {
    // ...
}
```

Observe que podem surgir complicações na implementação de várias interfaces. O que acontece se duas interfaces diferentes definem o mesmo método? Você pode resolver esse problema de três maneiras:

- Se os métodos de cada uma das interfaces tiverem assinaturas idênticas, você implementa um método em sua classe e essa definição satisfaz às duas interfaces.
- Se os métodos possuem listas de parâmetros diferentes, então, trata-se de um simples caso de sobrecarga de método; você implementa as duas assinaturas de método e cada definição satisfaz à sua respectiva definição de interface.
- Se os métodos possuem as mesmas listas de parâmetros, mas diferem no tipo de retorno, você não pode criar um método que satisfaça a ambos (lembre-se de que a sobrecarga de método é disparada pelas listas de parâmetros e não pelo tipo de retorno). Nesse caso, tentar compilar uma classe que implemente as duas interfaces produz um erro de compilador. Se esse problema for encontrado, isso sugere que suas interfaces têm algumas deficiências de design que talvez precisem ser reexaminadas.

Outros usos das interfaces

Lembre-se de que, em praticamente qualquer lugar onde você possa usar uma classe, pode usar uma interface em seu lugar. Assim, por exemplo, você pode declarar uma variável para ser um tipo de interface:

```
Runnable aRunnableObject = new MyAnimationClass( )
```

Quando uma variável é declarada para ser um tipo de interface, isso significa simplesmente que qualquer objeto a que a variável faça referência deve ter implementado essa interface; ou seja, espera-se que ele entenda todos os métodos que a interface especifica. Isso presume que uma promessa feita entre o projetista da interface e seus eventuais implementadores sejam mantidas. Nesse caso, como aRunnableObject contém um objeto do tipo Runnable, a suposição é de que você possa chamar aRunnableObject.run( ).

O ponto importante a se perceber aqui é que, embora se espere que aRunnabl eObj ect possa ter o método run ( ), você poderia escrever esse código bem antes que as classes que qualificam sejam realmente implementadas (ou mesmo criadas!). Na programação orientada a objetos tradicional, você é obrigado a criar uma classe com implementações "stub" (métodos vazios ou métodos que imprimem mensagens bobas) para obter o mesmo efeito. Você

também pode fazer a coerção de objetos para uma interface, exatamente como faz para outras classes. Assim, por exemplo, volte àquela definição da classe Orange, que implementou a interface Fruitlike (em sua superclasse, Frui t) e a interface Spherel i ke. Aqui, você pode fazer a coerção de instâncias de Orange para classes e interfaces:

```
Orange anOrange = new 0range( );
Fruit aFruit = (Fruit)anOrange;
Fruitlike aFruitlike = (Fruitlike)anOrange;
Spherelike aSpherelike = (Spherelike)anOrange;

aFruit.decay( ); // frutas deterioram
aFruitlike.squish( ); // e são espremidas
aFruitlike.toss( ); // as coisas que são do tipo fruta não são atiradas

aSpherelike.toss( ); // mas as coisas que são do tipo esférico, sim
anOrange.decay( ); //as laranjas podem fazer tudo isso
anOrange.squish( );
anOrange.toss( );
anOrange.rotate( );
```


As declarações e coerções são usadas nesse exemplo para restringir o comportamento de uma laranja para que atue de modo mais parecido com uma simples fruta ou esfera.

Finalmente, observe que, embora as interfaces sejam normalmente usadas para misturar comportamento em outras classes (assinaturas de método), elas também podem ser utilizadas para misturar constantes geralmente úteis. Assim, por exemplo, se uma interface define um conjunto de constantes e depois várias classes o utilizam, os valores dessas constantes poderiam ser alterados globalmente sem a necessidade de modificar várias classes. Esse é um outro exemplo de um caso em que o uso de interfaces para separar o design da implementação pode tornar seu código mais geral e fácil de manter.

Criando e ampliando interfaces

Depois que você utiliza um pouco as interfaces, o próximo passo é definir suas próprias interfaces. As interfaces são muito parecidas com as classes; são declaradas da mesma maneira e podem ser organizadas em uma hierarquia. Entretanto, você deve seguir certas regras para declarar interfaces.

Novas interfaces

Para criar uma nova interface, você a declara como segue:

```
public interface Growable {
    // ...
}
```

Essa declaração é, efetivamente, igual a uma definição de classe, com a palavra interface substituindo a palavra class. Dentro da definição da interface, você possui métodos e constantes. As definições de método dentro da

interface são métodos public e abstract; você pode declará-los explicitamente como tal, ou eles são transformados em métodos public e abstract, se esses modificadores não forem incluídos. Você não pode declarar um método dentro de uma interface para ser pri vate ou protected. Assim, por exemplo, aqui está uma interface Growabl e com um método explicitamente declarado como publ i c e abstract (growlt( )) e outro implicitamente declarado como tal (growItBigger( )):

```
public interface Growable {
    public abstract void growlt( ); // explicitamente public e abstract
    void growItBigger( ); // efetivamente public e abstract
}
```

Observe que, assim como acontece com os métodos abstratos nas classes, os métodos dentro de interfaces não possuem corpos. Lembre-se de que uma interface é puro design; nenhuma implementação está envolvida.

Além dos métodos, as interfaces também podem ter variáveis, mas essas variáveis devem ser declaradas publ i c, stati c e final (tornando-as constantes). Assim como ocorre nos métodos, você pode definir explicitamente uma variável para ser publ i c, stati c e final, ou ela é definida implicitamente como tal, se você não utilizar esses modificadores. A seguir há a mesma definição de Growable com duas novas variáveis:

```
public interface Growable {
    public static final int increment = 10;
    long maxnum = 1000000; // torna-se public static e final

    public abstract void growlt( ); //explicitamente public e abstract
    void growItBigger( ); // efetivamente public e abstract
}
```

As interfaces devem ter uma proteção pública ou de pacote, exatamente como as classes. Note, entretanto, que as interfaces sem o modificador publ i c não convertem automaticamente seus métodos em publ i c e abstract nem suas constantes para publ i c. Uma interface não-public também possui métodos e constantes não-public que podem ser usados apenas pelas classes e outras interfaces do mesmo pacote.

As interfaces, assim como as classes, podem pertencer a um pacote, se você incluir uma instrução package na primeira linha do arquivo de classe. As interfaces também podem importar outras interfaces e classes de outros pacotes, exatamente como acontece com as classes.

Métodos dentro de interfaces

A seguir há uma observação interessante a respeito de métodos dentro de interfaces: esses métodos são supostamente abstratos e se aplicam a qualquer tipo de classe, mas como você pode definir parâmetros para esses métodos? Você não sabe qual classe vai utilizá-los! A resposta reside no fato de que você usa um nome de interface em qualquer lugar que um nome de classe possa ser

usado, como aprendeu anteriormente. Definindo os parâmetros de seu método para serem tipos de interface, você pode criar parâmetros genéricos que se aplicam a qualquer classe que possa utilizar essa interface.

Assim, por exemplo, considere a interface Fruitlike, que define métodos (sem argumentos) para decay( ) e squish( ). Você também poderia ter um método para germinateSeeds( ), que possui um argumento: a própria fruta (fruit). De que tipo esse argumento vai ser? Ele não pode ser simplesmente Fruit, pois você pode ter uma classe que seja Fruitlike (isto é, uma que implemente a interface Fruitlike) sem ser realmente uma fruta. A solução é declarar o argumento simplesmente como Fruitlike na interface:


```
public interface Fruitlike {
    public abstract germinate(Fruitlike self) {
      // ...
    }
}
```

Então, em uma implementação real desse método em uma classe, você pode pegar o argumento genérico Fruitlike e fazer sua coerção para o objeto apropriado:

```
public class Orange extends Fruit {

public germinate(Fruitlike self) {
    Orange the Orange = (Orange)self;
    // •••
    }
}
```

Ampliando interfaces

Assim como acontece com as classes, você pode organizar as interfaces em uma hierarquia. Quando uma interface herda de outra, essa "subinterface" adquire todas as definições de método e constantes que sua "superinterface" definiu. Para ampliar uma interface, você usa a palavra-chave extends, exatamente como faz em uma definição de classe:

```
public interface Fruitlike extends Foodlike {

}
```

Note que, diferente das classes, a hierarquia de interface não possui o equivalente da classe Object; essa hierarquia não tem raiz em ponto algum. As interfaces podem existir completamente sozinhas ou herdar de outra interface.

Observe também que, diferente da hierarquia de classe, a hierarquia de herança possui propriedades de herança múltipla. Assim, por exemplo, uma única interface pode ampliar quantas classes precisar (separadas por vírgulas na parte extends da definição), e a nova interface conterá uma combinação de todos os métodos e constantes de origem. A seguir há uma definição para uma interface chamada Busylnterface, que herda de muitas outras interfaces:

```
public interface BusyInterface extends Runnable, Growable, Fruitlike,
    Observable {

    // ...
}
```

Nas interfaces de herança múltipla, as regras para gerenciamento de conflitos de nome de método são as mesmas das classes que utilizam várias interfaces; os métodos que diferem apenas no tipo de retorno resultam em um erro de compilador.

Um exemplo: enumerando listas encadeadas

Para concluir a lição de hoje, veja o seguinte exemplo, que usa pacotes, proteção de pacote e define uma classe que implementa a interface Enumeration (parte do pacote java.util). A listagem 15.3 mostra o código.

DIGITE

Listagem 15.3. O texto completo de linkedlist.java.

```
1: package collections;
2:
3: public class LinkedList {
4:     private Node root;
5:
6:    // ...
7:     public Enumeration enumerate( ) {
8:         return new LinkedListEnumerator(root);
9:    }
10: }
11:
12: class Node {
13:     private Object contents;
14:     private Node next;
15:
16:    // ...
17:     public Object contents( ) {
18:         return contents;
19:    }
20:
21:     public Node next( ) {
22:         return next;
23:    }
24: }
25:
26: class LinkedListEnumerator implements Enumeration {
27:     private Node currentNode;
28:
29:     LinkedListEnumerator(Node root) {
30:         currentNode = root;
31:    }
```

Listagem 15.3. Continuação

```
32:
33:      public boolean hasMoreElements( ) {
34:          return currentNode != null;
35:      }
36:
37:      public Object nextElement( ) {
38:          Object anObject = currentNode.contents( );
39:
40:          currentNode = currentNode.next( );
41:          return anObject;
42:   }
43: }
```


A seguir há uma utilização típica do enumerador:

```
collections.LinkedList aLinkedList = createLinkedList ( );
java.util.Enumeration e = aLinkedList.enumerate( );

while (e.hasMoreElements( )) {
    Object anObject = e.nextElement( );
    // faz algo útil com anObject
}
```

Observe que, embora você esteja usando o termo Enumeration e como se soubesse o que ele é, na verdade, você não sabe. De fato, trata-se de uma instância de uma classe oculta (LinkedListEnumerator) que você não pode ver nem usar diretamente. Usando uma combinação de pacotes e interfaces, a classe LinkedList conseguiu fornecer uma interface pública transparente para alguns de seus comportamentos mais importantes (através da interface java.util .Enumeration) já definida, ao mesmo tempo em que encapsula (oculta) suas duas classes de implementação (suporte).

A manipulação de um objeto como esse às vezes é chamada de venda. Freqüentemente, o "fornecedor" divulga um objeto que um recebedor não pode criar sozinho, mas que ele sabe como usar. Devolvendo o objeto para o fornecedor, o recebedor pode ver se ele tem certa capacidade, verificar por si mesmo ou realizar inúmeras tarefas úteis — tudo isso sem saber muito sobre o objeto vendido. Essa poderosa metáfora pode ser aplicada em uma ampla gama de situações.

Classes internas

Todas as classes com que você trabalhou até agora são membros de um pacote, ou porque foi especificado um nome de pacote com a instrução package ou porque o pacote padrão foi utilizado. As classes que pertencem a um pacote são conhecidas como classes de nível superior. Quando a linguagem Java apareceu, elas eram as únicas classes suportadas.

Iniciando com Java 1.1, você poderia definir uma classe dentro de outra, como se ela fosse um método ou uma variável. Esses tipos de classes são chamadas classes internas. A listagem 15.4 contém o applet Inner, que usa uma classe interna chamada BlueButton para representar os botões que podem receber um clique de mouse e que possuem a cor azul como fundo padrão.

DIGITE

Listagem 15.4. O texto completo de inner.java.

```
1: import java.awt.Button;
2: import java.awt.Color;
3:
4: public class Inner extends java.applet.Applet {
5:      Button b1 = new Button("0ne");
6:     BlueButton b2 = new BlueButton("Two");
7:
8:      public void init( ) {
9:           add(bl);
10:          add(b2);
11:     }
12:     class BlueButton extends Button {
13:          BlueButton(String label) {
14:               super(label);
15:               this.setBackground(Color.blue);
16:          }
17:     }
18: }
```

A Figura 15.1 foi produzida no appletviewer usando-se a seguinte tag HTML:

```
<applet code="Inner.class" width=100 height=100>
</applet>
```

Figura 15.1

O *applet Inner.*

Nesse exemplo, a classe BlueButton não é nada diferente de uma classe auxiliar que está incluída no mesmo arquivo-fonte como o arquivo de classe principal de um programa. A única diferença é que a classe auxiliar é definida dentro do arquivo de classe, o que traz diversas vantagens:

- As classes internas são invisíveis para todas as outras classes, o que significa que você não precisa se preocupar com conflitos de nome entre elas e outras classes.

- As classes internas podem ter acesso às variáveis e métodos dentro do escopo de uma classe de nível superior, acesso esse que não teriam como uma classe separada.

Em muitos casos, uma classe interna é um arquivo de classe pequeno que existe apenas para um propósito limitado. No applet Inner, como BlueButton não contém muito comportamento e atributos complexos, ela é conveniente para implementação como uma classe interna.

O nome de uma classe interna é associado ao nome da classe em que está contida, e ele é atribuído automaticamente, quando o programa é compilado. No exemplo da classe BlueButton, ela recebe o nome Inner$BlueButton.class pelo TDK.

Quando você está usando classes internas, deve tomar o máximo cuidado para incluir todos os arquivos . class, quando tornar um programa disponível. Cada classe interna possui seu próprio arquivo de classe e esses arquivos devem ser incluídos junto com todas as classes de nível superior. Se você publicar o applet Inner na World Wide Web, por exemplo, deve publicar os arquivos `Inner.class` e `Inner$BlueButton.class` juntos.

As classes internas, embora pareçam um aprimoramento secundário da linguagem Java, representam na verdade uma modificação significativa.

As regras que governam o escopo de uma classe interna combinam muito com as que governam variáveis. O nome de uma classe interna não é visível fora de seu escopo, exceto em um nome totalmente qualificado, o que ajuda na estruturação das classes dentro de um pacote. O código de uma classe interna pode usar nomes simples de escopos subjacentes, incluindo variáveis de classe e membro de classes subjacentes, bem como variáveis locais de blocos subjacentes.

Além disso, você pode definir uma classe de nível superior como um membro estático de outra classe de nível superior. Ao contrário do que acontece com uma classe interna, uma classe de nível superior não pode usar diretamente as variáveis de instância de qualquer outra classe. A capacidade de aninhar classes dessa maneira permite que qualquer classe de nível superior ofereça uma organização no estilo dos pacotes para um grupo de classes de nível superior secundário logicamente relacionadas.

Resumo

Hoje, você aprendeu a encapsular um objeto usando modificadores de controle de acesso para suas variáveis e métodos. Você também aprendeu a usar outros modificadores, como `static`, `final` e `abstract`, no desenvolvimento de classes Java e hierarquias de classe.

Para aumentar o trabalho de desenvolvimento de um conjunto de classes e utilizá-las, você aprendeu como as classes podem ser agrupadas em pacotes. Esses agrupamentos organizam melhor seus programas e permitem o compartilhamento de classes com os muitos outros programadores Java que estão tornando seu código disponível publicamente.

Finalmente, você aprendeu a implementar interfaces e classes internas, duas estruturas que são úteis ao se fazer o projeto de uma hierarquia de classe.

Perguntas e respostas

P **O uso de métodos de acesso em toda parte não diminui a velocidade de meu código Java?**

R Nem sempre. Quando os compiladores Java melhorarem e puderem criar mais otimizações, eles poderão tornar os métodos de acesso rápidos automaticamente, mas se você está preocupado com a velocidade, sempre pode declarar os métodos de acesso como final e eles serão tão rápidos quanto os acessos diretos à variável de instância.

P **Os métodos de classe (static) são herdados exatamente como os métodos de instância?**

R Não. Os métodos static (de classe) agora são final por definição. Como, então, você poderia declarar um método de classe não final? A resposta é que você não pode. A herança de métodos de classe não é permitida, quebrando a simetria com os métodos de instância.

P **Com base no que aprendi, os métodos private abstract e os métodos ou classes final abstract não parecem fazer sentido. Eles são válidos?**

R Não, eles são erros de compilação, como você supôs. Para serem úteis, os métodos abstract devem ser anulados e as classes abstract devem se transformar em subclasses, mas nenhuma dessas duas operações seria válida, se eles também fossem private ou final.

SEMANA 3

Dia 16

Circunstâncias especiais: tratamento de erros e segurança

Programadores de qualquer linguagem se empenham em escrever programas livres de erro, que nunca falham, que podem tratar de qualquer situação sem problemas e que possam se recuperar de situações incomuns sem causar nenhuma tensão indevida ao usuário. Deixando de lado as boas intenções, programas como esses não existem.

Em programas reais, os erros ocorrem porque o programador não antecipou todas as situações que seu código poderia enfrentar (ou não teve tempo para testar suficientemente o programa) ou por causa de situações fora do seu controle — dados errados dos usuários, arquivos danificados que não possuem os dados corretos, conexões de rede que não se estabelecem, dispositivos de hardware que não respondem, manchas de sol, gnomos, qualquer coisa.

Na linguagem Java, essas espécies de eventos estranhos, que podem fazer um programa falhar, são chamadas de *exceções*. A linguagem Java define vários recursos que tratam de exceções, incluindo o seguinte:

- Como tratar delas em seu código e recuperar-se de problemas em potencial
- Como dizer à linguagem Java e aos usuários de seus métodos que você está esperando uma possível exceção
- Como criar uma exceção, caso você detecte alguma
- Como seu código é limitado, embora mais robusto por causa das exceções

Além das exceções, você aprende o sistema que está sendo estabelecido para Java 2, que permite aos applets realizarem coisas em um programa que normalmente causariam exceções de segurança.

Exceções, o velho e confuso caminho

Na maioria das linguagens de programação, as condições de tratamento de erros exigem muito mais trabalho do que o tratamento de um programa que está funcionando corretamente. Ele pode criar uma estrutura bastante confusa de instruções, semelhantes em funcionalidade aos blocos `if...else` e switch da linguagem Java, para tratar dos erros que podem ocorrer.

Como exemplo, considere as seguintes instruções, que mostram a estrutura de como um arquivo poderia ser carregado do disco. Carregar um arquivo é algo que pode ser problemático, por causa do número de diferentes circunstâncias — erros de disco, erros de arquivo não encontrado e coisas assim. Se o programa precisa ter os dados do arquivo para funcionar corretamente, ele deve tratar de todas essas circunstâncias, antes de continuar.

A seguir há a estrutura de uma possível solução:

```
int status = loadTextfile( );
if (status != 1) (
    // algo incomum aconteceu, descreva o fato
    switch (status) {
        case 2:
            // arquivo não encontrado
        case 3:
            // erro de disco
        case 4:
            // arquivo danificado
        default:
            // outro erro
    }
} else {
    // arquivo carregado com sucesso, continue o programa
(
```

Esse código tenta carregar um arquivo com uma chamada de método para loadTextfile( ), que foi definido em outra parte do programa. Esse método retorna um inteiro que indica se o arquivo foi carregado corretamente (status == 1) ou se um erro ocorreu (status igual a qualquer diferente de 1).

Dependendo do erro que ocorre, o programa utiliza uma instrução switch para tentar contorná-lo. O resultado final é um bloco de código elaborado, em que a maior parte das circunstâncias comuns — um carregamento de arquivo com sucesso — pode ser perdida no meio do código de tratamento de erros. Isso apenas para tratar de um possível erro. Se houver outros erros que poderão ocorrer posteriormente em seu programa, você poderá acabar com mais blocos `if...else` e switch/case aninhados.

O gerenciamento de erros pode se tornar um problema sério quando você começa a criar sistemas maiores. Diferentes programadores podem usar diferentes valores especiais para o tratamento de erros e podem não documentá-los bem, se é que os documentam. Você pode usar erros incoerentemente em seus programas. O código para gerenciar esses tipos de erros freqüentemente podem obscurecer o objetivo original do programa, tornando esse código difícil de ler e manter. Finalmente, se você tentar tratar de erros desse modo atabalhoado, não haverá uma maneira fácil para o compilador verificar a coerência, do jeito que ele pode fazer para certificar-se de que você chamou um método com os argumentos corretos.

Embora o exemplo anterior utilize sintaxe Java, você não precisa tratar com os erros dessa forma em seus programas. A linguagem apresenta um modo melhor para tratar de circunstâncias excepcionais em um programa: com o uso de um grupo de classes chamadas exceções.

As exceções incluem erros que poderiam ser fatais para seu programa, mas também incluem outras situações incomuns. Gerenciando as exceções, você pode gerenciar erros e, possivelmente, contorná-los.

Através de uma combinação de recursos especiais da linguagem, verificação da coerência durante a compilação e de um conjunto de classes de exceção extensíveis, os erros e outras condições incomuns podem ser gerenciados muito mais facilmente nos programas Java.

Dados esses recursos, você pode agora acrescentar uma dimensão completamente nova ao comportamento e ao design de suas classes, de sua hierarquia de classe e de todo o seu sistema. Suas definições de classe e interface descrevem como seu programa deve supostamente se comportar, dadas as melhores circunstâncias. Integrando o tratamento de exceções no design de seu programa, você pode descrever de forma consistente como ele se comportará quando as circunstâncias não forem muito boas e permitirá que as pessoas que utilizam suas classes saibam o que esperar nesses casos.

Exceções Java

Neste ponto do livro, as chances são de que você tenha encontrado pelo menos uma exceção Java — talvez você tenha digitado um nome de método errado ou cometido um engano em seu código que causou um problema. Talvez você tenha tentado executar um applet Java escrito usando-se a versão 2 da linguagem, em um navegador que ainda não oferece suporte a ela, e visto uma mensagem Security Exception na linha de status do navegador.

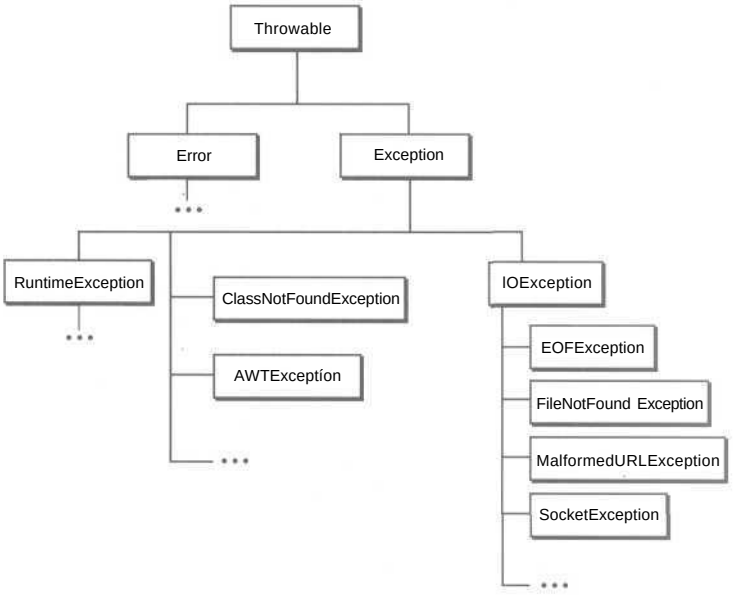
Existe a possibilidade de que um programa feche e produza a apresentação de vários erros na tela. Esses erros misteriosos são exceções. Quando seu programa fecha, é porque uma exceção foi *levantada* (registrada). As exceções podem ser levantadas pelo sistema ou explicitamente pelos programas que você escreve.

O termo "levantada" é adequado, pois as exceções também podem ser capturadas. Capturar uma exceção envolve o tratamento com uma circunstância excepcional para que seu programa não pare — você aprenderá mais sobre isso posteriormente. *Uma exceção foi levantada* é a terminologia da linguagem Java correta para "um erro aconteceu".

A parte central do sistema de exceções Java é a própria exceção. As exceções na linguagem Java são objetos reais, instâncias de classes que herdam da classe Throwable. Uma instância de uma classe Throwable é criada, quando uma exceção é levantada. A Figura 16.1 mostra uma hierarquia de classe parcial para exceções.

Figura 16.1

A *hierarquia de classe de exceção.*



Throwable possui duas subclasses: Error e Exception. As instâncias de Error são erros internos no ambiente de runtime java (a máquina virtual). Esses erros são raros e normalmente fatais; não há muito que se possa fazer a respeito deles (capturá-los ou levantá-los por si mesmo), mas eles existem para que a linguagem Java possa usá-los, se for necessário.

A classe Exception é mais interessante. As subclasses de Exception caem em dois grupos gerais:

- Exceções de runtime (subclasses da classe RuntimeException), como ArraylndexOutofBounds, SecurityException ou NullPointerException
- Outras exceções, como EOFException e Mal formedURLException

As exceções de runtime ocorrem normalmente depois de código que não é muito robusto. Uma exceção ArraylndexOutofBounds, por exemplo, nunca

deve ser levantada, se você estiver certificando-se corretamente de que seu código fica dentro dos limites de um array. As exceções NullPointerException não acontecerão, a não ser que você tente usar uma variável antes que ela tenha sido definida para conter um objeto.

Se seu programa está causando exceções de runtime sob quaisquer circunstâncias, você deve corrigir esses problemas antes mesmo de começar a tratar do gerenciamento de exceções.

O grupo de exceções final é o mais interessante, pois essas são as exceções que indicam que algo muito estranho e fora do controle está acontecendo. EOFExceptions, por exemplo, acontece quando você está lendo um arquivo e esse arquivo termina antes do esperado. As exceções MalformedURLException acontecem quando um URL não está no formato correto (talvez seu usuário o tenha digitado errado) .Esse grupo inclui exceções que você cria para sinalizar casos incomuns que podem ocorrer em seus programas.

As exceções são organizadas em uma hierarquia, assim como as outras classes, onde as superclasses Exception são os erros mais gerais e as subclasses são os erros mais específicos. Essa organização se tornará mais importante para você, quando tratar com exceções em seu próprio código.

A maior parte das classes de exceção faz parte do pacote java.lang (incluindo Throwable, Exception e RuntimeException). Muitos outros pacotes definem outras exceções e essas exceções são usadas em toda a biblioteca de classe. Por exemplo, o pacote java.io define uma classe de exceção geral chamada IOException, que se transforma em uma subclasse não apenas no pacote java.io para exceções de entrada e saída (EOFException, FileNotFound-Exception), mas também nas classes java.net para exceções de interligação em rede, como Mal FormedURLException.

Gerenciando exceções

Agora que você sabe o que é uma exceção, como tratar delas em seu próprio código? Em muitos casos, o compilador Java impõe um gerenciamento de exceção, quando você tenta usar métodos que utilizam exceções; você precisa tratar dessas exceções em seu próprio código ou ele simplesmente não compilará. Nesta seção, você aprende sobre essa verificação de coerência e como usar as palavras-chave try, catch e finally da linguagem, para tratar das exceções que possam ocorrer.

Verificação da coerência da exceção

Quanto mais você trabalhar com as bibliotecas de classe Java, mais provavelmente encontrará um erro de compilador (uma exceção!) semelhante ao seguinte:

```
BoogieDown.java:32: Exception java.lang.InterruptedException
must be caught or it must be declared in the throws clause
of this method.
```

O que será que significa isso? Em Java um método pode indicar os tipos de erros que ele possivelmente pode levantar. Por exemplo, os métodos que lêem arquivos podem levantar erros IOException, de modo que esses métodos são declarados com um modificador especial que indica erros em potencial. Quando você usa esses métodos em seus próprios programas Java, tem de proteger seu código dessas exceções. Essa regra é imposta pelo próprio compilador, da mesma maneira que o compilador verifica se você está usando métodos com o número correto de argumentos e se todos os seus tipos de variável combinam com o que lhes está sendo atribuído.

Por que essa verificação? Ela torna seus programas menos propensos a falhas com erros fatais, pois você sabe, antecipadamente, o tipo de exceções que podem ser levantadas pelos métodos que um programa utiliza. Você não precisa mais ler cuidadosamente a documentação ou o código de um objeto que vai usar, para garantir que tratou de todos os problemas em potencial — a linguagem Java faz a verificação para você. Por outro lado, se você definir seus métodos de modo que eles indiquem as exceções que podem levantar, a linguagem Java poderá dizer aos usuários de seus objetos para que tratem desses erros.

Protegendo o códiqo e capturando exceções

Suponha que você esteja codificando tranqüilamente e se depare com essa mensagem de exceção, durante um teste de compilação. De acordo com a mensagem, você tem de capturar o erro ou declarar que seu método o levanta. Trate do primeiro caso: capturar exceções em potencial.

Você faz duas coisas para capturar uma exceção:

- Você protege o código que contém o método que poderia levantar uma exceção dentro de um bloco try.
- Você testa e trata de uma exceção dentro de um bloco catch.

O que try e catch significam efetivamente é: "Experimente esse trecho de código que poderia causar uma exceção. Se ele executar sem problemas, continue com o programa. Se ele não executar, capture a exceção e trate dela."

Você já viu try e catch antes, quando lidou pela primeira vez com threads. No Dia 10, você aprendeu sobre um applet que criava um relógio digital; era feita uma pausa na animação uma vez por segundo, usando este trecho de código:

```
try {
        Thread.sleep(lOOO);
} catch (InterruptedException e) { }
```

Embora esse exemplo use try e catch, esse não é um bom uso deles. A seguir está o que está acontecendo nessas instruções: O método de classe Thread.sleep( ) poderia levantar uma exceção do tipo InterruptedException, o que significa que o thread foi interrompido por algum motivo.

Para tratar dessa exceção, a chamada a sleep( ) é colocada dentro de um bloco try e um bloco catch associado foi definido. Esse bloco catch recebe todos os objetos InterruptedException que são levantados dentro do bloco try.

O motivo de esse não ser um bom exemplo de tratamento de exceção é que não há nada dentro da cláusula catch — em outras palavras, você capturará a exceção se ela acontecer, mas não fará nada para responder a essa ocorrência. Em todos os casos, menos nos mais simples (como este, onde a exceção realmente não importa), você vai precisar de algo dentro do bloco catch que faça alguma coisa para realizar a limpeza, depois que a exceção acontecer.

A parte da cláusula catch que está entre parêntese é semelhante à lista de argumentos de uma definição de método. Ela contém a classe da exceção a ser capturada e um nome de variável (e é usado normalmente). Você pode fazer referência a esse objeto de exceção dentro do bloco catch.

Um uso comum desse objeto é para chamar seu método getMessage( ). Esse método está presente em todas as exceções e apresenta uma mensagem de erro detalhada, descrevendo o que aconteceu.

O exemplo a seguir é uma versão revisada da instrução try...catch utilizada no applet DigitalClock do Dia 10:

```
try {
      Thread.sleep(lOOO);
} catch (InterruptedException e) {
       System.out.println("Error: " + e.getMessage( ));
}
```

Para ver outro exemplo, reveja o assunto sobre tratamento de arquivos em Java. Se você possui um programa que lê um arquivo, é provável que ele utilize uma das classes de fluxo de entrada/saída, sobre as quais você aprendeu no Dia 17. A idéia básica é que você abre uma conexão com um arquivo e utiliza o método read ( ) para obter dados dele. Isso pode causar várias exceções, como um erro de disco ou uma tentativa de ler mais dados do que o arquivo contém. Nesses dois casos, o método read( ) levanta uma IOException, o que faria o programa parar de ser executado, se você não a capturou, ou faria o programa falhar.

Colocando seu método read ( ) dentro de um bloco try, você pode então tratar tranqüilamente desse erro, dentro de um bloco catch. Você poderia fazer a limpeza após o erro e retornar para algum estado seguro, acertando as coisas o suficiente para o programa prosseguir ou, se tudo falhar, pode salvar o máximo possível do estado atual do programa e sair.

O exemplo a seguir tenta ler o arquivo e captura as exceções, se elas acontecerem:

```
try {
    while (numBytes <= mybuffer.length) {
        mylnputStream.read(myBuffer);
        numBytes++;
    }
} catch (IOException e) {
    System.out.println("Oops! I0 Exception - only read " + numBytes);
    // outro código de limpeza
}
```

Aqui, o "outro código de limpeza" pode ser qualquer coisa; você pode continuar com o programa usando as informações parciais que obteve do arquivo ou talvez queira apresentar uma caixa de diálogo permitindo que o usuário selecione um arquivo diferente.

Os exemplos vistos até aqui capturam um tipo específico de exceção. Como as classes de exceção são organizadas em uma hierarquia e você pode usar uma subclasse em qualquer lugar que uma superclasse seja esperada, é possível capturar grupos de exceções dentro da mesma instrução catch.

Como exemplo, existem vários tipos diferentes de exceções IOException, como EOFException e FileNotFoundException. Capturando IOException, você também captura instâncias de qualquer subclasse IOException.

E se você quiser capturar tipos muito diferentes de exceções, que não estejam relacionados pela herança? Você pode usar vários blocos catch para um único try, como segue:

```
try {
    // código que poderia gerar exceções
} catch (IOException e) {
    // trata de exceções de I0
} catch (ClassNotFoundException e2) {
    // trata de exceções de classe não encontrada
} catch (InterruptedException e3) {
    // trata de exceções interrompidas
}
```

Como o escopo das variáveis locais em uma instrução catch é o mesmo do bloco mais externo em que o bloco try...catch está, você precisa usar diferentes variáveis locais para cada instrução catch específica.

Em um bloco catch múltiplo, o primeiro bloco que combinar será executado e o restante ignorado.

A cláusula finally

Suponha que exista alguma ação em seu código que seja absolutamente necessária, independentemente do que aconteça, seja levantada uma exceção ou não. Isso normalmente serve para liberar algum recurso externo após sua aquisição, para fechar um arquivo após sua abertura ou algo semelhante. Embora você pudesse colocar essa ação dentro de um bloco catch e fora dele, isso estaria duplicando o mesmo código em dois lugares diferentes. Em vez

disso, coloque uma cópia desse código dentro de uma parte opcional especial do bloco try.. .catch, chamada finally. O exemplo a seguir mostra como um bloco try... catch... finally é estruturado:

```
try {
     readTextfile( );
} catch (IOException e) {
     // trata dos erros de I0
} finally {
    closeTextfile( );
}
```

A instrução finallyé realmente útil fora das exceções; você também pode utilizá-la para executar código de limpeza, após um return, um break ou um continue, dentro do loop. Para este último caso, você pode usar uma instrução try com uma cláusula f i nal ly, mas sem uma instrução catch.

O próximo projeto mostra como uma instrução finally pode ser usada dentro de um método.

DIGITE

Listagem 16.1. O texto completo de final .java.

```
1: class Final {
2:     int[ ] numl = { 12, 15, 10, 8, -1, 7 };
3:     int[ ] num2 = { 1, 5, 20, 8, 1, 13 };
4:
5:     public static void main(String[ ] arguments) {
6:         Final fin = new Final( );
7:         System.out.println("First array: ");
8:         fin.readNumbers(fin.numl);
9:        System.out.println("Second array: " ) ;
10:        fin.readNumbers(fin.num2);
11:    }
12:
13:    void readNumbers(int[ ] numArray) {
14:        int count = 0;
15:        int lastNum = 0;
16:        try {
17:            while (count < numArray.length) {
18:                lastNum = numArray[count++];
19:                if (lastNum =  -1)
20:                    return;
21:            }
22:        } finally {
23:            System.out.println("Last number read: " + lastNum);
24:       }
25:       return;
26:    }
27: }
```

A saída desse programa é a seguinte:

```
First array:
Last number read: -1
Second array:
Last number read: 13
```

O bloco try...finally, nas linhas 16 a 24, faz algo incomum acontecer, quando a instrução return é encontrada na linha 20. Você esperaria que return fizesse o método ser abandonado imediatamente.

Como ele está dentro de um bloco try...finally, as instruções que estão dentro do bloco finally são executadas, independentemente de como se sai do bloco try. O texto "Last number read" sempre é apresentado.

Declarando métodos que podem levantar exceções

Em exemplos anteriores, você aprendeu a lidar com métodos (protegendo código e capturando todas as exceções que ocorrem) que podem levantar exceções. O compilador Java verifica se você tratou de algum modo das exceções de um método. Mas como ele sabia sobre quais exceções teria de informá-lo antecipadamente?

A resposta é que o método original indicado em sua assinatura contém as exceções que ele poderia levantar. Você pode usar esse mecanismo em seus próprios métodos — na verdade, é considerado um bom estilo de programação fazer isso para certificar-se de que os outros usuários de suas classes sejam alertados sobre os erros que seus métodos podem enfrentar.

Para indicar que um método pode levantar uma exceção, você usa uma cláusula especial na definirão de método, chamada throws

A cláusula throws

Para indicar que algum código no corpo de seu método pode levantar uma exceção, basta incluir a palavra-chave throws após a assinatura do método (antes da chave de abertura), com o nome (ou nomes) da exceção que ele levanta:

```
public boolean myMethod (int x, int y) throws AnException {
    // ...
}
```

Se seu método pode levantar vários tipos de exceções, você pode colocar todos eles na cláusula throws, separados por vírgulas:

```
public boolean myOtherMethod (int x, int y)
    throws AnException, AnotherException, AThirdException {
      // ...
}
```

Note que, assim como acontece com catch, você pode usar uma superclasse de um grupo de exceções para indicar que seu método pode levantar qualquer subclasse dessa exceção:

```
public void YetAnotherMethod( ) throws IOException {
    // ...
}
```

Lembre-se de que adicionar um método throws à sua definição de método significa simplesmente que o método pode levantar uma exceção se algo der errado, e não que ele realmente o fará. A cláusula throws simplesmente fornece informações extras para sua definição de método a respeito de exceções em potencial e permite que a linguagem Java certifique-se de que seu método está sendo usado corretamente por outras pessoas.

Considere a descrição global de um método como um contrato entre o projetista desse método (ou classe) e quem o chama (você pode estar em qualquer lado desse contrato, é claro). Normalmente, a descrição indica os tipos dos argumentos de um método, o que ele retorna e a semântica geral do que ele normalmente faz. Usando throws, você também inclui informações sobre as coisas anormais que ele pode fazer. Essa nova parte do contrato ajuda a separar e tornar explícitos todos os lugares em que as condições excepcionais devem ser tratadas em seu programa e que tornam mais fácil o projeto em larga escala.

Quais exceções você deve levantar?

Ao decidir declarar que seu método pode levantar uma exceção, você precisa decidir também quais exceções ele pode levantar (e realmente levantá-las ou chamar um método que as levante — você aprenderá levantar suas próprias exceções na próxima seção). Em muitas instâncias isso é aparente a partir da operação do próprio método. Talvez você esteja criando e levantando suas próprias exceções e nesse caso saberá exatamente quais deve levantar.

Você não tem de listar todas as exceções possíveis que seu método poderia levantar; algumas delas são tratadas pela própria execução e são tão comuns (não comuns em si, mas ubíquas) que você não precisa tratar delas. Em particular, as exceções da classe Error ou RuntimeException (ou qualquer uma de suas subclasses) não precisam estar listadas em sua cláusula throws. Elas recebem tratamento especial, pois podem ocorrer em qualquer parte dentro de um programa Java e normalmente são condições que você, como programador, não causou diretamente. Um bom exemplo é OutOfMemoryError, que pode acontecer em qualquer lugar, a qualquer momento e por diversos motivos. Esses dois tipos de exceções são chamadas de *exceções implícitas* e você não precisa se preocupar com elas.

As exceções implícitas são aquelas que são subclasses de RuntimeException e Error. As exceções implícitas normalmente são levantadas pela própria execução de Java. Você não precisa declarar que seu método as levanta.

Nota

É claro que, se quiser, você pode optar por listar esses erros e exceções de runtime em sua cláusula throws, mas quem chamar seus métodos não será obrigado a tratar deles; apenas as exceções que não são de runtime, ou seja, de execução, devem ser tratadas.

Todas as outras exceções são denominadas *exceções explícitas* e são candidatas em potencial para uma cláusula throws em seu método.

Passando exceções

Além de declarar métodos que levantam exceções, existe uma outra instância em que sua definição de método pode incluir uma cláusula throws. Nesse caso, você quer usar um método que levanta uma exceção, mas não quer capturar ou tratar dela. Em muitos casos poderia fazer mais sentido para o método que chama o seu, tratar dessa exceção, em vez de você tratar dela. Não há nada de errado com isso; trata-se de uma ocorrência bastante comum o fato de você não tratar de uma exceção, mas passá-la de volta para o método que chama o seu. De qualquer maneira, é melhor passar as exceções para os métodos que estão chamando, do que capturá-las e ignorá-las.

Em vez de usar as cláusulas try e catch no corpo de seu método, você pode declarar seu método com uma cláusula throws, de tal modo que ele também possa levantar a exceção apropriada. Assim, tratar dessa exceção fica sendo responsabilidade do método que chama o seu. Esse é o outro caso que satisfaz ao compilador Java, em que você fez algo com determinado método. Aqui está outra maneira de implementar um exemplo que lê caracteres de um fluxo:

```
public void readFile(String filename) throws IOException {
    // abre o arquivo, inicializa o fluxo aqui
    while (numBytes <= myBuffer.length) {
      myInputStream.read(myBuffer);
    numBytes;++
}
```

Esse exemplo é semelhante a outro, utilizado anteriormente hoje; lembre-se de que o método read( ) foi declarado para levantar uma IOException, de modo que você tinha de usar try e catch para utilizá-lo. Entretanto, uma vez que você declare seu método para levantar uma exceção, pode usar outros métodos que também levantam essas exceções dentro do corpo desse método, sem a necessidade de proteger o código ou capturar a exceção.

Nota

É claro que você pode tratar de outras exceções usando try e catch no corpo de seu método, além de passar as exceções listadas na cláusula throws. Você também pode tratar da exceção de algum modo e depois levantá-la novamente, para que o método que está chamando o seu tenha de tratar dela de qualquer maneira. Você aprenderá a levantar métodos na próxima seção.

throws e herança

Se sua definição de método anula um método em uma superclasse que inclui uma cláusula throws, existem regras especiais para o seu método anulado tratar de throws. Diferente das outras partes da assinatura do método, que devem imitar as do método que está anulando, seu novo método não exige o mesmo conjunto de exceções listado na cláusula throws.

Como existe uma possibilidade de que seu novo método possa tratar melhor das exceções, em vez de apenas levantá-las, é possível que o seu método levante menos tipos de exceções. Ele poderia até não levantar exceção alguma. Isso significa que você pode ter as duas definições de classe a seguir e tudo funcionaria muito bem:


```
public class RadioPlay {
    public void startPlaying( ) throws SoundException {
      // ...
    }
}
public class StereoPlay extends RadioPlay {
    public void startPlaying( ) {
      // ...
    }
}
```

O inverso dessa regra não é verdadeiro: um método de subclasse não pode levantar mais exceções (ou exceções de tipos diferentes ou classes de exceção mais gerais) do que seu método de superclasse.

Criando e levantando suas próprias exceções

Existem dois lados em cada exceção: o que levanta a exceção e o que a captura. Uma exceção pode ser levantada várias vezes para vários métodos, antes de ser capturada, mas finalmente ela será capturada e tratada.

O que faz o levantamento? De onde vêm as exceções? Muitas exceções são levantadas pelo runtime Java ou por métodos dentro das próprias classes Java. Você também pode levantar qualquer uma das exceções padrão que as bibliotecas de classe Java definem, ou pode criar e levantar suas próprias exceções. Esta seção descreve tudo isso.

Levantando exceções

Declarar que seu método levanta uma exceção é útil apenas para os usuários de seu método e para o compilador Java, que faz uma verificação para certificar-se de que todas as suas exceções estão sendo tratadas — mas a declaração em si não faz nada para realmente levantar essa exceção, se ela ocorrer; você mesmo tem de fazer isso no corpo do método.

Lembre-se de que todas as exceções são instâncias de alguma classe de exceção, da qual existem muitas definidas na biblioteca de classe Java padrão.

Para levantar uma exceção, você precisa criar uma nova instância de uma classe de exceção. Quando você tiver essa instância, use a instrução throw para levantá-la. O modo mais simples de levantar uma exceção é o seguinte:

```
NotInServiceException( ) nis = new NotInServiceException( );
throw nis;
```

Você só pode levantar objetos que são subclasses de Throwable. Isso difere das exceções do C++, que permitem levantar objetos de qualquer tipo.

Dependendo da classe de exceção que você estiver usando, a exceção também poderá ter argumentos para seu construtor, que podem ser usados. O mais comum deles é um argumento de string, que permite descrever o problema real com mais detalhes (o que pode ser muito útil para fins de depuração). A seguir há um exemplo:

```
NotInServiceException( ) nis = new
    NotInServiceException("Exception: Database Not in Service");
throw nis;
```

Quando uma exceção é levantada, o método é finalizado imediatamente, sem executar nenhum outro código (além do código dentro de finally, se esse bloco existir) e sem retornar um valor. Se o método que está chamando não tiver um try ou catch subjacente à chamada de seu método, o programa poderá muito bem sair, baseado na exceção que você levantou.

Criando suas próprias exceções

Embora exista um grande número de exceções na biblioteca de classe Java que podem ser usadas em seus próprios métodos, talvez você precise criar suas próprias exceções para tratar de diferentes tipos de erros que seus programas enfrentam. Felizmente, criar novas exceções é fácil.

Sua nova exceção deve herdar de alguma outra exceção da hierarquia Java. Todas as exceções criadas pelo usuário devem fazer parte da hierarquia Exception e não da hierarquia Error, que é reservada para erros que envolvem a máquina virtual Java. Procure uma exceção que seja parecida com a que você está criando; por exemplo, uma exceção para um formato de arquivo errado seria, logicamente, uma IOExcepti on. Se você não puder encontrar uma exceção intimamente relacionada para sua nova exceção, considere herdar de Excepti on, que forma o "topo" da hierarquia de exceção para exceções explícitas. (Lembre-se de que as exceções implícitas, que incluem as subclasses de Error e RuntimeException, herdam de Throwable.)

As classes de exceção normalmente têm dois construtores: o primeiro não recebe argumentos e o segundo recebe um string como único argumento.

Neste último caso, você deve chamar super( ) nesse construtor, para certificar-se de que o string seja aplicado no lugar correto da exceção.

Além dessas três regras, as classes de exceção são exatamente como as outras classes. Você pode colocá-las em seus próprios arquivos-fonte e compilá-las exatamente como faria com as outras classes. Por exemplo:

```
public class SunSpotException extends Exception {
    public SunSpotException( ) { }
    public SunSpotException(String msg) {
      super(msg);
    }
}
```

Combinando throws, try e throw

E se você quiser combinar todas as estratégias apresentadas até aqui? Você mesmo gostaria de tratar das exceções que chegam em seu método, mas também gostaria de passar a exceção para quem o chamou. Simplesmente usar try e catch não passa a exceção e apenas incluir uma cláusula throws não lhe dá a oportunidade de tratar da exceção. Se você quiser gerenciar a exceção e passá-la para o chamador, use os três mecanismos: a cláusula throws, a instrução try e uma instrução throw literalmente levantando novamente a exceção.

```
public void responsibleExceptionalMethod( ) throws IOException {
    MessageReader mr = new MessageReader( );

    try {
        mr.loadHeader( );
    } catch (IOException e) {
        // faz algo para tratar da
        // exceção de I0
        throw e; // levanta novamente a exceção
    }
)
```

Isso funciona porque as rotinas de tratamento de exceção podem ser aninhadas. Você pode tratar da exceção fazendo algo responsável com ela, mas também pode decidir que é muito importante não dar a uma rotina de tratamento de exceções, que poderia estar em seu chamador, uma chance de tratá-la. As exceções fluem para cima no encadeamento de chamadores de método desse modo (normalmente não sendo tratadas pela maioria deles), até que finalmente o próprio sistema trate das que não foram capturadas, abortando seu programa e imprimindo uma mensagem de erro. Essa não é uma má idéia em um programa independente, mas em um applet pode fazer um navegador falhar. A maioria dos navegadores se protege desse desastre capturando todas as exceções por si mesmos, ao executar um applet, mas você nunca pode saber ao certo. Se você puder capturar uma exceção e fazer algo inteligente com ela, então faça isso.

Quando usar e quando não usar exceções

Como o levantamento, a captura e a declaração de exceções são conceitos relacionados e podem ser muito confusos, aqui está um breve resumo de quando fazer o quê.

Quando usar exceções

Você tem três possibilidades se o seu método chama outro que possui uma cláusula throws:

- Tratar da exceção usando instruções try e catch.
- Passar a exceção para o encadeamento que está chamando, incluindo sua própria cláusula throws em sua definição de método.
- Efetuar os dois métodos anteriores capturando a exceção com catch e depois levantá-la outra vez explicitamente usando throw.

Nos casos em que um método levanta mais de uma exceção, você pode tratar de cada uma delas de forma diferente. Por exemplo, você poderia capturar algumas dessas exceções, enquanto permitisse que outras passem para o encadeamento que está chamando.

Se seu método levanta suas próprias exceções, você deve declarar que ele levanta esses métodos usando a instrução throws. Se seu método anula um método de superclasse que possui uma instrução throws, você pode levantar os mesmos tipos de exceções ou subclasses dessas exceções; você não pode levantar todos os tipos diferentes de exceções.

Por último, se seu método foi declarado com uma cláusula throws, não se esqueça de levantar realmente a exceção no corpo dele, usando a instrução throw.

Quando não usar exceções

Existem vários casos em que você não deve usar exceções, mesmo que elas pareçam apropriadas no momento.

Primeiramente, você não deve usar exceções se a exceção é algo esperado e poderia ser evitada facilmente com uma simples expressão. Por exemplo, embora você possa contar com uma exceção ArrayIndexOutofBounds para indicar quando o final do array foi ultrapassado, é fácil utilizar a variável 1 ength do array para evitar a extrapolação dos limites.

Além disso, se seus usuários vão introduzir dados que devem ser inteiros, fazer um teste para garantir que eles sejam inteiros é uma idéia muito melhor do que levantar uma exceção e tratar dela em outra parte.

As exceções ocupam muito tempo de processamento de seu programa Java. Um simples teste ou uma série de testes serão executados muito mais rapidamente do que o tratamento da exceção e tornarão o seu programa mais

eficiente. As exceções devem ser usadas apenas para casos verdadeiramente excepcionais, que estejam fora de seu controle.

Também é fácil ficar encantado com as exceções e tentar certificar-se de que todos os seus métodos tenham sido declarados para levantar todas as exceções possíveis. Isso torna seu código mais complexo de modo geral; além disso, se outras pessoas forem usar o seu código, elas terão de tratar de todas as exceções que seus métodos poderiam levantar. Quando você se entusiasma com as exceções, está dando mais trabalho para todos os envolvidos. Declarar um método para levantar poucas ou muitas exceções é uma questão de compromisso; quanto mais exceções seu método pode levantar, mais complexa é a utilização desse método. Declare apenas as exceções que tenham uma chance razoavelmente boa de acontecer e que façam sentido para o design global de suas classes.

Má utilização das exceções

Quando você começa a usar exceções, pode ser atraente contornar os erros de compilador que resultam ao utilizar um método que declarou instruções throws. Embora seja válido incluir uma cláusula catch vazia ou incluir uma instrução throws em seu próprio método (e existam razões apropriadas para se fazer essas duas coisas), abandonar intencionalmente as exceções sem tratar delas subverte as verificações que o compilador Java realiza para você.

O sistema de exceção da linguagem Java foi projetado de modo que, se um erro pode ocorrer, você é avisado a respeito. Ignorar esses avisos e contorná-los possibilita que erros fatais ocorram em seu programa — erros que você poderia ter evitado com algumas linhas de código. Pior ainda, incluir instruções throws em seus métodos para evitar exceções significa que os usuários de seus métodos (objetos mais acima no encadeamento de chamada) terão de tratar delas. Você apenas tornou seus métodos mais difíceis de usar.

Os erros de compilador relacionados às exceções estão lá para lembrá-lo de refletir sobre esses problemas. Você deve tratar das exceções que podem afetar o seu código. Esse cuidado extra será muito recompensado quando você reutilizar suas classes em projetos posteriores e em programas cada vez maiores. É claro que a biblioteca de classe Java foi escrita exatamente com esse nível de detalhe e esse é um dos motivos de ela ser suficientemente robusta para ser usada na criação de seus projetos Java.

Usando assinaturas digitais para identificar applets

Uma das suposições fundamentais da estratégia de segurança de applet da linguagem Java é que você não pode confiar em ninguém na World Wide Web. Tal pensamento poderia parecer cínico, mas, na prática, o que isso significa é;

a segurança da linguagem Java presume que qualquer um pode tentar escrever applets danosos; portanto, ela impede a tentativa de qualquer coisa similar. Como resultado, qualquer recurso da linguagem que seja potencialmente problemático foi bloqueado para uso em applets. Dentre os recursos proibidos estão os seguintes:

- Ler arquivos do sistema em que o applet está sendo executado.
- Escrever arquivos no sistema em que o applet está sendo executado.
- Obter informações sobre um arquivo no sistema.
- Excluir um arquivo do sistema.
- Estabelecer uma conexão de rede com qualquer máquina que não seja aquela que divulgou a página da Web que contém o applet.
- Exibir uma janela que não inclua o aviso padrão "Java applet window".

A linguagem Java 2 torna possível que os applets façam tudo o que um aplicativo Java pode fazer — mas apenas se forem provenientes de um provedor de applets confiável e tiverem assinatura digital para verificar sua autenticidade. Uma assinatura digital é um arquivo (ou arquivos) criptografado que acompanha um programa, indicando exatamente de quem o arquivo (ou arquivos) veio. O documento que representa essa assinatura digital é chamado de *certificado.*

Para estabelecer confiança, um provedor de applets deve verificar sua identidade usando um grupo chamado *autoridade de certificado.* De preferência, esses grupos não são afiliados do provedor de applets e devem ter uma reputação estabelecida como empresa confiável. Atualmente, as seguintes empresas estão oferecendo serviços de autenticação de certificado de alguma forma:

- VeriSign — A primeira e mais amplamente autoridade de certificado estabelecida, oferecendo autorização específica da Microsoft e da Netscape, http://www. verisign.com.
- Thawte Certification — Uma autoridade mais recente para certificados da Microsoft, Netscape e teste. http://www.thawte.cotn.

Outras empresas oferecem certificação para clientes em áreas geográficas específicas. A Netscape lista as autoridades de certificado com quem ela trabalha, no seguinte endereço da Web:

http://certs.netscape.com/client.html

Os usuários, conhecendo quem produziu um programa, podem decidir se esse grupo ou indivíduo deve merecer confiança. As pessoas que estão familiarizadas com os controles ActiveX reconhecerão esse sistema — ele é semelhante a como os programas ActiveX são disponibilizados em páginas da World Wide Web.

Nota

O modelo de segurança geral descrito aqui é o oficial, criado pela JavaSoft para uso em seu próprio navegador HotJava e em todos os navegadores que oferecem suporte integral a Java 2. A Netscape e a Microsoft introduziram seus próprios modelos de segurança em seus navegadores; portanto, quando este livro estava sendo redigido, um applet devia implementar diferentes sistemas para cada navegador em que devesse ser executado. Felizmente, os sistemas são similares, de modo que o domínio de um torna muito mais fácil aprender os outros.

Você também pode estabelecer níveis de segurança diferentes da confiança completa (um applet pode fazer qualquer coisa) ou nenhuma confiança (um applet não pode fazer nada que possa ser danoso). A linguagem Java 2 possibilita isso com um conjunto de classes chamadas *permissões*.

Por enquanto, todos os applets serão totalmente restritos, a não ser que o projetista execute as etapas necessárias para fazer a assinatura digital no applet e que um usuário passe pelo processo de estabelecer se o projetista é confiável.

Um exemplo de assinatura digital

Talvez você ache o entendimento do processo de confiança no applet mais fácil, se usar estas três entidades fictícias: um desenvolvedor de applets chamado Fishhead Software, um grupo industrial Java chamado Signatures 'R' Us e um usuário da Web chamado Gilbert.

A Fishhead Software oferece um applet de jogo em seu site da Web, que salva os escores mais altos e outras informações na unidade de disco rígido do usuário. Esse recurso normalmente não é possível com um applet — o acesso a disco é uma restrição definitiva. Para que o jogo possa ser executado, a Fishhead deve fazer uma assinatura digital no applet e permitir que os usuários estabeleçam a Fishhead como um programador confiável.

Esse processo tem cinco fases:

1. Fishhead Software usa keytool, uma ferramenta que acompanha o JDK, para criar dois arquivos criptografados, chamados de chave pública e chave privativa. Juntas, essas chaves representam um cartão de identificação eletrônica que identifica a empresa. A Fishhead garante que sua chave privativa não aparece para mais ninguém. Ela pode — e deve — tornar sua chave pública disponível para qualquer um, como uma forma de identificação parcial.
2. Fishhead Software precisa de uma entidade que possa verificar quem ela é. Ela envia sua chave pública e um arquivo descritivo a seu respeito para um grupo independente, em que esses usuários de Java provavelmente confiam — a Signatures 'R' Us.
3. Signatures 'R' Us verifica a Fishhead Software para ver se ele é um grupo legítimo, com a mesma chave pública que foi enviada para a

Signatures 'R' Us. Quando a Fishhead satisfizer os requisitos, a Signatures 'R' Us criará um novo arquivo criptografado chamado certificado. Ele é enviado de volta para a Fishhead.

4. Fishhead cria um repositório de arquivos Java que contém seu applet de jogo e todos os arquivos relacionados. Com uma chave pública uma chave privativa e um certificado, a Fishhead Software pode agora usar a ferramenta `jar` para fazer a assinatura digital do arquivo repositório.
5. Fishhead coloca o repositório de arquivos assinado no site da Web, junto com um modo de se carregar por download sua chave pública.

Seguir esse processo é tudo o que a Fishhead Software precisa para tornar o applet disponível para quem confiar o suficiente na empresa, para executá-lo através da Web. Uma das pessoas que decide confiar na Fishhead é um usuário da Web chamado Gilbert, que possui um navegador compatível com Java 2.

Seu processo é mais simples:

1. Gilbert percebe que não pode executar o novo applet de jogo da Fishhead sem estabelecer a empresa como um programador confiável. Ele carrega por download a chave pública da Fishhead.
2. Decidindo que a Fishhead é uma organização em que pode confiar, Gilbert usa outra ferramenta de segurança do JDK, `jarsigner`, em conjunto com a chave pública da Fishhead, para incluir a empresa na lista de programadores confiáveis de seu sistema.

Agora, Gilbert pode brincar à vontade com o applet de jogo da Fishhead. Dependendo de como as permissões de segurança estão estabelecidas dentro do applet, ele poderia ler e escrever arquivos e abrir outras conexões de rede, assim como fazer outras coisas inseguras. Isso significa que código malicioso ou não intencionalmente danoso pode ser executado no sistema de Gilbert, mas isso também é verdade para qualquer software que ele instale e execute em seu computador. A vantagem de uma assinatura digital é que os programadores estão claramente identificados. Pergunte a si mesmo quantos criadores de vírus distribuiriam seu trabalho, sob qualquer tipo de sistema, que deixariam um rastro digital levando diretamente as suas casas.

Um aspecto do novo modelo de segurança Java que pode não estar muito claro é por que você tem uma chave pública e uma chave privativa. Se elas podem ser usadas em conjunto para identificar alguém, como é que a chave pública sozinha pode ser utilizada como uma identificação da Fishhead?

Uma chave pública e uma chave privativa representam um conjunto combinado. Como elas identificam completamente a Fishhead Software, essa entidade é a única que tem acesso a ambas as chaves. Caso contrário, alguém poderia se fazer passar pela Fishhead e ninguém mais poderia saber que se

tratava de uma farsa. Se a Fishhead proteger sua chave privativa, ela protegerá sua identidade e sua reputação.

Quando a Signatures 'R' Us utiliza uma chave pública para verificar a identidade da Fishhead, sua principal função é garantir que a chave pública realmente pertence à empresa. Como as chaves públicas podem ser dadas a qualquer um, a Fishhead pode tornar a sua disponível em seu site da Web. Como parte de seu processo de certificação, a Signatures 'R' Us poderia carregar por download essa chave pública e compará-la com a que recebeu. O grupo que faz a certificação atua como uma espécie de substituto da chave privativa, verificando se a chave pública é legítima. O certificado emitido é vinculado à chave pública, que só pode ser usada com a chave privativa da Fishhead.

Qualquer um pode emitir um certificado para uma chave pública, usando o programa keytool — a Fishhead Software poderia até certificar a si mesma. No entanto, fazer isso tornaria muito mais difícil para os usuários confiarem na empresa do que se fosse utilizado um grupo de certificação independente e bem estabelecido.

Trabalhando em conjunto, a chave pública, a chave privativa e o certificado podem criar uma assinatura digital confiável para um repositório de arquivos Java. A documentação da Sun para keytool, jarsigner, permissões e outros novos recursos de segurança, estão disponíveis no seguinte endereço da Web:

```
http://www.sun.com
```

Assinaturas específicas do navegador

Quando este livro estava sendo redigido, o único modo de fazer uma assinatura digital em um applet era usando os procedimentos estabelecidos pelos projetistas da Netscape e da Microsoft para seus próprios navegadores da Web. Você tinha de usar as ferramentas deles próprios e assinar um applet usando os dois procedimentos, se quisesse atingir os usuários dos dois navegadores.

Assinar um applet para uso no Microsoft Internet Explorer exige o seguinte:

- Uma identificação digital Microsoft Authenticode de uma empresa como a VeriSign ou a Thawte, as quais verificam sua identidade.
- Internet Explorer 4.0 ou superior.
- As seguintes ferramentas do Microsoft Java Software Development Kit: cabarc.exe, chktrust.exe, signcode.exe e os arquivos .DLL javasign.dl 1 e signer.dll. Esse kit está disponível para carregamento por download na Microsoft, no endereço http://www.microsoft.com/java/download.htm.

Assinar um applet para navegadores Netscape Navigator exige o seguinte:

- Uma identificação digital de publicação de software Netscape Object Signing, que pode ser adquirida de uma das empresas listadas na página da Web com o endereço http://certs.netscape.com/client.html.
- O Netscape Signing Tool, que está disponível na página da Web com o endereço http://developer.netscape.com/software/signeclobj/ j arpack. html. O Signing Tool tem um recurso para usar um certificado de teste antes de você ter adquirido uma identificação digital.

Nota

A documentação para 0 uso dessas ferramentas está disponível nos locais na Microsoft e na Netscape em que foram carregados por download na Web. Além disso, Daniel Griscom, da Suitable System, compilou um recurso de assinatura de código Java excelente, no seguinte endereço da Web:

http://www.suitable.com/Doc CodeSigning.shtml

Planos de segurança

Antes da linguagem Java 2, havia a suposição implícita de que todos os aplicativos deveriam ser completamente confiáveis e que poderiam utilizar todos os recursos da linguagem.

Para tornar mais fácil criar aplicativos mais limitados, agora eles são mantidos sob o mesmo regime de segurança que os applets.

Na prática, isso não mudará o modo como os aplicativos são escritos ou executados — aqueles que você criou neste livro não devem ter encontrado nenhuma exceção de segurança ao serem executados em seu sistema. Isso ocorre porque o plano de segurança estabelecido durante a instalação do JDK é o mais liberal possível, permitindo que todos os recursos estejam disponíveis para os aplicativos.

O plano de segurança é armazenado em um arquivo chamado java. policy. Esse arquivo pode ser encontrado na subpasta 1 i b\securi ty\ da pasta de instalação principal do JDK. Esse arquivo pode ser editado com qualquer editor de textos; contudo, você não deve alterá-lo, a não ser que saiba muito bem como ele está estabelecido. Você também pode usar uma ferramenta gráfica de edição de plano incluída com o JDK, chamada pol i cytool.

Uma visão geral dos recursos de segurança implementados na linguagem Java 2 está disponível na Sun, na seguinte página da Web:

http://java.sun.com/product5/jdk/1.2/docs/guide/security/spec/security-spec.doc .html

Resumo

Hoje, você aprendeu a respeito de como as exceções auxiliam no design e robustez de seu programa. As exceções proporcionam um modo de gerenciar erros em potencial em seus programas e de alertar aos usuários de seus programas de que erros podem ocorrer. Usando try, catch e final1y, você pode proteger código que pode resultar em exceções; capturar e tratar dessas exceções, se elas ocorrerem; e executar código, se uma exceção foi gerada ou não.

O tratamento de exceções é apenas metade da equação; a outra metade é a geração e o levantamento das exceções por sua conta. Hoje, você aprendeu a respeito da cláusula throws, que diz aos usuários de seus métodos que o método pode levantar uma exceção. throws também pode ser usada para passar uma exceção de uma chamada de método no corpo de seu método.

Além das informações fornecidas pela cláusula throws, você aprendeu a criar e levantar seus próprios métodos, definindo novas classes de exceção e levantando instâncias de quaisquer classes de exceção usando throw.

Você também aprendeu os fundamentos da implementação do modelo de segurança da linguagem Java 2 e como os diferentes desenvolvedores de navegador estão oferecendo um modo de anular a segurança normal dos applets com assinaturas digitais.

Perguntas e respostas

P **Ainda não tenho certeza de que entendo as diferenças entre exceções, erros e exceções de runtime. Há outro modo de considerá-los?**

R Os erros são causados pela vinculação dinâmica ou por problemas de máquina virtual e, portanto, estão em um nível muito baixo para que a maioria dos programas se preocupem com eles — ou podem ser tratados, mesmo que não se preocupem com isso. As exceções de runtime são geradas pela execução normal do código Java e, embora reflitam ocasionalmente uma condição de que você desejará tratar explicitamente, com mais freqüência refletem um engano de codificação cometido pelo programador e, assim, precisam simplesmente imprimir um erro para ajudar a sinalizar esse engano. As exceções que não são de runtime (exceções IOException, por exemplo) são condições que, devido à sua natureza, devem ser tratadas explicitamente por qualquer código robusto e bem feito. A biblioteca de classe Java foi escrita usando-se apenas algumas delas, mas essas poucas são extremamente importantes para se usar o sistema segura e corretamente. O compilador o ajuda a tratar dessas

exceções corretamente, com suas verificações da cláusula throws e das restrições.

P **Há um modo de evitar as restrições rigorosas impostas aos métodos pela cláusula throws?**

R Sim. Suponha que pensou bastante e decidiu que precisa circundar essa restrição. Quase nunca isso acontece, pois a solução correta é voltar e projetar novamente seus métodos para que reflitam as exceções que você precisa levantar. Imagine, contudo, que por algum motivo uma classe de sistema o coloque em uma camisa de força. Sua primeira solução é transformar RuntimeException em uma subclasse para constituir sua própria nova e isenta exceção. Agora, você pode levantá-la como quiser, pois a cláusula throws que o estava aborrecendo não precisa incluir essa nova exceção. Se você precisar de muitas exceções dessas, uma estratégia elegante é misturar algumas interfaces de exceção singulares em suas novas classes Runtime. Você está livre para escolher que subconjunto dessas novas interfaces deseja capturar (nenhuma das exceções normais de Runtime precisa ser capturada), enquanto todas as exceções restantes de Runtime podem passar para esse método padrão (de outro modo, aborrecido) na biblioteca.

SEMANA 3

Dia 17

Tratamento de dados através de fluxos Java

Muitos dos programas que você criar com a linguagem Java precisarão interagir com algum tipo de origem de dados. Existem inúmeras maneiras pelas quais as informações podem ser armazenadas em um computador, incluindo arquivos em uma unidade de disco rígido ou CD-ROM, páginas em um site da Web e até a própria memória do computador.

Você poderia esperar que houvesse diferentes técnicas para tratar cada um dos diferentes dispositivos de armazenamento. Felizmente, esse não é o caso.

Em Java, as informações podem ser armazenadas e recuperadas usando-se um sistema de comunicação denominado fluxos, que são implementados no pacote `java.io`.

Hoje, você aprenderá a criar fluxos de entrada para ler informações e fluxos de saída para armazená-las. Você trabalhará com o seguinte:

- Fluxos de byte, que são usados para tratar bytes, inteiros e outros tipos de dados simples
- Fluxos de caractere, que tratam arquivos de texto e outras fontes de texto

Você pode tratar todos os dados da mesma maneira, quando souber como trabalhar com um fluxo de entrada, seja ele proveniente de um disco, da Internet ou mesmo outro programa. O inverso é verdadeiro para fluxos de saída.

Os fluxos representam um mecanismo poderoso para tratar dados, mas você não paga por esse poder com classes difíceis de implementar

Introdução aos fluxos

Todos os dados em Java são escritos e lidos usando-se fluxos. Os fluxos, assim como o volume d'água que compartilha o mesmo nome, carregam algo de um lado para outro.

NOVO TERMO Um *fluxo* é o caminho percorrido pelos dados em um programa. Um *fluxo de entrada* envia dados de uma origem para um programa e um *fluxo de saída* envia dados de um programa para um destino.

Hoje você conhecerá dois tipos diferentes de fluxos: fluxos de byte e fluxos de caractere. Os *bytes* transportam inteiros com valores que variam de 0 a 255. Uma grande variedade de dados pode ser expressa em formato de byte, incluindo dados numéricos, programas executáveis, comunicação na Internet e bytecode — os arquivos de classe que são executados por uma máquina virtual Java.

Na verdade, qualquer tipo de dados imaginável pode ser expresso usando-se bytes individuais ou uma série de bytes combinados entre si.

NOVO TERMO Os *fluxos de caractere* são um tipo especializado de fluxo de byte que trata apenas dados textuais. Eles são diferentes dos fluxos de byte porque o conjunto de caracteres da linguagem Java oferece suporte a Unicode, um padrão que inclui muito mais caracteres do que poderia ser expresso facilmente usando-se bytes.

Qualquer tipo de dado que envolva texto deve usar fluxos de caractere, incluindo arquivos de texto, páginas da Web e outros tipos comuns de texto.

Usando um fluxo

Esteja você usando um fluxo de byte ou um fluxo de caractere, o procedimento para utilizar um ou outro na linguagem Java é basicamente o mesmo. Antes de começar a trabalhar com os detalhes específicos das classes java. i o, é interessante passar pelo processo de criação e uso de fluxos.

No caso de um fluxo de entrada, o primeiro passo é criar um objeto que esteja associado à origem dos dados. Por exemplo, se a origem é um arquivo em sua unidade de disco rígido, um objeto FileInputStream poderia ser associado a esse arquivo.

Uma vez que esteja de posse de um objeto de fluxo, você pode ler informações desse fluxo usando um dos métodos do objeto. FilelnputStream inclui um método read ( ) que retorna um byte lido do arquivo.

Quando acabar de ler as informações do fluxo, você chama o método close( ) para indicar que acabou de usar o fluxo.

No caso de um fluxo de saída, você começa criando um objeto associado ao destino dos dados. Um objeto assim pode ser criado a partir da classe BufferedReader, que representa um modo eficiente de criar arquivos de texto.

O método write( ) é o modo mais simples de enviar informações para o destino do fluxo de saída. Por exemplo, um método BufferedReader write( ) pode enviar caracteres específicos para um fluxo de saída.

Assim como se faz com os fluxos de entrada, o método close( ) é chamado em um fluxo de saída, quando você não possui mais informações para enviar.

Filtrando um fluxo

O modo mais simples de usar um fluxo é criá-lo e depois chamar seus métodos para enviar ou receber dados, dependendo de se tratar um fluxo de saída ou de um fluxo de entrada.

Muitas das classes com que você vai trabalhar hoje conseguem resultados mais sofisticados associando um filtro a um fluxo, antes de ler ou escrever quaisquer dados.

NOVO TERMO

Um *filtro* é um tipo de fluxo que modifica o modo como um fluxo existente é tratado. Pense na estrutura de uma barragem que gera uma represa, a qual regula o fluxo de água da parte mais alta para a parte mais baixa. Trata-se de uma espécie de filtro — se for removido, a água fluirá de forma menos controlada.

O procedimento para se usar um filtro em um fluxo é basicamente o seguinte:

- Crie um fluxo associado a uma origem ou a um destino de dados.
- Associe um filtro a esse fluxo.
- Leia ou escreva dados do filtro, em vez de do fluxo original.

Os métodos que você chama em um filtro são os mesmos que chamaria em um fluxo: existem métodos read( ) ou write( ), exatamente como haveria em um fluxo não filtrado.

Você pode até associar um filtro a outro, de modo que o seguinte caminho para as informações é possível: um fluxo de entrada associado a um arquivo de texto, é passado por um filtro de tradução de espanhol para inglês, que é então passado por um filtro antiblasfêmia, sendo finalmente enviado para seu destino — um ser humano que deseja lê-lo.

Se isso ainda estiver confuso no plano abstrato, você terá muitas oportunidades de ver na prática, nas seções a seguir.

Fluxos de byte

Todos os fluxos de byte são uma subclasse de InputStream ou OutputStream. Essas classes são abstratas; portanto, você não pode criar um fluxo gerando objetos dessas classes. Em vez disso, você cria fluxos por uma de suas subclasses, como os seguintes:

- FileInputStream e FileOutputStream Fluxos de byte armazenados em arquivos no disco, CD-ROM ou outros dispositivos de armazenamento.
- Data lnputStream e Data OutputStream Um fluxo de byte filtrado a partir do qual podem ser lidos dados, como inteiros e números em ponto flutuante.

InputStream é a superclasse de todos os fluxos de entrada.

Fluxos de arquivo

Os fluxos de byte com que você provavelmente mais trabalha são fluxos de arquivo, que são utilizados para trocar dados com arquivos em suas unidades de disco, CD-ROMs ou outros dispositivos de armazenamento a que você pode fazer referência usando um caminho de pasta e um nome de arquivo.

Você pode enviar bytes para um fluxo de saída de arquivo e receber bytes de um fluxo de entrada de arquivo.

Fluxos de entrada de arquivo

Um fluxo de entrada de arquivo pode ser criado com o construtor FileInput-Stream(síring). O argumento string deve ser o nome do arquivo. Você pode incluir uma referência de caminho com o nome de arquivo, o que permite que o arquivo esteja em uma pasta diferente da classe que o está carregando. A instrução a seguir cria um fluxo de entrada de arquivo a partir do arquivo scores.dat:

```
FileInputStream fis = new FileInputStream("scores.dat");
```

Após criar um fluxo de entrada de arquivo, você pode ler bytes do fluxo chamando seu método read( ). Esse método retorna um inteiro contendo o próximo byte do fluxo. Se o método retornar -1, que não é um valor de byte possível, isso significa que o final do fluxo de arquivo foi atingido.

Para ler mais de um byte de dados do fluxo, chame seu método read *[byte[ ], int, int).* Os argumentos desse método são os seguintes:

- Um array de bytes onde os dados serão armazenados.
- O elemento dentro do array onde o primeiro byte dos dados deve estar armazenado.
- O número de bytes a serem lidos.

Diferente do outro método read ( ), este não retorna dados do fluxo. Em vez disso, ele retorna um inteiro que representa o número de bytes lidos ou -1, se nenhum byte foi lido antes que o final do fluxo tenha sido atingido.

As instruções a seguir usam um loop whi 1 e para ler os dados em um objeto FileInputStream chamado df:

```
int newByte = 0;
while (newByte != -1) {
    newByte = df.read( );
    System.out.print(newByte + " " ) ;
}
```

Esse loop lê o arquivo inteiro a que df faz referência, um byte por vez, e apresenta cada byte seguido de um caractere de espaço. Ele também apresentará o valor -1 quando o final do arquivo tiver sido atingido — você poderia prevenir-se contra isso facilmente, com uma instrução i f.

O aplicativo ReadBytes, na listagem *17.1,* usa uma técnica semelhante para ler um fluxo de entrada de arquivo. O método close( ) do fluxo de entrada é utilizado para fechar o fluxo depois que o último byte do arquivo for lido. Isso deve ser feito para liberar recursos do sistema associados ao arquivo aberto.

DIGITE **Listagem 17.1.** O texto completo de readbytes.java.

```
1: import java.io.*;
2:
3: public class ReadBytes {
4:     public static void main(String[ ] arguments) {
5:         try {
6:             FileInputStream file = new
7:                 FileInputStream("class.dat");
8:             boolean eof = false;
9:             int count = 0;
10:            while (!eof) {
11:                int input = file.read( );
12:                System.out.print(input + " ");
13:                if (input =  -1)
14:                    eof = true;
15:                else
16:                    count++;
17:            }
18:        file.close(       );
19:            System.out.println("\nBytes read: " + count);
20:        } catch (IOException e) {
21:            System.out.println("Error - " + e.toString( ));
22:        }
23:    }
24: }
```

Se você executar esse programa, obterá a seguinte mensagem de erro:

SAÍDA

```
Error      java.io.FileNotFoundException: class.dat
```

Essa mensagem de erro é parecida com os tipos de exceções geradas pelo compilador, mas, na verdade, ela é proveniente do bloco catch, nas linhas 21a 23 do aplicativo ReadBytes. A exceção está sendo levantada pelas linhas 6 e 7, pois o arquivo class. dat não pode ser encontrado.

Você precisa de um arquivo de bytes para ler. Pode ser qualquer arquivo — uma escolha conveniente é o arquivo de classe do programa, que contém as instruções em código de byte executadas pela máquina virtual Java. Crie esse arquivo fazendo uma cópia de ReadBytes.class e atribuindo-lhe o nome class.dat. Não atribua um novo nome a ReadBytes.class em si ou você não poderá executar o programa.

Os usuários de Windows 95 e Windows NT podem usar o prompt do MS-DOS para criar class. dat. Vá para a pasta que contém ReadBytes . class e use o seguinte comando DOS:

```
copy  ReadBytes.class  class.dat
```

Os usuários de UNIX podem digitar o seguinte na linha de comando:

```
cp  ReadBytes.class  class.dat
```

Quando você executar o programa, cada byte presente em class.dat será apresentado, seguido de uma contagem do número total de bytes. Se você utilizou ReadBytes .class para criar class. dat, as últimas linhas da saída deverão ser semelhantes às seguintes:

SAÍDA

```
49 43 182 0 23 54 4 178 0 20 187 0 11 89 21 4 184 0 26 183 0 16 18
   2 182 0 18 182 0 24 182 0 21 21 4 2 160 0 8 4 61 167 0 6 132 3 1
28 153 255 209 43 182 0 19 178 0 20 187 0 11 89 18 1 183 0 16 29 1
82 0 17 182 0 24 182 0 22 167 0 29 76 178 0 20 187 0 11 89 18 3 18
3 0 16 43 182 0 25 182 0 18 182 0 24 182 0 22 177 0 1 0 0 0 93 0 9
6 0 7 0 1 0 53 0 0 0 90 0 22 0 0 0 5 0 0 0 6 0 4 0 7 0 6 0 6 0 10
0 8 0 12 0 9 0 14 0 10 0 17 0 11 0 23 0 12 0 49 0 13 0 55 0 14 0 5
7 0 13 0 60 0 16 0 63 0 10 0 67 0 18 0 71 0 19 0 83 0 20 0 90 0 19
   0 93 0 5 0 96 0 21 0 97 0 22 0 122 0 4 0 1 0 58 0 0 0 2 0 57 -1
Bytes read: 1121
```

O número de bytes exibidos em cada linha da saída depende da largura da coluna que o texto pode ocupar em seu sistema. Os bytes exibidos dependem do arquivo utilizado para criar class.dat.

Fluxos de saída de arquivo

Um fluxo de saída de arquivo pode ser criado com o construtor FileOutput-*Stream(String).* A utilização é a mesma do construtor FileInput-*Stream{String);* portanto, você pode especificar um caminho junto com um nome de arquivo.

Você tem de tomar cuidado quando especificar o arquivo em que vai escrever um fluxo de saída. Se ele for o mesmo que um arquivo existente, o original será apagado quando você começar a escrever dados no fluxo.

Você pode criar um fluxo de saída de arquivo que anexa dados após o final de um arquivo existente, com o construtor Fi leOutputStream(StriA)g, *boolean).* O string especifica o arquivo e o argumento booleano deve ser igual a true para anexar dados, em vez de sobrescrever os dados já existentes.

O método write(int) do fluxo de saída de arquivo é usado para escrever bytes no fluxo. Depois que o último byte foi escrito no arquivo, o método close( ) do fluxo o fecha.

Para se escrever mais de um byte, o método write (byte*[ ], int, int)* pode ser usado. Isso funciona de maneira semelhante ao método read (*byte[ ], int, int)* descrito anteriormente. Os argumentos desse método são o array de bytes que contém os bytes a serem apresentados na saída, o ponto inicial no array e o número de bytes a serem escritos.

O aplicativo WriteBytes, na listagem 17.2, escreve um array de inteiros em um fluxo de saída de arquivo.

DIGITE **Listagem 17.2.** O texto completo de writebytes.java.

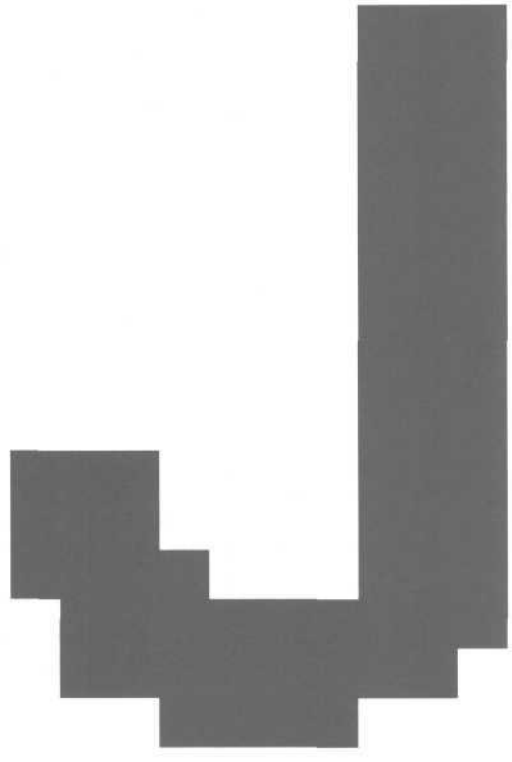
```
1: import java.io.*;
2:
3: public class WriteBytes {
4:     public static void main(String[ ] arguments) {
5:         int[ ] data = { 71, 73, 70, 56, 57, 97, 15, 0, 15, 0,
6:             128, 0, 0, 255, 255, 255, 0, 0, 0, 44, 0, 0, 0,
7:             0, 15, 0, 15, 0, 0, 2, 33, 132, 127, 161, 200,
8:             185, 205, 84, 128, 241, 81, 35, 175, 155, 26,
9:             228, 254, 105, 33, 102, 121, 165, 201, 145, 169,
10:            154, 142, 172, 116, 162, 240, 90, 197, 5, 0, 59 };
11:        try {
12:            FileOutputStream file = new
13:            File0utputStream("pi          c.gif");
14:            for (int i = 0; i < data.length; i++)
15:                file.write(data[i]);
16:            file.close( );
17:        } catch (IOException e) {
18:            System.out.println("Error - " + e.toString( ));
19:        }
20:    }
21: }
```

O seguinte está acontecendo nesse programa:

- Linhas 5 a 10 Um array de inteiros, chamado data, é criado com 66 elementos.
- Linhas 12 e 13 Um fluxo de saída de arquivo é criado com o nome de arquivo pic.gif, na mesma pasta que o arquivo WriteBytes.class.
- Linhas 14 e 15 Um loop for é usado para circular pelo array data e escrever cada elemento no fluxo de arquivo.
- Linha 16 O fluxo de saída de arquivo é fechado.

Após executar esse programa, você pode apresentar o arquivo pic.gif em qualquer navegador da Web ou ferramenta de edição gráfica. Trata-se de um pequeno arquivo de imagem no formato GIF, como se vê na Figura 17.1.

Figura 17.1
O arquivo pic.gif (ampliado).



Filtrando um fluxo

NOVO TERMO *Fluxos filtrados* são aqueles que modificam as informações enviadas por um fluxo existente. Eles são criados usando-se as subclasses FilterInputStream ou FilterOutputStream.

Essas classes não tratam de nenhuma operação de filtragem sozinhas. Em vez disso, elas possuem subclasses, como BufferlnputStream e DataOutput-Stream, que tratam tipos específicos de filtragem.

Filtros de byte

As informações são distribuídas mais rapidamente se elas puderem ser enviadas em trechos maiores, mesmo que esses trechos sejam recebidos mais rapidamente do que possam ser tratados.

Como um exemplo disso, considere qual das seguintes técnicas de leitura de livro é mais rápida:

- Um amigo lhe empresta um livro em sua totalidade e você o lê.
- Um amigo lhe empresta um livro uma página por vez e não lhe dá uma página nova enquanto você não terminar a anterior.

Obviamente, a primeira técnica será mais rápida e eficiente. As mesmas vantagens valem para os fluxos colocados em buffer na linguagem Java.

NOVO TERMO Um *buffer* é um local de armazenamento em que os dados podem ser mantidos antes de serem necessários em um programa que lê ou escreve esses dados. Usando um butter, você pode obter dados sem estar sempre voltando à fonte original dos dados.

Fluxos em buffer

Um fluxo de entrada colocado em buffer preenche esse buffer com dados que ainda não foram tratados e quando um programa precisar desses dados, ele olhará primeiro no buffer, antes de ir até a fonte original do fluxo. Isso é muito mais eficiente — usar um fluxo sem um buffer é semelhante a receber um livro uma página por vez. Todos os atrasos desse fluxo irão retardar os esforços de usá-lo.

Os fluxos de byte colocados em buffer usam as classes BufferedlnputStream e BufferedOutputStream.

Um fluxo de entrada colocado em buffer é criado usando-se um dos dois construtores a seguir:

- BufferedInputStream(inpuíStream) Cria um fluxo de entrada colocado em buffer para o objeto *InputStream* especificado.
- BufferedInputStream(InputStream, *int)* Cria o fluxo colocado em buffer *InputStream* especificado, com um buffer de tamanho *int.*

O modo mais simples de ler dados de um fluxo de entrada colocado em buffer é chamando seu método read( ) sem nenhum argumento, o que normalmente retorna um inteiro de 0 a 255 representando o próximo byte do fluxo. Se o final do fluxo for atingido e nenhum byte estiver disponível, -1 será retornado.

Você também pode usar o método read*[byte[ ], int, int)* disponível para outros fluxos de entrada, que carrega os dados do fluxo em um array de bytes.

Um fluxo de saída colocado em buffer é criado usando-se um dos dois seguintes construtores:

- BufferedOutputStream(0utpi/tS*tream)* Cria um fluxo de saída colocado em buffer para o objeto *OutputStream* especificado.
- BufferedOutputStream(OutputStream, *int)* Cria o fluxo colocado em buffer *OutputStream* especificado, com um buffer de tamanho *int*.

O método write (int) do fluxo de saída pode ser usado para enviar um byte para o fluxo e o método write*[byte[ ], int, int)* escreve vários bytes do array de bytes especificado. Os argumentos desse método são o array de bytes, o ponto de partida do array e o número de bytes a serem escritos.

Embora o método write( ) receba um inteiro como entrada, o valor deve estar entre 0 e 255. Se você especificar um número maior do que 255, ele será armazenado como o resto do número dividido por 256. Você pode testar isso quando executar o projeto criado posteriormente nesta seção.

Quando os dados forem direcionados para um fluxo colocado em buffer, eles não serão produzidos na saída de seu destino até que o fluxo seja preenchido ou que o método flush( ) do fluxo colocado em buffer seja chamado.

O próximo projeto, o aplicativo BufferDemo, escreve uma série de bytes em um fluxo de saída colocado em buffer, associados a um arquivo de texto. O primeiro e o último inteiro da série são especificados como dois argumentos de linha de comando, como na instrução a seguir:

```
java BufferDemo 7 64
```

Após escrever no arquivo de texto, BufferDemo cria um fluxo de entrada colocado em buffer a partir do arquivo e lê os bytes de volta. A listagem 17.3 contém o código-fonte.

Listagem 17.3. O texto completo de bufferdemo.java.

```
1: import java.io.*;
2:
3: public class BufferDemo {
4:     public static void main(String[ ] arguments) {
5:         int start = 0;
6:         int finish = 255;
7:         if (arguments.length > 1) {
8:             start = Integer.parseInt(arguments[0]);
```

Listagem 17.3. Continuação

```
9:            finish = Integer.parseInt(arguments[l]);
10:       } else if (arguments.length > 0)
11:           start = Integer.parseInt(arguments[O]);
12:       ArgStream as = new ArgStream(start, finish);
13:       System.out.println("\nWriting: " ) ;
14:       boolean success = as.writeStream( );
15:       System.out.println("\nReading: " ) ;
16:       boolean readSuccess = as.readStream( );
17:   }
18: }
19:
20: class ArgStream {
21:     int start = 0;
22:     int finish = 255;
23:
24:     ArgStream(int st, int fin) {
25:         start = st;
26:         finish = fin;
27:     }
28:
29:     boolean writeStream( ) {
30:        try {
31:             FileOutputStream file = new
32:                 Fi1eOutputStream("numbers.dat") ;
33:             BufferedOutputStream buff • new
34:                 BufferedOutputStream(file);
35:            for (int out = start; out <= finish; out++) {
36:                buff.write(out);
37:               System.out.print(" " + out);
38:            }
39:            buff.close( );
40:            return true;
41:        } catch (IOException e) {
42:           System.out.println("Exception: " + e.getMessage( ));
43:            return false;
44:        }
45:     }
46:
47:     boolean readStream( ) {
48:        try {
49:             FilelnputStream file = new
50:                 FileInputStream("numbers.dat");
51:             BufferedlnputStream buff = new
52:                 BufferedInputStream(file);
53:          int in = 0;
54:           do {
55:               in = buff.read( );
56:            if (in != -1)
```

Listagem 17.3. Continuação

```
57:                 System.out.print(" " + in);
58:             } while (in != -1);
59:              buff.close( );
60:              return true;
61:         } catch (IOException e) {
62:             System.out.println("Exception: " + e.getMessage( ))s
63:              return false;
64:         }
65:     }
66: }
```

A saída desse programa depende dos dois argumentos especificados na linha de comando. Se você usar java BufferDemo 4 13, a seguinte saída será apresentada:

SAÍDA

```
Writing:
 4 5 6 7 8 9 10 11 12 13
Reading:
 4 5 6 7 8 9 10 11 12 13
```

Esse aplicativo é composto de duas classes: BufferDemo e uma classe auxiliar chamada ArgStream. BufferDemo recebe os valores dos dois argumentos, caso eles sejam fornecidos, e os utiliza no construtor ArgStream( ).

O método writeStream( ) de ArgStream é chamado na linha 14 para escrever a série de bytes em um fluxo de saída colocado em buffer, e o método readStream( ) é chamado na linha 16 para ler esses bytes de volta.

Mesmo que estejam movendo dados em duas direções diferentes, os métodos writeStream( ) e readStream( ) são basicamente iguais. Eles assumem o seguinte formato:

- O nome de arquivo, numbers.dat, é usado para criar um fluxo de entrada ou de saída de arquivo.
- O fluxo de arquivo é usado para criar um fluxo de entrada ou de saída colocado em buffer.
- O método write( ) do fluxo colocado em buffer é usado para enviar dados, ou o método read( ) é usado para recebê-los.
- O fluxo colocado em buffer é fechado.

Como os fluxos de arquivo e colocado em buffer levantam objetos IOException, se um erro ocorrer, todas as operações envolvendo os fluxos são englobadas em um bloco try...catch para essa exceção.

Dica

Os valores de retorno booleanos em writeStream( ) e readStream( ) indicam se a operação de fluxo foi concluída com êxito. Eles não são usados nesse programa, mas é considerada uma boa prática permitir que os chamadores desses métodos saibam que algo deu errado.

Fluxos de dados

Se precisar trabalhar com dados que não são representados como bytes ou caracteres, você pode usar fluxos de entrada ou de saída de dados. Esses fluxos filtram um fluxo de byte existente para que cada um dos seguintes tipos primitivos possa ser lido ou escrito diretamente do fluxo: boolean, byte, double, float, int, long e short.

Um fluxo de entrada de dados é criado com o construtor DataInput-*Stream{InputStream).* O argumento deve ser um fluxo de entrada existente, como um fluxo de entrada colocado em buffer ou um fluxo de entrada de arquivo.

Analogamente, um fluxo de saída de dados exige o construtor DataOutputStream*(OutputStream),* que indica o fluxo de saída associado.

A lista a seguir indica os métodos de leitura e escrita que se aplicam aos fluxos de entrada e de saída de dados, respectivamente:

- readBoolean( ), writeBoolean(boolean)
- readByte( ), *writeByte{integer)*
- readDouble( ), writeDouble(double)
- readFloat( ), writeF1oat(float)
- readInt( ), writeInt(int)
- readLong( ), writeLong(long)
- readShort( ), writeShort(int)

Cada um dos métodos de entrada retorna o tipo de dado primitivo indicado pelo nome do método. Por exemplo, o método readFl oat ( ) retorna um valor float.

Também existem métodos readllnsignedByte( ) e readUnsignedShort( ) que lêem valores *byte short* e sem sinal. Esses não são tipos de dados suportados pela linguagem Java; portanto, são retornados como valores int.

Nota

Os bytes sem sinal possuem valores que variam de 0 a 255. Isso difere do tipo de variável byte da linguagem Java, que varia de -128 a 127. De acordo com isso, um valor short sem sinal varia de 0 a 65.535, em vez do intervalo de -32.768 a 32.767 suportado pelo tipo short da linguagem Java.

Nem todos os diferentes métodos de leitura de um fluxo de entrada de dados retornam um valor que pode ser usado como um indicator de que o final do fluxo foi atingido.

Como alternativa, você pode esperar que uma EOFException (exceção de final de arquivo) seja levantada quando um método de leitura atingir o final de um fluxo. O loop que lê os dados pode ser incluído em um bloco try e a instrução catch associada deve tratar apenas objetos EOFException. Você pode chamar close ( ) no fluxo e cuidar de outras tarefas de limpeza dentro do bloco catch.

Isso é demonstrado no próximo projeto. As listagens 17.4 e 17.5 contêm dois programas que usam fluxos de dados. O aplicativo WritePrimes escreve os primeiros 400 números primos como inteiros em um arquivo chamado 400primes .dat. O aplicativo ReadPrimes lê os inteiros desse arquivo e os apresenta.

DIGITE **Listagem 17.4.** O texto completo de writeprimes.java.

```
1: import java.io.*;
2:
3: class WritePrimes {
4:     public static void main(String arguments[ ]) {
5:         int[ ] primes = new int[400];
6:         int numPrimes = 0;
7:         // candidato: o número que pode ser primo
8:         int candidate = 2;
9:         while (numPrimes < 400) {
10:            if (isPrime(candidate)) {
11:                primes[numPrimes] = candidate;
12:                numPrimes++;
13:            }
14:            candidate++;
15:        }
16:
17:        try {
18:            // Grava a saída no disco
19:            FileOutputStream file = new
20:                FileOutputStreamC400primes.dat");
21:            BufferedOutputStream buff = new
22:                BufferedOutputStream(file);
23:            DataOutputStream data • new
24:                DataOutputStream(buff);
25:
26:        for (int i = 0; i < 400; i++)
27:            data.writeInt(primes[i]);
28:            data.close( );
29:        } catch (IOException e) {
30:            System.out.println("Error - " + e.toString( ));
31:        }
32:    }
```

Listagem 17.4. Continuação

```
33:
34:     public static boolean isPrime(int checkNumber) {
35:         double root = Math.sqrt(checkNumber);
36:         for (int i = 2; i <= root; i++) {
37:             if (checkNumber % i == 0)
38:                 return false;
39:         }
40:         return true;
41:     }
42: }
```

DIGITE

Listagem 17.5. O texto completo de readprimes.java.

```
1: import java.io.*;
2:
3: class ReadPrimes {
4:     public static void main(String arguments[ ]) {
5:         try {
6:             FileInputStream file = new
7:                FileInputStream("400primes.dat");
8:             BufferedInputStream buff = new
9:                 BufferedInputStream(file);
10:            DataInputStream data • new
11:                DataInputStream(buff);
12:
13:            try {
14:                while (true) {
15:                    int in = data.readInt( );
16:                    System.out.print(in + " ");
17:                }
18:            } catch (EOFException eof) {
19:                buff.close( );
20:            }
21:        } catch (IOException e) {
22:            System.out.println("Error - " + e.toString( ));
23:        }
24:    }
25: }
```


A maior parte do aplicativo WritePrimes se ocupa da lógica para encontrar os primeiros 400 números primos. Quando você obtém um array de inteiros contendo os primeiros 400 números primos, ele é escrito em um fluxo de saída de dados nas linhas *17 a 31*.

Esse aplicativo é um exemplo do uso de mais de um filtro em um fluxo. O fluxo é desenvolvido em um processo de três etapas:

- Um fluxo de saída de arquivo, associado a um arquivo chamado 400primes.dat é criado.
- Um novo fluxo de saída colocado em buffer é associado ao fluxo de arquivo.
- Um novo fluxo de saída de dados é associado ao fluxo colocado em buffer.

O método writeInt( ) do fluxo de dados é usado para registrar os números primos no arquivo.

O aplicativo ReadPrimes é mais simples, pois ele não precisa fazer nada a respeito de números primos — ele apenas lê inteiros de um arquivo usando um fluxo de entrada de dados.

As linhas 6 a 11 de Read Primes são praticamente idênticas às instruções do aplicativo WritePrimes, exceto que são usadas classes de entrada em vez de classes de saída.

O bloco try...catch que trata objetos EOFException está nas linhas 13 a 20. O trabalho de carregar os dados ocorre dentro do bloco try.

A instrução while(true) cria um loop infinito. Não se trata de um problema — uma EOFException ocorrerá automaticamente, quando o final do fluxo for encontrado em algum ponto, enquanto o fluxo de dados estiver sendo lido. O método readlnt( ), na linha 15, lê os inteiros do fluxo.

As últimas linhas da saída do aplicativo Read Primes devem ser semelhantes às seguintes:

SAÍDA

```
2137 2141 2143 2153 2161 2179 2203 2207 2213 2221 2237 2239 2243 22
51 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333 2339 2341 2347
2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423 2437 2
441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 255
7 2579 2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677
2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741
```

Fluxos de caractere

Quando você sabe como tratar fluxos de byte, possui também os conhecimentos necessários para tratar fluxos de caractere. Esses fluxos são usados para se trabalhar com qualquer texto que seja representado pelo conjunto de caracteres ASCII ou Unicode, um conjunto de caracteres internacional que inclui o ASCII.

Exemplos de arquivos com que você pode trabalhar através de um fluxo de caractere são os arquivos de texto puros, documentos HTML e arquivos-fonte Java.

Todas as classes usadas para ler e escrever esses fluxos são subclasses de Reader e Writer. Elas devem ser usadas para toda entrada de texto, em vez de se tratar diretamente com fluxos de byte.

Nota

As técnicas de tratamento de fluxos de caractere foram bastante aprimoradas após a versão 1.02 da linguagem Java, com o advento das classes Reader e Writer e suas subclasses; elas possibilitam o suporte a caracteres Unicode e um melhor tratamento de texto. Um applet Java compatível com a versão 1.02 pode ler caracteres usando as classes de fluxo de byte descritas anteriormente.

Lendo arquivos de texto

FileReader e a classe principal usada para se ler rluxos de caractere de um arquivo. Essa classe herda de InputStreamReader, que lê um fluxo de byte e converte os bytes em valores inteiros que representam caracteres Unicode.

Um fluxo de entrada de caractere é associado a um arquivo usando-se o construtor FileReader (St*ring)*. O string indica o arquivo e pode conter referências de caminho de pasta, além de um nome de arquivo.

A instrução a seguir cria um novo Fi 1 eReader chamado 1 ook e o associa a um arquivo de texto chamado index. html:


```
FileReader look = new FileReader("index.html");
```

Quando tiver um leitor de arquivos, você poderá chamar os métodos a seguir para ler caracteres do arquivo:

- read ( ) retorna o próximo caractere do fluxo como um inteiro.
- *read(char[ ], int, int)* lê caracteres no array de caracteres especificado, com o ponto de partida indicado e o número de caracteres-lidos.

O segundo método funciona como os métodos semelhantes das classes de fluxo de entrada de byte. Em vez de retornar o próximo caractere, retorna o número de caracteres que foram lidos ou -1, se nenhum caractere foi lido antes que o final do fluxo tivesse sido encontrado.

O método a seguir carrega um arquivo de texto usando o objeto text de FileReader e apresenta seus caracteres:

```
FileReader text = new
    FileReader("readme.txt");
int inByte;
do {
    inByte = text.read( );
    if (inByte != -1)
      System.out.print( (char)inByte );
} while (inByte != -1);
System.out.printlnC");
text.close( );
```

Como o método read( ) de um fluxo de caractere retorna um inteiro, você precisa fazer sua coerção para um caractere, antes de apresentá-lo, armazená-lo em um array ou usá-lo para formar um string. Todo caractere tem

um código numérico que representa sua posição no conjunto de caracteres Unicode. O inteiro lido do fluxo é esse código numérico.

Se você quiser ler uma linha de texto por vez, em lugar de ler um arquivo caractere por caractere, pode usar a classe BufferedReader em conjunto com um FileReader.

A classe BufferedReader lê um fluxo de entrada de caractere e o coloca em um buffer para obter uma maior eficiência. Você já deve ter um objeto Reader de algum tipo para criar uma versão em buffer. Os seguintes construtores podem ser usados para se criar um BufferedReader:

- BufferedReader *(Reader)* Cria um fluxo de caractere colocado em buffer, associado ao objeto Reader especificado, como FileReader.
- BufferedReader *{Reader, int)* Cria um fluxo de caractere colocado em buffer, associado ao Reader especificado e com um buffer de tamanho *int.*

Um fluxo de caractere colocado em buffer pode ser lido usando-se os métodos read( ) e *read(char[ ], int, int)* descritos para FileReader. Você pode ler uma linha de texto usando o método readLi ne ( ).

O método readLine( ) retorna um objeto String contendo a próxima linha de texto do fluxo, não incluindo o caractere (ou caracteres) que representa o final de uma linha. Se o final do fluxo for atingido, o valor do string retornado será igual a nul 1.

Um final de linha é indicado pelo seguinte:

- Um caractere *newline* ('\n')
- Um caractere *carriage return* ('\r')
- Um *carriage return* seguido de um *newline*

O projeto contido na listagem 17.6 é um aplicativo Java que lê seu próprio arquivo-fonte através de um fluxo de caractere colocado em buffer.

DIGITE **Listagem 17.6.** O texto completo de readsource.java.

```
1: import java.io.*;
2:
3: public class ReadSource {
4:     public static void main(String[ ] arguments) {
5:         try {
6:             FileReader file = new
7:                 FileReader("ReadSource.java");
8:             BufferedReader buff = new
9:                 BufferedReader(file);
10:            boolean eof = false;
11:            while (!eof) {
12:                String line = buff.readLine( );
13:                if (line == null)
```

Listagem 17.6. Continuação

```
14:                     eof = true;
15:                 else
16:                     System.out.println(line);
17:             }
18:             buff.close( );
19:             } catch (IOException e) {
20:                 System.out.println("Error - " + e.toString( ));
21:             }
22:     }
23: }
```

Grande parte desse programa é comparável aos projetos criados anteriormente hoje, como ilustrado:

- Linhas 6 e 7 Uma origem de entrada é criada — o objeto FileReader associado ao arquivo ReadSource.java.
- Linhas 8 e9 Um filtro em buffer é associado a essa origem de entrada — buff do objeto BufferedReader.
- Linhas 11 a 17 Um método readLine( ) é usado dentro de um loop whi 1 e para ler o arquivo de texto uma linha por vez. O loop termina quando o método retorna o valor null.

A saída do aplicativo ReadSource é o arquivo de texto ReadSource.java.

Escrevendo arquivos de texto

A classe FileWriter é usada para se escrever um fluxo de caractere em um arquivo. É uma subclasse de OutputStreamWriter, que possui comportamento para converter códigos de caractere Unicode em bytes.

Existem dois construtores FileWriter: FileWri*ter(String)* e FileWri ter *(String, boolean).* O string indica o nome do arquivo para o qual o fluxo de caractere será dirigido, o que pode incluir um caminho de pasta. O argumento booleano opcional deve ser true se o arquivo for anexado a um arquivo de texto já existente. Assim como acontece com outras classes de escrita de fluxo, você deve tomar cuidado para não sobrescrever acidentalmente um arquivo existente, quando não estiver anexando dados.

Existem três métodos de FileWriter que podem ser usados para se escrever dados em um fluxo:

- write*(int)* Escreve um caractere.
- write *(char[ ], int, int)* Escreve caracteres do array de caracteres especificado, com o ponto de partida indicado e o número de caracteres escritos.

- write(Síring, *int, int)* Escreve caracteres do string especificado, com o ponto de partida indicado e o número de caracteres escritos.

O exemplo a seguir escreve um fluxo de caractere em um arquivo usando a classe FileWriter e o método write(ínt):

```
FileWriter letters = new FileWriter("alphabet.txt");
for (int i = 65; i < 91; i++)
    letters.write( (char)i );
letters.close( );
```

O método close( ) é usado para fechar o fluxo depois que todos os caracteres tiverem sido enviados para o arquivo de destino. A seguir está o arquivo alphabet. txt produzido por esse código:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

A classe Buf feredWri ter pode ser usada para escrever um fluxo de caractere colocado em buffer. Os objetos dessa classe são criados com os construtores BufferedWriter(Writer) ou BufferedWriter*[Uriter, int)*. O argumento W*riter* pode ser qualquer uma das classes de fluxo de saída de caractere, como por exemplo FileWriter. O segundo argumento opcional é um inteiro indicando o tamanho do buffer a ser usado.

BufferedWri ter possui os mesmos três métodos de saída de FileWriter: write(int), wri*te(char[ ], int, int)* e *write(String, int, int).*

Outro método de saída útil é newLine( ), que envia o caractere (ou caracteres) de final de linha preferido para a plataforma que está sendo usada para executar o programa.

Dica

Os diferentes marcadores de final de linha podem gerar problemas de conversão ao transferir arquivos de um sistema operacional para outro, como quando um usuário de Windows 95 faz upload de um arquivo para um servidor da Web que está executando o sistema operacional Linux. O uso de newLine( ) em lugar de uma literal (como ' \n ') torna seu programa mais amigável para o usuário de diferentes plataformas.

O método close( ) é chamado para fechar o fluxo de caractere colocado em buffer e garantir que todos os dados que estejam em buffer sejam enviados para o destino do fluxo.

Arquivos e filtros de nome de arquivo

Em todos os exemplos até agora, um string foi usado para fazer referência ao arquivo que está envolvido em uma operação de fluxo. Com freqüência, isso é suficiente para um programa que utiliza arquivos e fluxos, mas se você quiser copiar arquivos, renomear arquivos ou tratar de outras tarefas, um objeto Fi 1 e pode ser usado.

File, que também faz parte do pacote j ava. i o, representa uma referência a arquivo ou à pasta. Os seguintes construtores Fi 1 e podem ser usados:

- File(String) Cria um objeto File com a pasta especificada — nenhum nome de arquivo é indicado, de modo que isso se refere apenas a uma pasta de arquivo.
- File(Stríng, *String)* Cria um objeto File com o caminho de pasta e o nome especificados.
- File(File, *String)* Cria um objeto File com seu caminho representado pelo Fileespecificado e seu nome indicado pelo String especificado.

Você pode chamar vários métodos em um objeto Fi1e.

O método exi sts ( ) retorna um valor booleano indicando se o arquivo existe sob o nome e caminho de pasta estabelecidos quando o objeto Fi 1 e foi criado. Se o arquivo existe, você pode usar o método length( ) para retornar um inteiro 1 ong indicando o tamanho do arquivo em bytes.

O método renameTo(File) atribui um novo nome ao arquivo de acordo com o que estiver especificado pelo argumento *File.* Um valor booleano é retornado, indicando se a operação teve êxito.

O método delete( ) ou deleteOnExit( ) deve ser chamado para excluir um arquivo ou uma pasta. O método delete( ) tenta uma exclusão imediata (retornando um valor booleano que indica se isso funcionou). O método deleteOnExi t ( ) espera para tentar a exclusão até que o restante do programa tenha terminado de ser executado. Esse método não retorna um valor — você não pode fazer nada com as informações — e o programa deve terminar em algum ponto para que ele funcione.

O método mkdir( ) pode ser usado para criar a pasta especificada pelo objeto Fi 1 e em que ele é chamado. Ele retorna um valor booleano indicando sucesso ou falha. Não existe um método comparável para remover pastas, pois delete( ) pode ser usado em pastas e em arquivos.

Assim como acontece com todas as operações de tratamento de arquivos, esses métodos devem ser tratados com cuidado para se evitar a exclusão dos arquivos e pastas errados ou a perda de dados. Não existe um método disponível para recuperar um arquivo ou pasta excluída.

Cada um dos métodos levantará uma SecurityException, se o programa não tiver a segurança para realizar a operação de arquivo em questão; portanto, eles precisam ser tratados através de um bloco try.. .catch ou de uma cláusula throws em uma declaração de método.

O programa da listagem 17.7 converte todo o texto de um arquivo para caracteres maiúsculos. O arquivo é extraído usando-se um fluxo de entrada colocado em buffer e é lido um caractere por vez. Depois que o caractere é convertido para a forma maiúscula, ele é enviado para um arquivo temporário,

usando-se um fluxo de saída colocado em buffer. Para indicar os arquivos envolvidos, são usados objetos File em lugar de strings, o que torna possível renomear e excluir arquivos conforme for necessário.

DIGITE **Listagem 17.7.** O texto completo de allcapsdemo.javar

```
1: import java.io.*;
2:
3: public class AllCapsDemo {
4:    public static void main(String[ ] arguments) {
5:       AllCaps cap = new AlíCaps(arguments[0]);
6:       cap.convert( );
7:    }
8: }
9:
10: class AllCaps {
11:     String sourceName;
12:
13:     AllCaps(String sourceArg) {
14:         sourceName = sourceArg;
15:    }
16:
17:     void convert( ) {
18:        try {
19:           // Cria objetos de arquivo
20:           File source = new File(sourceName);
21:          File temp = new File("cap" + sourceName + ".tmp");
22:
23:            // Cria fluxo de entrada
24:            FileReader fr = new
25:                FileReader(source);
26:            BufferedReader in = new
27:                BufferedReader(fr);
28:
29:           // Cria fluxo de saída
30:           FileWriter fw = new
31:                FileWriter(temp);
32:           BufferedWriter out = new
33:                BufferedWriter(fw);
34:
35:           boolean eof » false;
36:            int inChar = 0;
37:           do {
38:              inChar = in.read( );
39:              if (inChar != -1) {
40:                char outChar = Character.toUpperCase( (char)inChar );
41:                out.write(outChar);
42:              } else
43:                   eof = true;
44:            } while (!eof);
```

Listagem 17.7. Continuação

```
45:                 in.close( );
46:                 out.close( );
47:
48:                 boolean deleted = source.delete( );
49:                 if (deleted)
50:                     temp.renameTo(source);
51:             } catch (IOException e) {
52:                 System.out.println("Error - " + e.toString( ));
53:             } catch (SecurityException se) {
54:                 System.out.println("Error - " + se.toString( ));
55:             }
56:         }
57: }
```


Após compilar o programa, você precisa de um arquivo de texto que possa ser convertido para letras maiúsculas. Uma opção é fazer uma cópia de Al 1 - CapsDemo. java e dar-lhe um nome como TempFile. java.

O nome do arquivo a ser convertido é especificado na linha de comando, ao se executar Al 1 CapsDemo, como no exemplo a seguir:

```
java AliCapsDemo TempFile.java
```

Esse programa não produz nenhuma saída. Carregue o arquivo convertido em um editor de textos para ver o resultado do aplicativo.

Resumo

Hoje você aprendeu a trabalhar com fluxos em duas direções diferentes: extraindo dados para um programa com um fluxo de entrada e enviando dados para fora de um programa com um fluxo de saída.

Você usou fluxos de byte para muitos tipos de dados não-textuais e fluxos de caractere para tratar texto. Os filtros foram associados aos fluxos para alterar o modo como as informações foram transmitidas através de um fluxo ou para alterar a própria informação.

A lição de hoje aborda a maior parte das classes do pacote java. io, mas existem outros tipos de fluxos que talvez você queira explorar. Os fluxos pipe são úteis quando se comunica dados entre diferentes threads e os fluxos de array de bytes podem conectar programas à memória de um computador.

Como as classes de fluxo em Java são tão intimamente coordenadas, você já possui a maior parte do conhecimento necessário para utilizar esses outros tipos de fluxos. Os construtores, os métodos de leitura e os métodos de escrita são praticamente idênticos.

Os fluxos representam um meio poderoso para se estender a funcionalidade de seus programas Java, pois oferecem uma conexão para qualquer tipo de dado com que você possa querer trabalhar.

Amanhã, você verá como os fluxos chegam à maior origem de dados imaginável: a Internet.

Perguntas e respostas

P **Um programa em C que utilizo, cria um arquivo de inteiros e outros dados. Posso ler isso usando um programa Java?**

R Você pode, mas um detalhe que precisa considerar é se seu programa em C representa inteiros da mesma maneira que um programa Java. Como você pode se lembrar, todos os dados podem ser representados como um byte específico ou como uma série de bytes. Um inteiro é representado em Java usando quatro bytes, que são organizados no que se chama ordem big-endian. Você pode determinar o valor inteiro combinando os bytes da esquerda para a direita. É provável que um programa em C implementado em um PC Intel represente inteiros na ordem little-endian, o que significa que os bytes devem ser organizados da direita para a esquerda para se determinar o resultado. Talvez você tenha de aprender técnicas avançadas, como deslocamento de bit, para utilizar um arquivo de dados criado com uma linguagem de programação que não seja a Java.

SEMANA 3

Dia 18

Comunicando-se via Internet

Uma das características mais notáveis da linguagem Java desde seu aparecimento é sua compatibilidade com a Internet. Como você deve se lembrar do Dia 1, a linguagem Java foi desenvolvida inicialmente como algo que pudesse controlar uma rede de dispositivos interativos, chamada Star7. Duke — o mascote animado da Sun — era a estrela desses dispositivos.

A biblioteca de classe Java inclui o pacote java.net, que torna possível estabelecer comunicação através de uma rede com seus programas Java. O pacote fornece abstrações entre plataforma para operações simples de interligação em rede, incluindo conectar e recuperar arquivos usando protocolos comuns da Web e criando sockets básicos do tipo UNIX.

Utilizadas em conjunto com os fluxos de entrada e saída, a leitura e a escrita de arquivos através da rede se torna tão fácil quanto ler ou escrever em arquivos no disco local.

Hoje, você escreverá alguns aplicativos Java compatíveis com a Internet e aprenderá por que é mais difícil fazer o mesmo com um applet. Você cria um programa que carrega um documento pela World Wide Web e investiga como são criados os programas cliente-servidor.

Interligacão em rede com Java

NOVO TERMO

Interligação em rede é a capacidade de estabelecer conexões a partir de seu applet ou aplicativos com um sistema, através da rede. Na lin-

guagem Java, a interligação em rede envolve classes do pacote java.net, que fornece abstrações entre plataformas para operações simples de interligação em rede, incluindo a conexão e a recuperação de arquivos usando protocolos comuns da Web e a criação de sockets básicos do tipo UNIX. Utilizadas em conjunto com os fluxos de entrada e saída, a leitura e a escrita de arquivos através da rede se torna tão fácil quanto ler ou escrever em arquivos no disco local.

Existem restrições, é claro. Os applets Java normalmente não podem ler ou escrever no disco da máquina em que o navegador está sendo executado. Os applets Java não podem estabelecer conexão com sistemas que não sejam aquele em que foram armazenados originalmente. Mesmo com essas restrições, você pode fazer muita coisa e tirar proveito da Web para ler e processar informações na rede.

Esta seção descreve duas maneiras simples através das quais você pode se comunicar com sistemas na Net:

- getlnputstream( ), um método que abre uma conexão com um URL e permite que você extraia dados dessa conexão
- As classes de socket, Socket e ServerSocket, que permitem abrir conexões de socket padrão com hosts e ler e escrever a partir dessas conexões

Abrindo conexões da Web

Em vez de solicitar ao navegador para que apenas carregue o conteúdo de um arquivo, às vezes você pode querer capturar o conteúdo desse arquivo para que seu applet o utilize. Se o arquivo que você quer capturar estiver armazenado na Web e puder ser acessado com as formas mais comuns de URL (http, FTP etc), seus programas Java poderão utilizar a classe URL para obtê-lo.

Por questões de segurança, por definição os applets podem estabelecer conexão apenas com o mesmo host do qual foram carregados originalmente. Isso significa que, se seus applets estiverem armazenados em um sistema chamado www.prefect.com, a única máquina em que eles poderão abrir uma conexão será essa mesma — e esse mesmo nome de host; portanto, cuidado com os alias de host. Se o arquivo que o applet quer recuperar estiver nesse mesmo sistema, a utilização de conexões de URL é a maneira mais fácil para se obtê-lo.

Essa restrição de segurança mudará o modo como você escreve e testa applets que carregam arquivos através de seus URLs. Como você não precisa lidar com conexões de rede, pode realizar todos os seus testes no disco local, simplesmente abrindo os arquivos HTML em um navegador ou com a ferramenta appl etvi ewer. Você não pode fazer isso com applets que abrem conexões de rede. Para que esses applets funcionem corretamente, você deve executar uma das seguintes tarefas:

- Executar seu navegador na mesma máquina em que seu servidor da Web está sendo executado. Se você não tem acesso ao seu servidor da Web, muitas vezes pode instalar e executar um servidor em sua máquina local.
- Fazer upload de sua classe e de arquivos HTML para seu servidor da Web sempre que quiser testá-los. Então, você executa o applet fora da página de que fez o upload, em vez de executá-lo localmente.

Você saberá quando não estiver fazendo as coisas certas com relação a garantir que seu applet e a conexão que ele está abrindo estejam no mesmo servidor. Se você tentar carregar um applet ou um arquivo a partir de servidores diferentes, obterá uma exceção de segurança, junto com muitas outras mensagens de erro assustadoras impressas em sua tela ou na console Java.

Por isso, talvez você queira trabalhar com aplicativos quando estiver estabelecendo conexão com a Internet e usar seus recursos.

Passemos para os métodos e classes para recuperação de arquivos a partir

Abrindo um fluxo na Net

Como você aprendeu no Dia 17, existem várias maneiras de extrair informações através de um fluxo para seus programas Java. As classes e métodos escolhidos dependem da forma que as informações possuem e do que se quer fazer com elas.

Um dos recursos que você pode atingir a partir de seus programas Java é um documento de texto na World Wide Web, seja ele um arquivo HTML ou algum outro tipo de documento de texto puro.

Você pode usar um processo de quatro passos para carregar um documento de texto da Web e lê-lo linha a linha:

- Crie um objeto URL que represente o endereço da World Wide Web do recurso.
- Crie um objeto URLConnection que possa carregar esse URL e estabelecer uma conexão com o site que o está hospedando.
- Usando o método getInputStream( ) desse objeto URLConnection, crie um leitor de fluxo InputStreamReader que possa ler um fluxo de dados a partir do URL.
- Usando esse leitor de fluxo de entrada, crie um objeto BufferedReader que possa ler caracteres eficientemente de um fluxo de entrada.

Muita interação ocorre entre o Ponto A — o documento da Web — e o Ponto B — seu programa Java. O URL é usado para estabelecer uma conexão de URL, que é usada para definir um leitor de fluxo de entrada, o qual é usado para definir um leitor de fluxo de entrada colocado em buffer. A necessidade

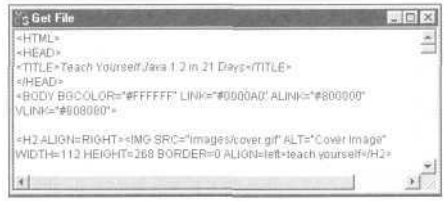
de capturar todas as exceções que ocorrem pelo caminho aumenta a complexidade do processo.

Esse é um processo confuso; portanto, é útil percorrer um programa que o implementa. A seguir há um exemplo de aplicativo que utiliza a técnica de quatro passos para abrir uma conexão com um site da Web e ler um documento HTML dele. Quando o documento está totalmente carregado, ele é apresentado em uma área de texto.

A listagem 18.1 mostra o código; a Figura 18.1 mostra o resultado depois que o arquivo foi lido.

Figura 18.1

O *aplicativo GetFile.*



Listagem 18.1. O texto completo de getfile.java.

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import java.net.*;
4: import java.io.*;
5:
6: public class GetFile extends Frame implements Runnable {
7:     Thread runner;
8:     URL page;
9:    TextArea box = new TextArea("Getting text ...");
10:
11:    public GetFile( ) {
12:        super("Get File");
13:        add(box);
14:      try {
15:            page = new
               URL("http://www.prefect.com/java21/index.html");
16:       }
17:       catch (MalformedURLException e) {
18:            System.out.println("Bad URL: " + page);
19:       }
20:    }
21:
22:    public static void main(String[ ] arguments) {
23:        GetFile frame = new GetFile( );
24:
25:        WindowListener 1 = new WindowAdapter( ) {
26:            public void windowClosing(WindowEvent e) {
27:                System.exit(0);
```

Listagem 18.1. Continuação

```
28:                 }
29:             };
30:             frame.addWindowListener(l);
31:
32:             frame.pack( );
33:             frame.setVisible(true);
34:             if (frame.runner == null) {
35:                 frame.runner = new Thread(frame);
36:                  frame.runner.start( );
37:             }
38:         }
39:
40:         public void run( ) {
41:            URLConnection conn = null;
42:             InputStreamReader in;
43:             BufferedReader data;
44:             String line;
45:             StringBuffer buf = new StringBuffer( );
46:             try {
47:                  conn = this.page.openConexão( );
48:                  conn.connect( );
49:                  box.setText("Conexão opened ...");
50:                  in = new InputStreamReader(conn.getInputStream( ));
51:                  data = new BufferedReader(in);
52:                  box.setText("Reading data ...");
53:                  while ((line = data.readLine( )) != null) {
54:                      buf.append(line + "\n");
55:                  }
56:                  box.setText(buf.toString( ));
57:             }
58:             catch (IOException e) {
59:                 System.out.println("10 Error:" + e.getMessage( ));
60:         }
61:     }
62: }
```

Esse aplicativo é uma subclasse da classe Frame e contém apenas um componente: uma área de texto. O programa utiliza threads para que o trabalho de carregar o documento HTML ocorra em seu próprio thread. Isso se torna necessário porque as conexões de rede e a entrada de fluxo podem ser tarefas demoradas, que utilizam muito o processador. Colocar esse trabalho em seu próprio thread deixa o restante do sistema de janelas livre para cuidar de suas tarefas normais, como a atualização da área de exibição e a recepção de entrada de usuário.

O método construtor GetFile( ), nas linhas 11 a 20, define o objeto URL e a área de texto em que esse documento será apresentado. O URL desse

exemplo é http://www.prefect.com/java21/index.html, que corresponde àpágina principal do site da Web deste livro. Ela pode ser qualquer outra página da Web que você conheça — experimente outras, se quiser.

O trabalho ocorre dentro do método run( ) (linhas 40 a 61). Primeiramente, os três objetos necessários para se obter os dados do documento são inicializados — URLConnection, InputStreamReader e BufferedReader. Além disso, dois objetos são criados para conter realmente os dados, quando eles chegarem — um String e um StringBuffer.

As linhas 47 e 48 abrem uma conexão de URL, o que é necessário para se obter um fluxo de entrada dessa conexão.

Alinha50usaométodogetInputStream( ) da conexão de URL para criar um novo leitor de fluxo de entrada.

A linha 51 usa esse leitor de fluxo de entrada para criar um novo leitor de fluxo de entrada colocado em buffer — um objeto BufferedReader chamado data.

Quando tiver esse leitor colocado em buffer, você pode usar seu método readLi ne ( ) para ler uma linha de texto do fluxo de entrada. O leitor colocado em buffer insere os caracteres em um buffer à medida que eles chegam e os retira desse buffer, quando solicitado.

O loop whi 1 e, nas linhas 53 a 55, lê o documento da Web linha por linha, anexando cada uma delas ao objeto StringBuffer que foi criado para conter o texto da página. Um buffer de string é usado, em vez de um string, porque você não pode modificar um string dessa maneira, durante a execução, ou seja, em runtime.

Quando todos os dados foram lidos, a linha 56 converte o buffer de string em um string com o método toStri ng ( ) e depois coloca esse resultado na área de texto do programa.

Um detalhe a ser observado nesse exemplo é que a parte do código que abriu uma conexão de rede, leu o arquivo e criou um string está colocada entre instruções try e catch. Se quaisquer erros ocorrerem enquanto você estiver tentando ler ou processar o arquivo, essas instruções permitirão a recuperação sem que o programa inteiro falhe. (Neste caso, o programa termina com um erro, pois não há muito mais a ser feito se o aplicativo não puder ler o arquivo.) As instruções try e catch proporcionam a você a capacidade de tratar e recuperar-se de erros.

Um detalhe que não está discutido aqui é o código de tratamento de eventos, nas linhas 25 a 30. Isso é necessário para que o aplicativo seja fechado com êxito, quando sua janela for fechada — você aprenderá a respeito das rotinas de tratamento de eventos para aplicativos durante o Dia 21.

Sockets

Para aplicativos de interligação em rede que exigem mais do que as classes URL e URLConnection oferecem (por exemplo, para outros protocolos ou para aplicativos de interligação em rede mais gerais), a linguagem Java fornece as classes Socket e ServerSocket como uma abstração das técnicas de programação de socket TCP.

Nota

A linguagem Java também fornece recursos para uso de sockets de datagrama (UDP), que não são abordados aqui. Consulte o pacote `java.net` na documentação Java, se você estiver interessado em trabalhar com datagramas.

A classe Socket fornece uma interface de socket no lado do cliente semelhante aos sockets UNIX padrão. Crie uma nova instância de Socket para abrir uma conexão (onde *hostName* é o host a ser conectado *eportNum* é o número da porta):

```
Socket conection = new Socket(hostName, portNum);
```

Nota

Quando usa sockets em um applet, você ainda está sujeito às restrições de segurança de applet padrão, que o impedem de estabelecer conexão com qualquer outro sistema que não seja aquele de onde veio o applet.

Uma vez que o socket esteja aberto, você pode usar fluxos de entrada e saída para ler e escrever a partir desse socket:

```
BufferedInputStream bis = new
    BufferedInputStream(connection.getInputStream( ));
DatainputStream in = new DataInputStream(bis);
BufferedOutputStream bos = new
    BufferedOutputStream(connection.getOutputStream( ));
DataOutputStream out= new DataOutputStream(bos);
```

Quando acabar de usar o socket, não se esqueça de fechá-lo. (Isso também fecha todos os fluxos de entrada e saída que você possa ter definido para esse socket.)

```
connection.close( );
```

Os sockets no lado do servidor funcionam de modo semelhante, com exceção do método accept( ). Um socket servidor verifica a existência de uma conexão em uma porta TCP; quando um cliente estabelece conexão com essa porta, o método accept( ) aceita uma conexão desse cliente. Usando sockets de cliente e servidor, você pode criar aplicativos que se comunicam através da rede.

Crie uma nova instância de ServerSocket com o número de porta para gerar um socket servidor e vinculá-lo a uma porta:

```
ServerSocket sConnection = new ServerSocket(8888);
```

Use o método accept ( ) para controlar essa porta (e aceitar uma conexão de quaisquer clientes, se algum for estabelecida):

```
sConnectiori.accept( );
```

Uma vez estabelecida a conexão de socket, você pode usar fluxos de entrada e saída para ler e escrever no cliente.

Na próxima seção, você trabalhará com algum código para implementar um aplicativo baseado em socket simples.

Para ampliar o comportamento das classes de socket — por exemplo, para permitir que as conexões de rede funcionem com uma barreira de proteção ou um proxy —, você pode usar a classe abstrata SocketImpl e a interface SocketImpl Factory para criar uma nova implementação de socket de camada de transporte. Esse design se adequa ao objetivo original das classes de socket da linguagem Java: permitir que essas classes sejam portáveis para outros sistemas com diferentes mecanismos de transporte. O problema desse mecanismo é que, embora funcione para os casos mais simples, ele impede que você inclua outros protocolos sobre o TCP (por exemplo, para implementar um mecanismo de criptografia, como o SSL) ou tenha várias implementações de socket por runtime Java.

Por esses motivos, os sockets foram ampliados após a versão 1.02 da linguagem Java, para que as classes Socket e ServerSocket não sejam finais e extensíveis. Você pode criar subclasses dessas classes que usem a implementação de socket padrão ou uma de sua própria criação. Isso possibilita a existência de recursos de rede muito mais flexíveis.

Vários outros novos recursos do pacote java.net foram acrescentados:

- Novas opções de sockets, baseadas nas opções de socket do BSD (por exemplo, TCP_NODELAY, IP_MULTICAST_LOOP, SO_BINDADDR)
- Muitas novas subclasses da classe SocketException, para representar erros de rede em um nível mais refinado de granularidade do que na Java 1.0.2 (por exemplo, NoRouteToHostException ou ConnectException)

Trivia: um socket cliente e servidor simples

Para concluir a discussão sobre interligação em rede na linguagem Java, a seguir há um exemplo de programa Java que utiliza as classes Socket para implementar um aplicativo simples baseado em rede, chamado Trivia.

O exemplo Trivia funciona da seguinte forma: o programa servidor espera pacientemente que um cliente estabeleça conexão. Quando isso acon-

tece, o servidor envia uma pergunta e espera por uma resposta. No outro lado, o cliente recebe a pergunta e solicita uma resposta do usuário. O usuário digita uma resposta, que é enviada ao servidor. O servidor, então, verifica se a resposta está correta e notifica o usuário. O servidor dá continuidade ao processo perguntando ao cliente se ele quer fazer outra pergunta. Se assim for, o processo se repete.

Fazendo o desian do Trivia

Normalmente, é uma boa idéia realizar um breve design preliminar, antes de começar a mexer com código. Com isso em mente, dê uma olhada no que é necessário para o servidor e cliente Trivia. Com relação ao servidor, você precisa de um programa que monitore as conexões de cliente de uma porta em particular na máquina host. Quando um cliente é detectado, o servidor escolhe uma pergunta aleatória e a envia para o cliente através da porta especificada. O servidor, então, entra em um estado de espera até receber resposta do cliente. Quando recebe uma resposta, o servidor a verifica e notifica o cliente do fato de ela estar correta ou não. Em seguida, o servidor quer saber do cliente se ele deseja fazer outra pergunta e, em caso afirmativo, ele entra em outro estado de espera até que o cliente responda. Finalmente, o servidor repete o processo, fazendo outra pergunta, ou finaliza a conexão com o cliente. Em resumo, o servidor executa as seguintes tarefas:

1. Espera que um cliente estabeleça uma conexão.
2. Aceita a conexão do cliente.
3. Envia uma pergunta aleatória ao cliente.
4. Espera por uma resposta do cliente.
5. Verifica a resposta e notifica o cliente.
6. Quer saber do cliente se ele tem outra pergunta.
7. Espera por uma resposta do cliente.
8. Volta ao passo 3, se necessário.

O lado cliente desse exemplo Trivia é um aplicativo executado a partir de uma linha de comando (é mais fácil demonstrá-lo desse modo). O cliente é responsável por estabelecer conexão com o servidor e esperar por uma pergunta. Quando recebe uma pergunta, o cliente a apresenta para o usuário e permite que este digite uma resposta. Essa resposta é enviada de volta para o servidor e o cliente espera novamente pela resposta do servidor. O cliente apresenta a resposta do servidor para o usuário e permite que este confirme se quer fazer outra pergunta. Então, o cliente envia a resposta do usuário para o servidor e termina, se o usuário não tiver mais perguntas a fazer. As principais tarefas do cliente são as seguintes:

1. Estabelecer conexão com o servidor.
2. Esperar que uma pergunta seja enviada.
3. Apresentar a pergunta e introduzir a resposta do usuário.
4. Enviar a resposta para o servidor.
5. Esperar por uma resposta do servidor.
6. Apresentar a resposta do servidor e pedir para que o usuário confirme outra pergunta.
7. Enviar a resposta do usuário ao servidor.
8. Voltar ao passo 2, se necessário.

Implementando o servidor Trivia

A parte principal do exemplo Trivia está no servidor. O programa servidor Trivia é chamado TriviaServer. As variáveis de instância definidas na classe TriviaServer são as seguintes:

```
private static final int PORTNUM = 1234;
private static final int WAITFORCLIENT = 0;
private static final int WAITFORANSWER = 1;
private static final int WAITFORCONFIRM = 2;
private String[ ] questions;
private String[ ] answers;
private ServerSocket serverSocket;
private int numQuestions;
private int num = 0;
private int state = WAITFORCLIENT;
private Random rand = new Random( );
```

As variáveis WAITFORCLIENT, WAITFORANSWER e WAITFORCONFIRM são constantes de estado que definem os diferentes estados em que o servidor pode estar; você verá essas constantes em ação em breve. As variáveis de perguntas e respostas são arrays de strings usados para armazenar as perguntas e as respostas correspondentes. A variável de instância serverSocket acompanha a conexão servidor-socket. numQuesti ons é usado para armazenar o número total dg perguntas, enquanto num é o número da pergunta que está sendo feita. A variável state contém o estado atual do servidor, de acordo com o que estiver definido pelas três constantes de estado (WAITFORCLIENT, WAITFORANSWER e WAITFORCONFIRM). Finalmente, a variável rand é usada para escolher perguntas aleatoriamente.

O construtor TriviaServer não faz muita coisa, exceto criar um ServerSocket, em vez de um DatagramSocket. Veja:

```
public TriviaServer( ) {
    super("TriviaServer");
    try {
        serverSocket = new ServerSocket(PORTNUM);
        System.out.println("TriviaServer up and running ...");
        }
```

```
 catch (IOException e) {
   System.err.println("Exception: couldrTt create socket");
   System.exit(I);
   }
}
```

E no método run( ) da classe Trivia Server que está a maior parte da ação. O código-fonte do método run( ) é o seguinte:

```
public void run( ) {
    Socket clientSocket = null;
    // Inicializa os arrays de perguntas e respostas
    if (!initQnA( )) {
        System.err.println("Error: couldn't initialize questions and answers");
        return;
    }

    // Procura clientes e faz perguntas triviais
    while (true) {
        // Espera por um cliente
        if (serverSocket == null)
        return;
        try {
            clientSocket = serverSocket.accept( );
        }
        catch (IOException e) {
            System.err.println("Exception: couldn't connect to client socket");
            System.exit(l);
     }

     // Realiza o processamento de pergunta/resposta
     try {
         InputStreamReader isr = new
InputStreamReader(clientSocket.getInputStream( ));
          BufferedReader is = new BufferedReader(isr);
          PrintWriter os = new PrintWriter(new
              BufferedOutputStream(clientSocket.getOutputStream( )), false);
          String outLine;
          // Produz a saída do pedido do servidor
          outLine = processlnput(nul1);
          os.println(outLine);
          os.flush( );

          // Processa e produz a saida da entrada do usuário
          while (true) {
          String inLine = is.readl_ine( );
          if (inLine.length( ) > 0) {
              outLine = processInput(inLine);
              os.println(outLine);
              os.flush( );
              if (outLine.equals("Bye."))
                  break;
          }
       }

         // Limpeza
        os.closef );
```

```
            is.close( );
            clientSocket.close( );
          }
          catch (Exception e) {
              System.err.println("Exception: " + e);
              e.printStackTrace( );
          }
        }
}
```

Primeiramente, o método run( ) inicializa as perguntas e respostas, chamando initQnA( ). Você aprenderá a respeito do método initQnA( ) em breve. Um loop whi 1 e infinito, que espera por uma conexão de cliente, é então introduzido. Quando um cliente estabelece uma conexão, são criados os fluxos de I/O apropriados e a comunicação é tratada através do método process Input( ). Você aprenderá a respeito de processlnput( ) em seguida, process-Input( ) processa continuamente as respostas do cliente e manipula todas as novas perguntas até que o cliente decida não receber mais nenhuma pergunta. Isso é evidenciado pelo fato de o servidor enviar o string " Bye". Então, o método run( ) faz a limpeza dos fluxos e do socket cliente.

O método processlnput( ) acompanha o estado do servidor e gerencia a lógica de todo o processo de pergunta/resposta. O código-fonte de process-Input é o seguinte:

```
String processInput(String inStr) {
   String outStr = null;

   switch (state) {
       case WAITFORCLIENT:
          // Faz uma pergunta
          outStr = questions[num];
          state = WAITFORANSWER;
          break;

       case WAITFORANSWER:
          // Verifica a resposta
          if (inStr.equalsIgnoreCase(answers[num]))
          outStr = "Thafs correct! Want another? (y/n)";
          ei se
             outStr = "Wrong, the correct answer is " + answers[num] +
                 ". Want another? (y/n)";
         state = WAITFORCONFIRM;
         break;

     case WAITFORCONFIRM:
        // Verifica se o usuário quer fazer outra pergunta
        if (inStr.equalsIgnoreCase("Y")) {
            num = Math.abs(rand.nextlnt( )) % questions.length;
            outStr = questions[num];
            state = WAITFORANSWER;
        }
        else {
            outStr = "Bye.";
            state = WAITFORCLIENT;
```

```
            }
            break;
        }
        return outStr;
    }
```

O primeiro detalhe a ser observado com relação ao método processInput( ) é a variável local outStr. O valor desse string é enviado de volta ao cliente no método run, quando processInput retorna; portanto, veja como próceSSInput utiliza outStr para transmitir informações para o cliente.

No TriviaServer, o estado WAITFORCLIENT representa o servidor quando ele está ocioso e esperando por uma conexão de cliente. Entenda que cada instrução case presente em processInput( ) representa o servidor que está saindo de um determinado estado. Por exemplo, a instrução case WAITFORCLIENT é introduzida quando o servidor acabou de sair do estado WAITFORCLIENT — um cliente acabou de estabelecer conexão com o servidor. Quando isso ocorre, o servidor ajusta o string de saída de acordo com a pergunta atual e define o estado para WAITFORANSWER.

Se o servidor está deixando o estado WAITFORANSWER, o cliente acabou de responder. processInput( ) compara a resposta do cliente com a resposta correta e ajusta o string de saída de acordo. Em seguida, ele define o estado como WAITFORCONFIRM.

O estado WAITFORCONFIRM representa o servidor esperando por uma resposta de confirmação do cliente. Em processInput( ), a instrução case WAITFORCONFIRM indica que o servidor está deixando o estado, pois o cliente retornou uma confirmação (sim ou não). Se o cliente respondeu afirmativamente com umy, então processInput escolhe uma nova pergunta e define o estado devolta para WAITFORANSWER. Caso contrário, o servidor apresenta "Bye" para o cliente e volta ao estado WAITFORCLIENT para esperar uma nova conexão de cliente.

As perguntas e respostas presentes em Trivia são armazenadas em um arquivo de texto chamado QnA.txt, que é organizado com perguntas e respostas em linhas alternadas. Cada pergunta é seguida de sua resposta na linha seguinte, a qual, por sua vez, é seguida da próxima pergunta. A seguir está uma listagem do arquivo QnA.txt:

```
What caused the craters on the moon?
meteorites
How far away is the moon (in miles)?
239000
How far away is the sun (in millions of miles)?
93
Is the Earth a perfect sphere?
no
What is the internai temperature of the Earth (in degrees F)?
9000
```

O método initQnA( ) realiza o trabalho de ler as perguntas e respostas do arquivo de texto e armazená-las em arrays de strings separados, como mostrado a seguir:

```
private boolean initQnA( ) {
    try {
        File inFile = new FileCQnA.txt");
        FileInputStream inStream = new FileInputStream(inFile);
        byte[ ] data = new byte[(int)inFile.length( )];
        // Lê as perguntas e respostas em um array de bytes
        if (inStream.read(data) <= 0) {
            System.err.println("Error: couldrTt read questions and answers");
            return false;
        }
            // Verifica quantos pares pergunta/resposta existem
            for (int i = 0; i < data.length; i++)
                if (data[i] == (byte)'\n')
                    numQuestions++;
            numQuestions /= 2;
            questions = new String[numQuestions];
            answers = new String[numQuestions];
            // Analisa as perguntas e respostas em arrays de strings
            int start = 0, index = 0;
            boolean isQ = true;
            for (int i = 0; i < data.length; i++)
                if (data[i] == (byte)'\n') {
                    if (isQ) {
                       questions[index] = new String(data, start, i - start - 1);
                         isQ = false;
            }
            else {
               answers[index] = new String(data, start, i - start - 1);
                isQ = true;
                 index++;
             }
         start = i + 1;
     }
     }
     catch (FileNotFoundException e) {
       System.err.println("Exception: couldn't find the question file");
     return false;
     }
   catch (IOException e) {
       System.err.println("Exception: 1/0 error trying to read questions");
       return false;
     }
       return true;
}
```

O método initQnA( ) utiliza dois arrays e os preenche com strings alternados do arquivo QnA.txt: primeiro uma pergunta e, em seguida, uma resposta, alternando até que o final do arquivo seja atingido.

O único método restante em Tri vi aServer é mai n ( ); mai n simplesmente cria o objeto server e o inicia com uma chamada ao método start:

```
public static void main(String[ ] arguments) {
    TriviaServer server = new TriviaServer( );
    server.start( );
}
```

A listagem 18.2 contém o código-fonte completo do aplicativo servidor.

DIGITE

Listagem 18.2 O texto completo de triviaserver.java.

```
1: import java.io.*;
2: import java.net.*;
3: import java.util.Random;
4:
5: public class TriviaServer extends Thread {
6:   private static final int PORTNUM - 1234;
7:   private static final int WAITFORCLIENT = 0;
8:   private static final int WAITFORANSWER = 1;
9:   private static final int WAITFORCONFIRM = 2;
10:  private String[ ] questions;
11:  private Stringf ] answers;
12:  private ServerSocket serverSocket;
13:  private int numQuestions;
14:  private int num = 0;
15:  private int state = WAITFORCLIENT;
16:  private Random rand • new Random( );
17:
18:  public TriviaServer( ) {
19:      super("TriviaServer");
20:     try {
21:         serverSocket = new ServerSocket(PORTNUM);
22:         System.out.println("TriviaServer up and running ...");
23:     }
24:     catch (IOException e) {
25:         System.err.println("Exception: couldn't create socket");
26:         System.exit(l);
27:     }
28:  }
29:
30:    public static void main(String[ ] arguments) {
31:        TriviaServer server = new TriviaServer( );
32:        server.start( );
33:   }
34:
35:    public void run( ) {
36:        Socket clientSocket = null;
37:
38:        // Inicializa os arrays de perguntas e respostas
39:        if (!initQnA( )) {
40:            System.err.println("Error: couldn't initialize questions
               and answers");
41:            return;
```

Listagem 18.2 Continuação

```
42:       }
43:
44:       // Procura clientes e faz perguntas triviais
45:       while (true) {
46:           // Espera por um cliente
47:           if (serverSocket == null)
48:               return;
49:       try {
50:           clientSocket = serverSocket.accept( );
51:       }
52:       catch (IOException e) {
53:           System.err.println("Exception: couldn't connect to
              client socket");
54:           System.exit(l);
55:       }
56:
57:       // Executa o processamento de pergunta/resposta
58:       try {
59:           InputStreamReader isr = new
              InputStreamReader(clientSocket.getInputStream( ));
60:           BufferedReader is = new BufferedReader(isr);
61:           PrintWriter os = new PrintWriter(new
62:           BufferedOutputStream(clientSocket.getOutputStream( )),
              fai se);
63:           String outLine;
64:
65:           // Produz a saída do pedido do servidor
66:           outLine = processlnput(null);
67:           os.println(outLine);
68:           os.flush( );
69:
70:           // Processa e produz a saída da entrada do usuário
71:           while (true) {
72:               String inLine = is.readLine( );
73:               if(inLine.length( ) > 0) {
74:                   outLine = processInput(inLine);
75:                   os.printin(outLine);
76:                   os.flush( );
77:                    if (outLine.equals("Bye."))
78:                        break;
79:               }
80:           }
81:
82:           // Limpeza
83:           os.close( );
84:           is.close( ) ;
85:           clientSocket.close( );
86:       }
```

Listagem 18.2 Continuação

```
87:         catch (Exception e) {
88:            System.err.println("Exception: " + e);
89:             e.printStackTrace( );
90:         }
91:       }
92: }
93:
94:   private boolean initQnA( ) {
95:     try {
96:           File inFile = new File("QnA.txt");
97:           FileInputStream inStream = new FileInputStream(inFile);
98:           byte[ ] data = new byte[(int)inFile.length( )];
99:
100:          // Lê as perguntas e respostas em um array de bytes
101:          if (inStream.read(data) <= 0) {
102:              System.err.println("Error: couldn't read questions
                  and answers");
103:              return false;
104:          }
105:
106:          // Verifica quantos pares pergunta/resposta existem
107:         for (int i = 0; i < data.length; i++)
108:             if (data[i] == (byte)'\n')
109:                 numQuestions++;
110:          numQuestions /= 2;
111:          questions = new String[numQuestions];
112:          answers = new String[numQuestions];
113:
114:          // Analisa as perguntas e respostas em arrays de strings
115:          int start = 0, index = 0;
116:          boolean isQ = true;
117:         for (int i = 0; i < data.length; i++)
118:              if (data[i] == (byte)'\n') {
119:                if (isQ) {
120:                    questions[index] = new String(data, start,
                        i - start - 1);
121:                    isQ = false;
122:                }
123:                else {
124:                    answers[index] = new String(data, start,
                        i - start - 1);
125:                    isQ = true;
126:                    index++;
127:                }
128:              start = i + 1;
129:          }
130:      }
131:       catch (FileNotFoundException e) {
```

Listagem 18.2 Continuação

```
132:          System.err.println("Exception: couldn't find the question file");
133:          return false;
134:       }
135:       catch (IOException e) {
136:          System.err.println("Exception: I/O error trying to read questions");
137:          return false;
138:       }
139:
140:       return true;
141:     }
142:
143:     String processInput(String inStr) {
144:        String outStr = null;
145:
146:        switch (state) {
147:            case WAITFORCLIENT:
148:               // Faz uma pergunta
149:               outStr = questions[num];
150:               State = WAITFORANSWER;
151:               break;
152:
153:           case WAITFORANSWER:
154:              // Verifica a resposta
155:              if (inStr.equalsIgnoreCase(answers[num]))
156:                 outStr = "Thafs correct! Want another? (y/n)";
157:              else
158:                 outStr = "Wrong, the correct answer is " +
                       answers[num] +
159:                       ". Want another? (y/n)";
160:              state = WAITFORCONFIRM;
161:              break;
162:
163:           case WAITFORCONFIRM:
164:              // Verifica se o usuário quer fazer outra pergunta
165:              if (inStr.equalsIgnoreCase("Y")) {
166:              num = Math.abs(rand.nextlnt( )) % questions.length;
167:              outStr = questions[num];
168:               state = WAITFORANSWER;
169:           }
170:       else      {
171:                 outStr = "Bye.";
172:                 state = WAITFORCLIENT;
173:           }
174:           break;
175:        }
176:        return outStr;
177:     }
178: }
```

Implementando o cliente Trivia

Como o lado cliente do exemplo Trivia exige que o usuário digite respostas e receba retorno do servidor, é mais simples implementá-lo como um aplicativo de linha de comando. Talvez isso não seja tão bonito como um applet gráfico, mas torna muito fácil ver os eventos de comunicação à medida que eles ocorrerem. O aplicativo cliente é chamado Trivia.java.

A única variável de instância definida na classe Trivia é PORTNUM, que define o número de porta usado pelo cliente e pelo servidor. Também existe apenas um método definido na classe Trivia: main( ). O código-fonte do método main( ) está incluído na listagem 18.3.

DIGITE

Listagem 18.3. O texto completo de trivia.java.

```
1: import java.io.*;
2: import java.net.*;
3:
4: public class Trivia {
5:     private static final int PORTNUH = 1234;
6:
7:     public static void main(String[ ] arguments) {
8:         Socket socket - null;
9:         InputStreamReader isr = null;
10:        BufferedReader in = null;
11:        PrintWriter out = null;
12:        String address;
13:
14:        // Verifica o endereço do host nos argumentos de linha de comando
15:        if (arguments.length != 1) {
16:            System.out.println("Usage: java Trivia <address>");
17:            return;
18:        }
19:        else
20:            address = arguments[0];
21:
22:        // Inicializa o socket e os fluxos
23:      try {
24:            socket = new Socket(address, PORTNUM);
25:            isr = new InputStreamReader(socket.getInputStream( ));
26:            in = new BufferedReader(isr);
27:            out = new PrintWriter(socket.getOutputStream( ),true);
28:        }
29:        catch (IOException e) {
30:            System.err.println("Exception: couldrTt create stream
               socket "
31:                + e.getMessage( ));
32:            System.exit(l);
33:        }
34:
```

Listagem 18.3. Continuação

```
35:         // Processa a entrada do usuário e as respostas do servidor
36:          try {
37:              StringBuffer str = new StringBuffer(128);
38:              String inStr;
39:              int c;
40:
41:              while ((inStr = in.readLine( )) != null) {
42:                  System.out.println("Server: " + inStr);
43:                   if (inStr.equals("Bye."))
44:                       break;
45:                  while ((c = System.in.read( )) != '\n')
46:                       str.append((char)c);
47:                  System.out.println("Client: " + str);
48:                      out.println(str.toString( ));
49:                      out.flush( );
50:                     str.setLength(O);
51:              }
52:              // Limpeza
53:              out.close( );
54:              in.close( );
55:              socket.close( );
56:          }
57:          catch (IOException e) {
58:              System.err.println("I/0 error: "+ e.toString( ));
59:          }
60:       }
61: }
```

O primeiro detalhe interessante que você poderia notar com relação ao método main( ) é que ele procura um argumento de linha de comando. O endereço do servidor (como pref ect. com), é o argumento de linha de comando do cliente Tri vi a. Como se trata de um aplicativo Java e não de um applet, não é suficiente estabelecer uma conexão com o servidor de onde veio o applet — não existe servidor padrão; portanto, você pode estabelecer conexão com qualquer servidor que queira. No aplicativo cliente, você precisa codificar o endereço do servidor ou solicitá-lo como um argumento de linha de comando. Se você codificar isso, precisará recompilar sempre que quiser mudar algo. Daí o argumento de linha de comando!

A maioria dos leitores provavelmente não terá acesso a um servidor da Web que execute programas Java no lado do servidor, como o aplicativo Tri vi aServer. Em alguns sistemas operacionais, você pode testar programas servidores executando o servidor Trivia em uma janela e o cliente Trivia em outra, usando o nome de domínio "localhost". A seguir está um exemplo:

```
java Trivia "localhost"
```

Isso faz a linguagem Java procurar no host local — em outras palavras, no sistema que está executando o aplicativo —, um servidor para fazer contato. Dependendo de como foram configuradas as conexões com a Internet em seu sistema, talvez você precise estabelecer conexão com a Net antes que uma conexão de socket com êxito possa ser feita entre o cliente `Trivia` e seu servidor.

Se o argumento de linha de comando com o endereço do servidor for válido (não null), o método main( ) criará o socket e os fluxos de I/O necessários. Em seguida, ele entra em um loop while, onde processa informações do servidor e transmite os pedidos do usuário de volta para o servidor. Quando o servidor sai, solicitando informações, o loop whi 1 e falha e o método main( ) faz a limpeza do socket e dos fluxos — e isso é tudo para o cliente Trivia!

Executando o Trivia

Assim como o Fortune, o servidor Tri vi a deve estar sendo executado para que o cliente funcione. Para fazer com que tudo comece, primeiramente você deve executar o servidor, usando o interpretador Java. Isso é feito a partir de uma linha de comando, como a seguinte:

```
java TriviaServer
```

O cliente Tri vi a também é executado a partir de uma linha de comando, mas você deve especificar um endereço de servidor como único argumento. Um exemplo de execução do cliente Trivia e da conexão com o servidor 1 ocal host aparece a seguir:

```
java Trivia "localhost"
```

Você também pode tentar executá-lo usando o endereço IP que representa a porta "localhost". Esse comando é o seguinte:

```
java Trivia "127.0.0.1"
```

Após executar o cliente Trivia e responder a algumas perguntas, você deverá ver uma saída semelhante à seguinte:

SAÍDA

```
Server: What is the internai temperature of the Earth (in degrees F)?
Client: meteorites
Server: Wrong, the correct answer is 9000. Want another? (y/n)
Client: y
Server: Is the Earth a perfect sphere?
Client: 93
Server: Wrong, the correct answer is no. Want another? (y/n)
Client: y
```

```
Server: What is the internai temperature of the Earth (in degrees F)?
Client: 9000
Server: Thafs correct! Want another? (y/n)
Client: n
Server: Bye.
```

Resumo

A interligação em rede possui muitas aplicações, das quais seus aplicativos podem fazer uso. Você pode não ter percebido, mas o projeto GetFi 1 e foi um navegador da Web rudimentar. Ele trouxe o texto de uma página da Web para um programa Java e o apresentou. E claro que a análise do código HTML é o que transforma as várias tags de marcação em uma página da Web real. A Sun escreveu um navegador da Web inteiro em Java — o Hotjava.

Hoje, você aprendeu a usar URLs, conexões de URL e fluxos de entrada em conjunto com a extração de dados da World Wide Web para seu programa.

Você também aprendeu como os programas cliente e servidor são escritos em Java e como um programa servidor fica em uma porta da Internet, esperando que um programa cliente entre em contato com ele.

Perguntas e respostas

P **Como posso imitar o envio de um formulário HTML em um applet Java?**

R Atualmente, os applets tornam difícil fazer isso. O melhor (e o modo mais fácil) é usar a notação GET para fazer o navegador enviar o conteúdo do formulário para você.

Os formulários HTML podem ser enviados de duas maneiras: usando-se o pedido GET ou usando-se POST. Se você usar GET, as informações de seu formulário serão codificadas no próprio URL, algo parecido com o seguinte:

```
http://www.bl ah.com/cgi-bi n/myscri pt?foo=l&bar=2&name=Laura
```

Como a entrada do formulário é codificada no URL, você pode escrever um applet Java para imitar um formulário, ober entrada do usuário e depois construir um novo objeto URL com os dados do formulário incluídos no final; em seguida, basta passar esse URL para o navegador, usando getAppletContext( ), showDocument( ) e o navegador enviará os resultados do formulário sozinho. Para formulários simples, basta isso.

P **Como posso postar envios de formulário?**

R Você tem de imitar o que um navegador faz para enviar formulário usando POST. Abra um socket para o servidor e envie os dados, o que é semelhante ao seguinte. (O formato exato é determinado pelo protocolo HTTP; este é apenas um subconjunto dele.)

```
POST /cgi-bin/mailto.cgi HTTP/1.0
Content-type: application/x-www-form-urlencoded
Content-length: 36
{os dados de seu formulário codificado ficam aqui}
```

Se fizer tudo certo, você obterá a saída do formulário CGI de volta do servidor. Então, dependerá de seu applet manipular essa saída corretamente. Observe que, se a saída for um código HTML, não haverá um modo fácil de passá-la para o navegador que ainda está executando seu applet. Esse recurso pode acabar nas versões futuras da linguagem Java. Entretanto, se você obtiver um URL em retorno, poderá redirecionar o navegador para esse URL.

SEMANA 3

Dia 19

JavaBeans e outros recursos avançados

Em uma linguagem que está crescendo tão rapidamente quanto a Java é fácil ficar espantado com a quantidade de classes oferecidas pela Sun. Todo novo lançamento da linguagem traz interessantes recursos avançados, como JavaBeans, Java Database Connectivity e geração de imagens bidimensionais.

Felizmente, você não precisa dominar todas as partes da biblioteca de classe Java padrão, antes de poder criar programas úteis. Você pode se concentrar nos pacotes e classes que são necessários em sua área de atuação e ampliar seus conhecimentos em novas áreas quando for necessário.

Hoje, você verá alguns recursos avançados oferecidos nas versões mais recentes da linguagem Java, incluindo o seguinte:

- JavaBeans
- Java Database Connectivity
- Remote Method Invocation
- Segurança

De modo geral, o objetivo de hoje é familiarizá-lo com o assunto como o primeiro passo para a utilização dessas classes. Entretanto, você também tem uma chance de realizar algum trabalho prático com a transferência de dados e com a comunicação entre applets e navegadores.

JavaBeans

Uma tendência crescente no campo de desenvolvimento de software é a idéia dos componentes reutilizáveis — elementos de um programa que podem ser usados com mais de um pacote de software.

NOVO TERMO Um *componente de software* é um software isolado em uma estrutura separada e facilmente reutilizável.

Se você desenvolver partes de um programa de modo que elas sejam totalmente independentes, é possível que esses componentes sejam montados em programas com uma eficiência de desenvolvimento muito maior. Essa noção de reutilização cuidadosa de software empacotado foi emprestada, até certo ponto, da estratégia de linha de montagem que se tornou muito popular nos Estados Unidos, durante a revolução industrial, bem antes da moderna era dos computadores. A idéia, aplicada ao software, é construir uma só vez pequenos componentes reutilizáveis e depois reutilizá-los o máximo possível, otimizando assim todo o processo de desenvolvimento.

Talvez a maior dificuldade que o software componente tenha de enfrentar seja a ampla gama de microprocessadores e sistemas operacionais díspares em uso atualmente. Tem havido uma variedade de tentativas razoáveis com relação a software de componente, mas elas sempre estiveram limitadas e um sistema operacional específico. As arquiteturas de componente VBX e OCX da Microsoft tiveram grande sucesso no mundo dos PCs Intel, mas elas pouco fizeram para diminuir o distanciamento entre outros tipos de sistemas operacionais.

Nota

A tecnologia ActiveX da Microsoft, que tem por base sua tecnologia OCX, tem por objetivo fornecer uma tecnologia de componentes de propósito geral compatível com uma ampla variedade de plataformas. No entanto, considerando a dependência do ActiveX com relação ao código Windows de 32 bits, ainda não se sabe como a Microsoft vai resolver a questão da dependência de plataforma.

Algumas tecnologias de componente existentes também sofrem o fato de terem sido desenvolvidas em uma linguagem de programação em particular ou em um ambiente de desenvolvimento específico. Assim como a dependência de plataforma incapacita componentes em tempo de execução, limitar o desenvolvimento de componentes a uma linguagem de programação ou um ambiente de desenvolvimento em particular os incapacita igualmente. Os desenvolvedores de software querem decidir sozinhos qual linguagem é a mais apropriada para uma tarefa em particular. Do mesmo modo, os desenvolvedores querem selecionar o ambiente de desenvolvimento mais adequado às suas necessidades.

em vez de serem forçados a utilizar um ambiente baseado nas restrições de uma tecnologia de componentes. Portanto, qualquer tecnologia de componentes realista a longo prazo deve tratar do problema da dependência de plataforma e da dependência da linguagem.

A linguagem Java foi um fator importante para tornar realidade a confecção de um ambiente de software realmente independente da plataforma, e ela oferece desenvolvimento de componentes de software através do JavaBeans.

O JavaBeans é um conjunto de classes independente de arquitetura e de plataforma para a criação e uso de componentes de software Java. O JavaBeans começa onde as outras tecnologias de componente terminam, usando a plataforma portável Java como base para o fornecimento de uma solução de software de componente completa.

O obietivo do JavaBeans

O JavaBeans foi projetado para ser compacto, pois freqüentemente os componentes serão usados em ambientes distribuídos, onde componentes inteiros podem ser transferidos por uma conexão de baixa largura de banda com a Internet. A segunda parte desse objetivo está relacionada à facilidade com que os componentes são construídos e utilizados. Não é muito difícil imaginar componentes que sejam fáceis de usar, mas criar uma arquitetura de componente que torne fácil construir componentes é uma questão totalmente diferente.

Os componentes JavaBeans são amplamente baseados na estrutura[1] de classe já em uso na programação de applets Java tradicional e os applets projetados usando-se o Abstract Windowing Toolkit podem se transformar facilmente em novos componentes JavaBeans. Isso também tem o efeito positivo de tornar os componentes JavaBeans muito compactos, pois os applets Java já são muito eficientes em termos de tamanho.

O segundo principal objetivo do JavaBeans é ser totalmente portável. Como resultado, os desenvolvedores não precisarão se preocupar com a inclusão de bibliotecas específicas de plataforma em seus applets Java.

A arquitetura Java existente já oferece uma ampla gama de benefícios facilmente aplicados aos componentes. Um dos mais importantes recursos da linguagem Java, porém raramente mencionado, é seu mecanismo interno de descoberta de classe, que permite que os objetos interajam entre si de forma dinâmica. Isso resulta em um sistema em que os objetos podem ser integrados entre si independentemente de suas respectivas origens ou histórico de desenvolvimento. O mecanismo de descoberta de classe não é apenas um interessante recurso da linguagem Java; ele é um requisito necessário em qualquer arquitetura de componente.

NOVO TERMO

Outro exemplo da herança da funcionalidade existente na linguagem Java por parte do JavaBeans é *a persistência*, que se refere à capacidade de um objeto armazenar e recuperar seu estado interno. A persistência é

manipulada automaticamente no JavaBeans, usando-se simplesmente o mecanismo de serialização já presente na linguagem Java. *Serialização* é o processo de armazenar ou recuperar informações através de um protocolo padrão. Alternativamente, os desenvolvedores podem criar soluções de persistência personalizada, quando necessário.

Embora não seja um elemento fundamental da arquitetura JavaBeans, é fornecido suporte à computação distribuída. Os desenvolvedores de componentes JavaBeans têm a opção de selecionar uma estratégia de computação distribuída que seja mais adequada às suas necessidades. A JavaSoft fornece uma solução de computação distribuída em sua tecnologia RMI (Remote Method Invocation), mas os desenvolvedores de JavaBeans não estão presos a essa solução. Outras opções incluem o CORBA (Common Object Request Broker Architecture) e o DCOM (Distributed Component Object Model) da Microsoft, entre outros.

A computação distribuída foi devidamente abstraída do JavaBeans para manter as coisas concisas, enquanto oferece uma ampla gama de opções aos desenvolvedores que exijam suporte distribuído. O último objetivo de design do JavaBeans trata dos problemas de projeto e como os desenvolvedores criam aplicativos usando componentes JavaBeans.

A arquitetura JavaBeans inclui suporte à especificação de propriedades durante o design e mecanismos para facilitar a edição visual de componentes JavaBeans. O resultado é que os desenvolvedores poderão utilizar ferramentas visuais para montar e modificar componentes JavaBeans de maneira transparente, de forma muito parecida com o modo como as ferramentas visuais existentes para PC funcionam com componentes como controles VBX ou OCX. Dessa maneira, os desenvolvedores de componente especificam o modo como os componentes serão utilizados e manipulados em um ambiente de desenvolvimento.

Como o JavaBeans se relaciona com a linguagem Java

Embora a natureza orientada a objetos da linguagem Java forneça um meio para os objetos trabalharem em conjunto, existem algumas regras ou padrões que governam o modo como as interações de objeto são conduzidas. Esses tipos de regras são necessárias para uma solução de software de componente robusta e elas são fornecidas através do JavaBeans.

O JavaBeans especifica um rico conjunto de mecanismos para a interação entre objetos, junto com ações comuns a que a maioria dos objetos precisará oferecer suporte, como persistência e tratamento de eventos.

O JavaBeans fornece a estrutura através da qual essa comunicação de componente pode ocorrer. Ainda mais importante é o fato de que os componentes JavaBeans podem ser facilmente otimizados com um conjunto padrão de propriedades bem definidas. O JavaBeans mescla o poder dos applets Java

completos com o fato de os componentes Java AWT, como os botões) serem compactos e reutilizáveis.

Entretanto, os componentes JavaBeans não estão limitados aos objetos de interface com o usuário, como os botões. Você pode, com a mesma facilidade, desenvolver componentes JavaBeans não-visuais, que executam alguma função em segundo plano em conjunto com outros componentes. Desse modo, o JavaBeans mescla o poder dos applets Java com os aplicativos Java não-visuais em uma estrutura de componente coerente.

NOVO TERMO Um *componente não-visual* é qualquer componente que não tenha saída visível. Quando se considera os componentes em termos de objetos AWT, como botões e menus, isso pode parecer um pouco estranho. No entanto, lembre-se de que um componente é apenas um programa fortemente empacotado e não tem nenhum requisito específico de ser visual. Um bom exemplo de componente não-visual é um cronômetro, que dispara eventos de temporização em intervalos específicos. Os componentes do cronômetro são muito populares em outros ambientes de desenvolvimento de componentes, como o Microsoft Visual Basic.

Você pode usar uma variedade de componentes JavaBeans em conjunto, sem necessariamente escrever nenhum código usando ferramentas visuais. Os componentes JavaBeans expõem suas interfaces visualmente, fornecendo um modo de editar suas propriedades sem programação. Além disso, você pode simplesmente "soltar" um componente JavaBeans diretamente em um aplicativo, sem escrever nenhum código através de um editor visual. Esse é um nível totalmente novo de flexibilidade e reutilização, anteriormente impossível na linguagem Java.

A AN JavaBeans

Em última instância, o JavaBeans é uma interface de programação, significando que todos os seus recursos são implementados como extensões da biblioteca de classe Java padrão; toda a funcionalidade fornecida pelo JavaBeans é, na verdade, implementada na API JavaBeans, um conjunto de APIs menores, dedicadas a funções (serviços) específicas. A seguir está uma lista dos principais serviços de componente presentes na API JavaBeans, que são necessários para facilitar todos os recursos sobre os quais você aprendeu hoje:

- Mesclagem de interface gráfica com o usuário
- Persistência
- Tratamento de eventos
- Introspecção
- Suporte a construtor de aplicativos

Entendendo esses serviços e como eles funcionam, você terá uma idéia melhor sobre a espécie de tecnologia em que o JavaBeans se enquadra. Cada um desses serviços é implementado na forma de APIs menores, contidas dentro da API JavaBeans, mais abrangente.

As APIs de mesclagem de interface com o usuário fornecem um modo para um componente mesclar seus elementos com um container. A maioria dos containers tem menus e barras de ferramentas que precisam apresentar todos os recursos especiais fornecidos pelo componente. As APIs de mesclagem de interface permitem que o componente acrescente recursos no menu e na barra de ferramentas do documento container. Essas APIs também definem o mecanismo que facilita o layout da interface entre os componentes e seus containers.

As APIs persistentes especificam o mecanismo pelo qual os componentes podem ser armazenados e recuperados dentro do contexto de um documento contêiner. Por definição, os componentes herdam o mecanismo de serialização automática fornecido pela linguagem Java. Os desenvolvedores também estão livres para fazer o design de soluções de persistência mais elaboradas, baseadas nas necessidades específicas de seus componentes.

As APIs de tratamento de eventos especificam uma arquitetura dirigida por eventos que define como os componentes interagem entre si. O AWT Java já inclui um poderoso modelo de tratamento de eventos que serve como base para as APIs de componente de tratamento de eventos. Essas APIs são importantes para permitir aos componentes a liberdade de interagir entre si de maneira coerente.

As APIs de introspecção definem as técnicas pelas quais os componentes tornam sua estrutura interna prontamente disponível, durante o projeto. Essas APIs são compostas da funcionalidade necessária para permitir que ferramentas de desenvolvimento consultem um componente quanto ao seu estado interno, incluindo as interfaces, métodos e variáveis membro de que o componente é composto. As APIs são divididas em duas seções distintas, baseadas no nível em que estão sendo utilizadas. Por exemplo, as APIs de introspecção de baixo nível permitem que as ferramentas de desenvolvimento acessem diretamente os detalhes internos do componente, que é uma função que você não desejaria necessariamente que estivesse nas mãos dos usuários do componente. Isso o leva às APIs de alto nível. As APIs de alto nível utilizam as APIs de baixo nível para determinar as partes de um componente que são exportadas para modificação pelo usuário; portanto, embora as ferramentas de desenvolvimento indubitavelmente façam uso das duas APIs, elas usarão as de alto nível apenas quando estiverem fornecendo informações sobre o componente para o usuário.

As APIs de suporte a construtor de aplicativos fornecem a sobrecarga necessária para a edição e manipulação de componentes durante o projeto. Essas APIs são usadas amplamente por ferramentas de desenvolvimento

visuais, para fornecer um modo de dispor e editar visualmente os componentes, enquanto se constrói um aplicativo. A seção de um componente que fornece recursos de edição visual é projetada de forma específica para ser fisicamente separada do componente em si. Isso porque os componentes de runtime independentes podem ser o mais compactos possível. Em um ambiente puramente de runtime, os componentes são transferidos apenas com o componente de runtime necessário. Os desenvolvedores que quiserem usar os recursos de componente durante o projeto podem obter facilmente a parte relacionada ao projeto do componente.

As especificações do JavaBeans estão disponíveis no site da Web sobre Java, no endereço:

```
http://www.sun.com
```

Truques de applet

NOVO TERMO Se você é um surfista da Web ativo, provavelmente já viu applets Java que apresentam mensagens ao longo da *linha de status* — a parte de um navegador da Web que indica o hyperlink em que se está conectado, dentre outras informações. Esta seção o ensina como obter esse efeito e outras técnicas conhecidas para applets.

O método showStatus( )

O método showStatus( ) da classe Applet permite que você apresente um string na barra de status do navegador que está executando o applet. Você pode usar isso para apresentar mensagens de erro, hyperlinks, ajuda e outras mensagens de status.

Esse método pode ser chamado com uma instrução como a seguinte:

```
getAppletContext( ).showStatus("Click applet window to begin");
```

O método getAppletContext( ) permite que seu applet acesse recursos do navegador que o contém. showStatus ( ) usa esse mecanismo para apresentar mensagens de status.

Informações de applet

O Abstract Windowing Toolkit fornece um mecanismo para associar as informações ao seu applet, como o autor, data do copyright e outros detalhes relevantes. Um navegador da Web pode incluir um mecanismo para apresentar essas informações, se o desenvolvedor do applet as tiver fornecido.

Para fornecer informações sobre seu applet, anule o método getAppletInfo( ) no seguinte método:

```
public String getAppletInfo( ) {
    return "GetRaven Copyright 1998 Laura Lemay";
}
```

Criando vínculos dentro de applets

Como os applets são executados dentro de navegadores da Web, é interessante usar a capacidade desse navegador para carregar novas páginas da Web. A linguagem Java fornece um mecanismo para se instruirão navegador a carregar uma nova página. Uma das maneiras de se usar esse mecanismo seria para criar mapas de imagem animados que carregam uma nova página quando recebem um clique de mouse.

Você cria uma nova instância da classe URL para vincular-se a uma nova página, mas é muito mais fácil com isso aqui do que nos projetos de ontem.

Para criar um novo objeto URL, use um de quatro construtores diferentes:

- URL (Si*ring)* cria um objeto URL a partir de um endereço completo da Web, como por exemplo http://www.prefect.com/java21 ou ftp://ftp.netscape.com.
- U*RL(URL, String)* cria um objeto URL com um endereço de base fornecido pelo *URL* especificado e com um caminho relativo fornecido pelo *String*. Você pode usar getDocumentBase ( ) para o URL da página que contém seu applet ou getCodeBase ( ) para o URL do arquivo de classe do applet. O caminho relativo será anexado ao endereço de base.
- U*RL(String, String, int, String)* cria um novo objeto URL, dado um protocolo (como http ou ftp), um nome de host (como www.prefect.com ou ftp. netcom.com), um número deporta (80 para http) eum nome de arquivo ou de caminho.
- U*RL(String, String, String)* é igual ao construtor anterior, menos o número de porta.

Quando você usa o construtor *URl(String),* deve lidar com objetos MalformedURLException. Você poderia usar um bloco try.. .catch, como apresentado a seguir:

```
try {
    theURL = new URL("http://www.mcp.com");
} catch (MalformedURLException e) {
    System.out.println("Bad URL: " + theURL);
}
```

Uma vez que você tenha um objeto URL, basta apenas passá-lo para o navegador; isso o faz carregar esse endereço:

```
getAppletContext( ).showDocument(theURL);
```

Então, o navegador que contém o applet Java com esse código carregará e apresentará o documento que está nesse URL.

A listagem 19.1 contém duas classes: ButtonLink e uma classe auxiliar chamada Bookmark. O applet ButtonLink apresenta três botões que representam importantes locais da Web; os botões estão ilustrados na Figura 19.1.

Dar um clique nos botões faz o documento ser carregado dos locais a que esses botões se referem.

Figura 19.1

O applet *ButtonLink.*

Nota

Esse applet deve ser executado dentro de um navegador para que os vínculos funcionem e foi escrito com técnicas de tratamento de eventos 1.02, para que possa ser executado na gama mais ampla de navegadores. Você recebe um aviso deprecated, quando o compila com a ferramenta javac da linguagem Java 2.

DIGITE **Listagem 19.1.** O texto completo de buttonlink. java.

```
1: import java.awt.*;
2: import java.net.*;
3:
4: public class ButtonLink extends java.applet.Applet {
5:     Bookmark bmList[ ] = new Bookmark[3];
6:
7:     public void init( ) {
8:         bmList[0] = new Bookmark("Sams Teach Yourself Java 1.2 in 21
           Days",
9:                 "http://www.prefect.com/java21");
10:        bmList[l] * new Bookmark("Macmillan Computer Publishing",
11:              "http://www.mcp.com");
12:        bmList[2] = new Bookmark("JavaSoft",
13:              "http://java.sun.com");
14:
15:        GridLayout gl = new GridLayout(bmList.length, 1, 10, 10);
16:        setLayout(gl);
17:        for (int i = 0; i < bmList.length; i++) {
18:             add(new Button(bmList[i].name));
19:        }
20:    }
21:
22:    public boolean action(Event evt, Object arg) {
23:        if (evt.target instanceof Button) {
24:            linkTo( (String)arg );
25:            return true;
26:        }
27:        ei se return false;
```

Listagem 19.1. Continuação

```
28:     }
29:
30:     void linkTo(String name) {
31:         URL theURL = null;
32:         for (int i = 0; i < bmList.length; i++) {
33:               if (name.equals(bmList[i].name))
34:                   theURL = bmList[i].uri;
35:         }
36:          if (theüRL != null)
37:              getAppletContext( ).showDocument(theURL);
38:     }
39: }
40:
41: class Bookmark {
42:     String name;
43:     URL uri;
44:
45:     Bookmark(String name, String theURL) {
46:         this.name = name;
47:          try {
48:              this.url = new URL(theURL);
49:          } catch (MalformedURLException e) {
50:              System.out.println("Bad URL: " + theURL);
51:          }
52:     }
53: }
```

Esse applet pode ser testado usando-se o seguinte código HTML:

```
<APPLET CODE="ButtonLink.class" HEIGHT=120 WIDTH=240>
</APPLET>
```

Duas classes compõem esse applet: A primeira, ButtonLink, implementa O applet em si; a segunda, Bookmark, é uma classe representando um marcador de página. Os marcadores de página possuem duas partes: um nome e um URL.

Esse applet em particular cria três instâncias de marcador de página (linhas 8 a 13) e as armazena em um array de marcadores de página. Esse applet poderia ser facilmente modificado para aceitar marcadores de página como parâmetros de um arquivo HTML. Para cada marcador de página, é criado um botão, cujo rótulo é o valor do nome do marcador.

O método linkTo( ) é chamado quando os botões são pressionados. 1 i nkTo ( ), definido nas linhas 30 a 38, extrai o nome do botão a partir do evento, utiliza-o para pesquisar o URL real do objeto Bookmark e, em seguida, diz ao navegador para que carregue o URL a que esse marcador de página faz referência.

Fazendo a comunicação entre applets

Às vezes, você quer ter uma página HTML que contenha vários applets diferentes. Para fazer isso, basta incluir várias iterações diferentes da tag <APPLET>. O navegador criará diferentes instâncias de seu applet para cada uma que aparecer na página HTML.

E se você quiser fazer a comunicação entre esses applets? E se quiser que uma alteração em um applet afetem os outros applets de alguma forma? A melhor maneira de se fazer isso é usar o contexto do applet para obter diferentes applets na mesma página.

Antes de trabalhar extensivamente com a comunicação entre applets, fique avisado de que o mecanismo descrito nesta seção é implementado de forma diferente (e freqüentemente falível) em diferentes navegadores e ambientes Java. Se você precisa contar com a comunicação entre applets para suas páginas da Web, certifique-se de testar esses applets extensivamente, em diferentes navegadores e em diferentes plataformas.

Como você aprendeu ao tratar com imagens gráficas, um *contexto* é um modo de descrever o ambiente de que algo faz parte. Um *contexto deapplet*, queé definido na classe AppletContext, é usado na comunicação entre applets.

Para obter uma instância dessa classe para seu applet, você usa o método getAppletContext( ), em vez de chamar um construtor de algum tipo. Você já viu o uso do método getAppletContext( ) para outras coisas; você também pode usá-lo para trabalhar com os outros applets da página.

Por exemplo, para chamar um método denominado sendMessage( ) em todos os applets de uma página, incluindo o atual, use o método getAppl ets ( ) e um loop for semelhante ao seguinte:

```
for (Enumeration e = getAppletContext( ).getApplets( );
    e.hasMoreElementsf );) {
    Applet current = (MyAppletSubclass)(e.nextElement( ));
    current.sendMessage( );
}
```

O método getAppl ets ( ) retorna um objeto Enumerati on com uma lista dos applets que estão presentes na página. A iteração através do objeto Enumeration dessa forma permite que você acesse cada elemento por sua vez na enumeração (Enumerati on). Note que cada elemento do objeto Enumerati on é uma instância da classe Object; para fazer esse applet se comportar do modo desejado (e aceitar mensagens de outros applets), você precisa fazer sua coerção para que seja uma instância da subclasse de seu applet (aqui, a classe MyAppletSubclass).

É um pouco mais complicado chamar um método em um applet específico. Para fazer isso, você atribui um nome aos seus applets e depois faz referência a eles pelo nome, dentro do corpo do código desse applet.

Para atribuir um nome a um applet, use o atributo NAME para <APPLET> em seu arquivo HTML:

```
<P>This applet sends information:
<APPLET CODE="MyApplet.c1ass" WIDTH=100 HEIGHT=150
NAME="sender">
</APPLET>
<P>This applet receives information from the sender:
<APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=150
NAME="receiver">
</APPLET>
```

Para obter uma referência a outro applet na mesma página, use o método getApplet( ) a partir do contexto de applet com o nome desse applet. Isso fornece uma referência ao applet com esse nome. Você pode, então, fazer referência a esse applet como se ele fosse apenas outro objeto: chamar métodos, definir suas variáveis de instância etc. Aqui está um código que faz exatamente isso:

```
// obtém o applet receptor
Applet receiver = (MyAppletSubclass) getAppletContext( ).
getApplet("receiver");
// diz para que ele se atualize.
receiver.update(text, value);
```

Nesse exemplo, você usa o método getAppl et ( ) para obter uma referência ao applet com o nome *receiver*. Observe que o objeto retornado pelo método getAppl et ( ) é uma instância da classe genérica Applet; provavelmente, você desejará fazer a coerção desse objeto em uma instância de sua subclasse. Dada a referência ao applet nomeado, você pode então chamar métodos nesse appíet como se ele fosse simplesmente outro objeto em seu próprio ambiente. Aqui, por exemplo, se os dois applets possuírem um método update( ), você pode dizer para que o receptor se atualize, usando as informações que o applet corrente possui.

Atribuir nomes aos seus applets e depois fazer referência a eles usando os métodos descritos nesta seção, permite que seus applets se comuniquem e estejam sincronizados entre si, fornecendo um comportamento uniforme para todos os applets presentes em sua página.

Recortar, copiar e colar

A partir da versão 1.1 da linguagem Java, foi inserido suporte às operações de recortar, copiar e colar entre componentes de uma interface com o usuário Abstract Windowing Toolkit e outros programas que estejam em execução na plataforma e que não sejam implementados em Java.

Anteriormente, no AWT, você só podia copiar e colar ciados entre os componentes que já tivessem essa funcionalidade incorporada nas plataformas nativas (por exemplo, texto podia ser copiado e colado entre campos de texto e áreas de texto). Isso foi ampliado para que outros dados ou objetos pudessem ser transferidos de um componente para outro.

Você precisa definir um objeto transferível para poder transferir dados de um componente para outro, depois precisa modificar ou criar componentes com a capacidade de transferir esse objeto.

As classes e interfaces para se fazer isso estão contidas no pacote java.awt.datatransfer.

Criando objetos transferíveis

NOVO TERMO Um *objeto transferível* é aquele que pode ser movido de um componente para outro usando-se o mecanismo de transferência de dados do AWT e que encapsula parte do conjunto de dados a ser transferido (texto formatado, por exemplo). Mais especificamente, um objeto transferível é aquele que implementa a interface Transferable.

Ao criar um objeto transferível, primeiramente você deve decidir que "versões" esse objeto vai suportar. Uma *versão* é basicamente o formato dos dados que estão sendo transferidos. Por exemplo, se você copiar texto formatado em HTML de um navegador e tentar colá-lo, esses dados poderiam ser colados como várias versões diferentes: texto formatado, texto puro ou como código HTML. As versões de dados determinam como o que está sendo copiado e o que está sendo colado negociam a transferência dos dados em si. Se a origem e o destino da transferência de dados não oferecerem suporte às mesmas versões, os dados não poderão ser transferidos.

As versões de dados são descritas usando-se tipos MIME, o mesmo mecanismo de negociação de conteúdo utilizado por muitos programas de correio eletrônico e pela própria World Wide Web. Se você não conhece os tipos MIME, o RFC 1521 contém a especificação MIME (a qual você deve encontrar em qualquer site da Web ou FTP que contenha os diversos documentos Internet RFC, sendo um exemplo o endereço http://ds.internic.net/rfc/rfcl521.txt). Além do nome lógico, a versão dos dados também possui um nome "legível para seres humanos", que pode ser traduzido para diferentes idiomas. As versões de dados também podem ter uma classe Java representativa — por exemplo, se a versão dos dados fosse um string Unicode, a classe Stri ng representaria essa versão. Se a versão não possui classe para representá-la, a classe InputStream será usada.

Para criar uma nova versão de dados, você gera uma nova instância da classe DataFlavor, usando um destes dois construtores:

- DataFlavor(Class, *String)* cria uma versão de dados que representa uma classe Java. O argumento *String* serve para a descrição da espécie legível para seres humanos. O objeto DataFlavor resultante terá um tipo MIME igual a appl ication/x-javaserial izedobject.
- DataFlavor(Sirzng, *String)* cria uma versão de dados que representa um tipo MIME, onde o primeiro argumento é o tipo MIME e o segundo é o string legível por seres humanos. A classe que representa essa versão será InputStream.

Com esse objeto de versão de dados, você pode consultar seus valores ou comparar tipos MIME com outros objetos de espécie de dados para negociar como os dados serão transferidos.

As versões de dados são usadas por objetos transferíveis, que são definidos usando-se a interface Transferable. Um objeto transferível incluirá os dados a serem transferidos e as instâncias de cada versão de dados que representa esse objeto. Você também precisa implementar os métodos getTransf er-DataFlavors( ), isDataF1avorSupported( ) egetTransferData( ),paraqueseu objeto transferível seja realmente negociado e transferido. (Consulte a interface Transferabl e para obter detalhes.)

A classe Stri ngSel ecti on implementa um objeto transferível simples para a transferência de strings de texto, usando objetos DataFlavor e a interface Transferable. Se você estiver especialmente interessado em copiar texto, StringSelection é um lugar excelente para começar (e, na verdade, pode ser o único objeto transferível de que você precisa).

Observe que os objetos transferíveis são utilizados para encapsular dados e para descrever seu formato; eles nada fazem para formatar esses dados. Isso é responsabilidade de seu programa, quando você utiliza a Área de Transferência para obter dados de uma origem.

Usando a Área de Transferência

Uma vez que tenha um objeto transferível, você pode usar uma Área de Transferência para transferir esse objeto entre componentes e de dentro da linguagem Java para a plataforma nativa. A linguagem Java 2 fornece um mecanismo de Área de Transferência fácil de utilizar, que permite colocar dados na Área de Transferência e recuperá-los dela. Você pode usar a Área de Transferência de sistema padrão para mover dados em outros programas que estejam sendo executados na plataforma nativa ou pode usar suas próprias instâncias da Área de Transferência para criar áreas de transferência especiais ou vários conjuntos especializados delas.

Na linguagem Java, as áreas de transferência são representadas pela classe Clipboard, também parte do pacote java.awt.datatransfer. Você pode obter o Cl i pboard de sistema padrão usando os métodos getTool ki t( ) e getSystem-ClipBoard( ) (como você aprendeu na última seção, getTool kit( ) fornece um modo de acessar vários recursos de sistema nativos), como segue:

```
Clipboard clip = getToolkit( ).getSystemClipboard( )
```

Uma observação importante para a Área de Transferência do sistema: os applets não podem acessar a Área de Transferência do sistema por questões de segurança (podem existir dados sigilosos contidos nessa Área de Transferência). Isso impede que os applets copiem e colem qualquer coisa na plataforma nativa. Você pode, entretanto, usar suas áreas de transferência internas para copiar e colar entre os componentes de um applet.

Qualquer componente que queira usar a Área de Transferência — tanto para colocar dados na Área de Transferência usando operações de recorte ou cópia como para obter dados dela usando a operação de colagem — deve implementar a interface ClipboardOwner. Essa interface possui um método: lostOwnershi p( ), que é chamado quando algum outro componente assume o controle da Área de Transferência.

Execute os seguintes passos para implementar as operações de recortar ou copiar:

1. Crie uma instância de seu objeto Transferable para armazenar os dados a serem copiados.
2. Crie uma instância do objeto que implementa a interface Cl i pboard-Owner (que pode ser a classe atual, ou o objeto Transferable também pode implementar uma Área de Transferência proprietária).
3. Se você estiver usando a Área de Transferência de sistema, use o método getSystemCl i pboard ( ) para obter uma referência a essa Área de Transferência.
4. Chame o método setContents( ) da Área de Transferência com o objeto transferível e com o objeto que implementa ClipboardOwner como argumentos. Com esse método, seu objeto tem a "posse" assegurada da Área de Transferência.
5. O método lostOwnership( ) é chamado quando outro objeto assume o controle da Área de Transferência. Você vai querer implementar esse método de modo que faça algo quando isso ocorrer (ou criar um método vazio, se não se preocupar com o fato de alguém ter substituído o conteúdo de sua Área de Transferência).

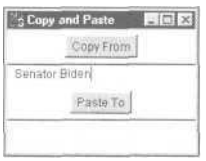
Complete os passos a seguir para implementar uma operação de colagem:

1. Use o método getContents( ) da Área de Transferência, que retorna um objeto transferível.
2. Use ométodo getTransferDataFlavors( ) do objeto transferível para descobrir a quais versões esse objeto oferece suporte. Determine a versão que vai usar.
3. Recupere os dados na versão correta, usando o método getTransf er-Data( ) do objeto transferível.

A seguir há um exemplo muito simples de um aplicativo que utiliza a Área de Transferência para copiar texto de um campo de texto para outro (ilustrado na Figura 19.2). A listagem 19.2 mostra o código desse exemplo.

Figura 19.2

O *applet CopyPoste.*



DIGITE

Listagem 19.2. O texto completo de copypaste.java.

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import java.awt.datatransfer.*;
4:
5: public class CopyPaste extends Frame
6:     implements ActionListener, ClipboardOwner {
7:
8:     Button copy, paste;
9:     TextField tfCopy, tfPaste;
10:    Clipboard clip;
11:
12:    public static void main(String[ ] arguments) {
13:        CopyPaste test = new CopyPaste( );
14:        WindowListener 1 = new WindowAdapter( ) {
15:            public void windowClosing(WindowEvent e) {
16:                System.exit(0);
17:            }
18:        };
19:        test.addWindowListener(l);
20:        test.setSize(200, 150);
21:        test.show( );
22:    }
23:
24:    CopyPaste( ) {
25:        super("Copy and Paste");
26:        clip = getToolkit( ).getSystemClipboard( );
27:        FlowLayout fio = new FlowLayout( );
28:        setLayout(flo);
29:
30:        copy = new Button("Copy From");
31:        tfCopy = new TextField(25);
32:        paste = new Button("Paste To");
33:        tfPaste = new TextField(25);
34:
35:        copy.addActionListener(this);
36:        paste.addActionListener(this);
37:        paste.setEnabled(false);
```

Listagem 19.2. Continuação

```
38:
39:         add(copy);
40:         add(tfCopy);
41:         add(paste);
42:         add(tfPaste);
43:     }
44:
45:     void doCopy( ) {
46:         if (tfCopy.getText( ) != null) {
47:             String txt = tfCopy.getText( );
48:             StringSeiection trans = new StringSelection(txt);
49:             clip.setContents(trans, this);
50:             paste.setEnabled(true);
51:         }
52:     }
53:
54:     void doPaste( ) {
55:         Transferable toPaste = clip.getContents(this);
56:         if (toPaste != null) {
57:             try {
58:                 String txt = (String)toPaste.getTransferData(
59:                     DataFlavor.stringFlavor);
60:                 tfPaste.setText(txt);
61:                 paste.setEnabled(false);
62:             } catch (Exception e) {
63:                 System.out.println("Error - " + e.toString( ));
64:             }
65:         }
66:     }
67:
68:     public void actionPerformed(ActionEvent e) {
69:         if (e.getSource( ) == copy)
70:             doCopy( );
71:         eise if (e.getSource( ) == paste)
72:             doPaste( );
73:     }
74:
75:     public void lostOwnership(Clipboard clip,
76:         Transferable contents) {
77:     }
78: }
```

Grande parte do código do aplicativo CopyPaste contém comportamento para o design de uma interface gráfica com o usuário e para responder a eventos de usuário. Você aprenderá a respeito dessas técnicas durante os próximos dois dias, quando desenvolver aplicativos Swing usando Java 2.

As instruções específicas para copiar e colar são as seguintes:

- Linha 3 A instrução import torna o pacote java.awt.datatransfer disponível.
- Linhas 5 e 6 A cláusula extends indica que essa classe implementará a interface Cl i pboardOwner.
- Linha 10 Uma referência é criada para um objeto Área de Transferência, chamado cl ip.
- Linha 26 O objeto Área de Transferência clip é associado à Área de Transferência de sistema, por uma chamada ao método getSystem-Clipboard( ).
- Linhas 45 a 52 O método doCopy( ) é chamado quando um usuário dá um clique no botão Copy From. Se houver algo no campo de texto Copy From, o texto será extraído do campo, na linha 47, usado para criar um novo objeto transferível, na linha 48, e colocado na Área de Transferência, na linha 49. Depois disso, o botão Paste To é desativado.
- Linhas 54 a 66 O método doPaste( ) é chamado, quando o botão Paste To recebe um clique de mouse. Um objeto Transf erabl e é usado para verificar o conteúdo da Área de Transferência. Se ela não estiver vazia, os dados são recuperados (linhas 58 e 59) e colocados no campo de texto Paste To (linha 60). Quando isso é feito, o botão Paste To é desativado.
- Linhas 75 a 77 Um método lost0wnership( ) é exigido, pois a classe implementa a interface Cl i pboardOwner. No entanto, nada é feito com relação às mudanças de posse; portanto, o método está vazio.

Rernote Method Invocation

O RMI (Remote Method Invocation) é usado para criar aplicativos Java que podem entrar em contato com outros aplicativos Java através de uma rede. Para sermos mais específicos, o RMI permite que um aplicativo chame métodos e acesse variáveis dentro de outro aplicativo (que pode estar em execução em diferentes ambientes Java ou em sistemas completamente diferentes) e troque objetos através de uma conexão de rede. O RMI é um mecanismo mais sofisticado para a comunicação entre objetos Java distribuídos do que seria uma simples conexão de socket, pois os mecanismos e protocolos através dos quais você se comunica entre objetos são definidos e padronizados. Você pode entrar em contato com outro programa Java usando RMI, sem a necessidade de saber antecipadamente qual é o protocolo utilizado ou como se faz para entrar em contato com ele.

Nota

Outra forma de comunicação entre objetos é chamada RPC (chamadas de procedure remotas), com que você pode chamar métodos ou executar procedures em outros programas, através de uma conexão de rede. Embora o RPC e o RMI tenham muito em comum, a principal diferença é que o RPC envia apenas chamadas de procedure por via eletrônica, com os argumentos passados ou descritos de tal modo que possam ser reconstruídos no outro lado. O RMI transmite realmente objetos inteiros via Net e, portanto, é mais conveniente para um modelo de objeto distribuído totalmente orientado a objetos.

Embora o conceito de RMI possa trazer a visão de objetos de todo o mundo alegremente se comunicando uns com os outros, o RMI é mais usado em um cenário cliente/servidor mais tradicional: um único aplicativo servidor recebe conexões e pedidos de vários clientes. O RMI é simplesmente o mecanismo pelo qual o cliente e o servidor se comunicam.

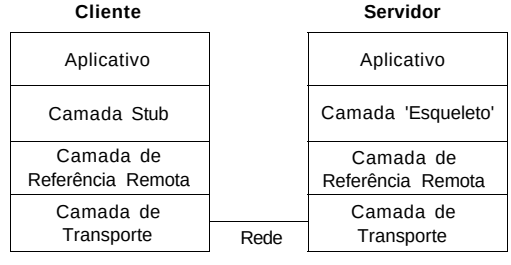
A arquitetura RMI

Os objetivos do RMI eram integrar um modelo de objeto distribuído na linguagem Java sem quebrá-la nem romper o modelo de objeto existente, além de tornar a interação com um objeto remoto tão fácil quanto a interação com um objeto local. Por exemplo, você deve ser capaz de usar objetos remotos exatamente da mesma maneira como utiliza objetos locais (atribuí-los a variáveis, passá-los como argumentos para métodos etc.) e a chamada de métodos em objetos remotos deve ser realizada da mesma maneira que as chamadas locais. Contudo, além disso, o RMI inclui mecanismos mais sofisticados para a chamada de métodos em objetos remotos, para passar objetos inteiros ou partes de objetos por referência ou por valor, assim como exceções adicionais para o tratamento de erros de rede que possam ocorrer enquanto uma operação remota está acontecendo.

O RMI possui várias camadas para atingir todos esses objetivos e uma simples chamada de método passa por várias dessas camadas para chegar ao seu destino (ver Figura 19.3). Na verdade, existem três camadas:

Figura 19.3

Camadas RMI.



- As camadas "stub" e "esqueleto" no cliente e no servidor, respectivamente. Essas camadas se comportam como objetos substitutos em cada lado, ocultando da implementação das classes a característica "remota" da chamada de objeto. Por exemplo, em seu aplicativo cliente, você pode chamar métodos remotos exatamente da mesma maneira que chama métodos locais; o objeto stub é um substituto local do objeto remoto.
- A camada de referência remota, que trata do empacotamento de uma chamada de método e de seus parâmetros e retorna valores para transporte através da rede.
- A camada de transporte, que representa a conexão de rede real de um sistema para outro.

Ter três camadas para RMI permite que cada uma seja controlada ou implementada independentemente. Os stubs e esqueletos permitem que as classes de cliente e servidor se comportem como se os objetos com que estão lidando fossem locais e utilizem exatamente os mesmos recursos da linguagem Java para acessar esses objetos. A camada de referência remota separa o processamento do objeto remoto em sua própria camada, que pode então ser otimizada ou novamente implementada, independentemente dos aplicativos que dependem dela. Finalmente, a camada de transporte de rede é usada independentemente das outras duas, de modo que você pode usar diferentes tipos de conexões de socket para RMI (TCP, UDP ou TCP com algum outro protocolo, como o SSL).

Quando um aplicativo cliente faz uma chamada de método remota, a chamada passa para o stub e depois para a camada de referência, que empacota os argumentos se necessário; em seguida, ela passa, através da camada de rede, para o servidor, onde, posteriormente, a referência no lado do servidor desempacota os argumentos e os transmite para o esqueleto e depois para a implementação do servidor. Os valores de retorno da chamada de método fazem então a viagem de volta para o lado do cliente.

NOVO TERMO O empacotamento e a passagem de argumentos de método são um dos aspectos mais interessantes do RMI, pois os objetos precisam ser convertidos em algo que possa ser passado via rede. Essa conversão é chamada de *serialização*. Desde que um objeto possa ser serializado, o RMI pode utilizá-lo como um parâmetro de método ou como um valor de retorno. Os objetos que podem ser serializados incluem todos os tipos primitivos Java, objetos Java remotos e quaisquer outros objetos que implementem a interface Serial izeabl e (que inclui muitas das classes do JDK padrão, como todos os componentes AWT).

Os objetos Java remotos utilizados como parâmetros de método ou valores de retorno são passados por referência, exatamente como aconteceria localmente. Outros objetos, entretanto, são copiados. Observe que esse comportamento afetará o modo como você escreve seus programas Java, quando eles utilizam chamadas de método remotas — você não pode, por exemplo, passar um array como argumento para um método remoto, fazer o objeto remoto mudar esse array e esperar que a cópia local seja modificada. Não é assim que os objetos locais se comportam, onde todos os objetos são passados como referências.

Criando aplicativos RMI

Para criar um aplicativo que use RMI, você utiliza as classes e interfaces definidas pelos pacotes java.rmi, que incluem o seguinte:

- java.activation Para a ativação de objeto remoto.
- java.rmi.server Para classes no lado do servidor.
- java.rmi.registry Contém as classes para localizar e registrar servidores RMI em um sistema local.
- java.rmi.dgc Para coletar lixo de objetos distribuídos.

O próprio pacote java.rmi contém as interfaces, classes e exceções RMI gerais.

Para implementar um aplicativo cliente-servidor baseado em RMI, primeiramente você define uma interface que contenha todos os métodos a que seu objeto remoto vai oferecer suporte. Todos os métodos dessa interface devem incluir uma instrução throws RemoteException, que tratará de problemas de rede em potencial; isso pode impedir que o cliente e o servidor se comuniquem.

O próximo passo é implementar a interface remota em um aplicativo no lado do servidor, o que normalmente amplia a classe UnicastRemoteObject. Dentro dessa classe, você implementa os métodos na interface remota e também pode criar e instalar um gerenciador de segurança para esse servidor (para impedir que clientes aleatórios estabeleçam conexão e façam chamadas de método não autorizadas). Você pode, é claro, configurar o gerenciador de segurança para ativar ou desativar várias operações. No aplicativo servidor, você também "registrará" o aplicativo remoto, o que o vinculará a um host e a uma porta.

No lado cliente, você implementa um aplicativo simples que usa a interface remota e chama métodos nessa interface. Uma classe Naming (em java.rmi) permite que o cliente estabeleça conexão de forma transparente com o servidor.

Com o código escrito, você pode compilá-lo usando o compilador Java padrão — mas há um outro passo: você usa o programa rmic para gerar as camadas de stubs e esqueletos, para que o RMI possa realmente trabalhar entre os dois lados do processo.

Finalmente, o programa rmiregistry é utilizado para estabelecer a conexão do aplicativo servidor com a própria rede e vinculá-lo a uma porta, para que as conexões remotas possam ser estabelecidas.

Isso, é claro, é uma tremenda simplificação do processo que você vai usar para criar um aplicativo baseado em RMI. Para obter mais informações sobre o RMI e como usar as classes RMI, um bom lugar para começar é com as informações do site da Web da Sun:

http://www.sur.com

Java Database Connectivitv

O JDBC (Java Database Connectivity) define uma interface estruturada para bancos de dados SQL (Structured Query Language), que representa a estratégia padrão do setor industrial para acessar bancos de dados relacionais. Oferecendo suporte a SQL, o JDBC permite que os desenvolvedores interajam e ofereçam suporte a uma grande variedade de bancos de dados. Isso significa que os detalhes específicos da plataforma de banco de dados subjacente são praticamente irrelevantes quando se trata do JDBC, o que é uma boa notícia para os desenvolvedores de Java.

Os bancos de dados SQL são bancos construídos no padrão SQL, que é amplamente aceito e define um protocolo restrito para o acesso e a manipulação de dados.

A estratégia da API JDBC para acessar bancos de dados SQL é comparável às técnicas existentes para o desenvolvimento de banco de dados, de modo que a interação com um banco de dados SQL usando o JDBC não é muito diferente do que a interação com um banco de dados SQL usando as ferramentas de banco de dados tradicionais. Isso deve dar aos programadores Java que já possuam alguma experiência com banco de dados a confiança de que eles podem fazer o que quiserem com o JDBC. A API JDBC já foi amplamente aprovada pelos líderes do setor, incluindo alguns fornecedores de ferramenta de desenvolvimento que anunciaram um suporte futuro para JDBC em seus produtos.

A API JDBC inclui classes para construções de banco de dados SQL comuns, como conexões de banco de dados, instruções SQL e conjuntos de resultado. Os programas Java JDBC possuem a capacidade de usar o conhecido modelo de programação SQL que executa instruções SQL e processa os dados resultantes. A API JDBC é amplamente dependente de um gerenciador de driver que ofereça suporte a múltiplas conexões de drivers com diferentes bancos de dados. Os drivers de banco de dados JDBC podem ser escritos inteiramente em Java ou implementados usando-se métodos nativos para ligar os aplicativos Java às bibliotecas de acesso a banco de dados já existentes.

O JDBC também inclui uma ponte entre ele e o ODBC, a interface comum da Microsoft para acessar bancos de dados SQL. A ponte JDBC-ODBC permite que drivers JDBC sejam usados como drivers ODBC.

As classes JDBC fazem parte da linguagem Java 2 como o pacote j ava. sql e incluem classes para gerenciar drivers, estabelecer conexões com bancos de dados, construir consultas SQL e manipular os resultados.

O site da Web da Sun contém muitas informações e a especificação relativas ao JDBC.

Resumo

Quando a versão 1 da linguagem Java foi lançada, em 1995, ela era mais conveniente à programação de applets do que ao design de aplicativos maiores.

Isso não é mais verdade desde a linguagem Java 2, agora que ela oferece suporte robusto a recursos como design de componente de software, Remote Method Invocation, conectividade de banco de dados e serialização de objetos.

O material abordado hoje é um trampolim para uma maior exploração dos pacotes e classes que oferecem suporte a esses recursos. Quando você tiver dominado os fundamentos da linguagem Java, estará pronto para abordar esses assuntos avançados em livros, cursos e em sua própria exploração da API Java.

Nesse sentido, os próximos dois dias irão coroar sua introdução à linguagem Java. Você vai aprender a criar interfaces gráficas com o usuário usando o Swing, a nova solução de janelas oferecida na linguagem Java 2, e a transformar essas interfaces em aplicativos funcionais.

Perguntas e respostas

P showStatus( ) não funciona no meu navegador. Como posso fornecer aos meus leitores informações de status?

R O fato de um navegador oferecer suporte a showStatus( ) depende desse navegador. Se você precisa ter comportamento do tipo status em seu applet, pense na criação de um rótulo de status no próprio applet, que seja atualizado com as informações que precisam ser apresentadas.

P Tentei fazer a comunicação entre dois applets em minha página da Web usando os métodos getAppletContext( ) e getApplet( ). Meus applets falham com erros NullPointerException. O que significa isso?

R O mecanismo descritopara a comunicação entre applets é como a Sun e a biblioteca de classe Java dizem que ela deve funcionar.

Entretanto, assim como showStatus( ), se um navegador implementa ou não esse mecanismo, ou se ele está implementado corretamente, depende desse navegador. As versões do Netscape antes da 3.0 e do Internet Explorer possuem problemas estranhos com relação à comunicação entre applets.

SEMANA 3

Dia 20

Criando uma interface com o usuário com Swing

Durante os dois últimos dias deste livro, você trabalhará com um conjunto de classes chamado Swing, que pode implementar um estilo de interface com o usuário chamado Metal.

Parece que alguém da Sun adora música ou é um músico frustrado. A próxima tecnologia que vai emprestar o nome de um gênero musical deverá se chamar Ska, Hip-hop ou Beer Barrei Polka.

O Swing, que faz parte da biblioteca Java Foundation Classes, é uma extensão do Abstract Windowing Toolkit que foi integrada à linguagem Java 2.

O Swing oferece uma funcionalidade bastante aprimorada em relação ao seu predecessor — novos componentes, recursos de componente expandidos, melhor tratamento de eventos e aparência e comportamento selecionáveis.

Hoje você usará o Swing para criar interfaces Java e amanhã aprenderá a transformar essas interfaces em programas completos.

Nota

O Swing também está disponível como um complemento separado para a linguagem Java 1.1. Se você estiver escrevendo um applet ou aplicativo usando a linguagem Java 1.1, em vez de 2, pode usar as classes Swing, carregando por download a versão compatível com 1.1 a partir das Java Foundation Classes, da JavaSoft. Atualmente elas estão disponíveis na seguinte página:

http://java.sun.com/products/jfc/

Como o Swing foi totalmente incorporado à linguagem Java 2, você não precisa fazer o download dele separadamente, caso esteja usando essa versão do JDK.

As vantagens do Swing

"Aparência e comportamento" é uma expressão freqüentemente utilizada ao se descrever programação de interface. O significado é auto-explicativo — ele descreve a aparência de uma interface gráfica com o usuário e como ela se comporta com um usuário.

Aparência e comportamento constituem algo que se tornou relevante na linguagem Java com a introdução do Swing, as classes de janelas incluídas em Java 2. Você trabalhará com o Swing o tempo todo, durante os dois últimos dias deste livro.

O Swing permite que um programa Java utilize uma aparência e um comportamento diferentes no controle do programa ou mesmo para o usuário de um programa.

Esse recurso oferece a mudança mais significativa em relação ao AWT. O Swing permite que você crie um programa Java com uma interface que utilize o estilo do sistema operacional nativo — como o Windows ou o Solaris — ou um novo estilo exclusivo da linguagem Java, que foi denominado Metal.

Os componentes do Swing, ao contrário de seus predecessores nas versões anteriores da linguagem Java, são implementados inteiramente em Java. Isso significa uma melhor compatibilidade entre diferentes plataformas do que a que se consegue com programas que você criou usando o Abstract Windowing Toolkit.

Todos os elementos do Swing fazem parte do pacote java.awt.swing. Para usar uma classe Swing, você precisa utilizar uma instrução import com essa classe ou uma instrução abrangente, como a seguinte:

```
import java.awt.swing.*;
```

O processo de usar um componente do Swing não é diferente do que acontecia para os componentes do Abstract Windowing Toolkit. Você cria o componente chamando seu método construtor, chamando métodos dos componentes, caso eles sejam necessários para uma configuração correta, e incluindo o componente em um contêiner.

O Swing usa a mesma infra-estrutura de classes que o Abstract Windowing Toolkit, o que torna possível misturar componentes do Swing e do AWT na mesma interface. No entanto, em alguns casos, os dois tipos de componentes não serão renderizados corretamente em um contêiner. Para evitar esses problemas, é melhor usar um sistema de janelas ou outro exclusivamente, em um programa.

Todos os componentes do Swing são subclasses da classe Jcomponent.

Uma estrutura de aplicativo

O primeiro passo na criação de um aplicativo Swing simples é gerar uma classe que seja uma subclasse de JFrame. A classe JFrame é uma extensão de Frame e pode ser usada de maneira semelhante.

O código da listagem 20.1 pode ser uma estrutura para todos os aplicativos que você criar que utilizem uma janela principal.

DIGITE **Listagem 20.1.** O texto completo de framework.java.

```
1:  import java.awt.GridLayout;
2:  import java.awt.event.*;
3:  import java.awt.swing.*;
4:
5:  public class Framework extends JFrame {
6:
7:       public Framework( ) {
8:            super("Application Title");
9:
10:          // Inclui componentes aqui
11:     }
12:
13:     public static void main(String[ ] args) {
14:          JFrame frame = new Framework( );
15:
16:          WindowListener 1 - new WindowAdapter( ) {
17:               public void windowClosing(WindowEvent e) {
18:                    System.exit(0);
19:               }
20:          };
21:          frame.addwindowListener(l);
22:
23:          frame.pack( );
24:          frame.setVisible(true);
25:     }
26: }
```

Esse aplicativo é uma subclasse de JFrame e todo o trabalho envolvido na criação da interface com o usuário do quadro é realizado no método construtor Framework( ).

A linha 8 do construtor fornece o texto para a barra de título do quadro, usando o método *super (String)*. A interface com o usuário deve ser construída dentro desse construtor — componentes podem ser incluídos nos containers aqui, e os containers podem ser incluídos no quadro.

Ocorre o seguinte no método main( ) do aplicativo:

- Linha 14 O construtor Framework ( ) é usado para criar uma nova instância da classe J Frame. Essa instância é a janela principal do aplicativo.

- Linhas 16 a 21 Esse código de tratamento de eventos padrão cuida da tarefa de fechar o aplicativo quando o quadro for fechado. Amanhã você aprenderá sobre as rotinas de tratamento de eventos, como a WindowListener.
- Linha 23 Chama o método pack( ) do quadro para ser redimensionado com o menor tamanho possível para conter todos os seus componentes. Usando pack( ), você pode incluir componentes no quadro e saber que haverá espaço para eles.
- Linha 24 Torna o quadro visível, usando seu método setVisible(*boolean*). Se o argumento fosse false em vez de true, o quadro se tornaria invisível.

Embora você possa compilar essa estrutura sem problemas, ela não produzirá nada de útil — a janela do quadro se abrirá com um tamanho mínimo e você nem mesmo poderá ver a barra de título inteira no quadro.

Os componentes devem ser incluídos antes que esse comece a parecer um aplicativo real.

Adicionando componentes a um quadro Swing

NOVO TERMO

Trabalhar com um objeto JFrame é mais complicado do que trabalhar com seu correspondente AWT. Em vez de incluir contêineres e componentes diretamente no quadro, você deve inclui-los em um conteiner intermediário, chamado *painel de conteúdo*.

Um objeto JFrame é subdividido em vários painéis diferentes. O painel principal com que você trabalha é o painel de conteúdo, que representa a área total de um quadro, ao qual os componentes podem ser adicionados.

Para adicionar um componente em um painel de conteúdo, faça o seguinte:

- Crie um objeto JPanel (a versão Swing de um painel).
- Adicione todos os componentes (que podem ser contêineres) ao JPanel, usando seu método add(*Component*).
- Transforme esse JPanel no painel de conteúdo, usando o método setContentPane(Contaíner). O objeto JPanel deve ser o único argumento.

O programa que aparece na listagem 20.2 usa a estrutura de aplicativo e insere um botão no painel de conteúdo do quadro. O botão é criado a partir da classe JButton, a versão Swing do botão que pode receber um clique de mouse. Esse programa é semelhante ao projeto SIacker do Dia 11.

DIGITE

Listagem 20.2. O texto completo de swinger.java.

```
1:  import java.awt.GridLayout;
2:  import java.awt.event.*;
3:  import java.awt.swing.*;
4:
5:  public class Swinger extends JFrame {
6:
7:      public Swinger( ) {
8:          super("Swinger");
9:
10:         String note = "I receive a disproportionate amount of " +
11:             "joy from being clicked. Please interact with me.";
12:         JButton hotButton = new JButton(note);
13:
14:         JPanel pane • new JPanel( );
15:         pane.add(hotButton);
16:
17:         setContentPane(pane);
18:     }
19:
20:     public static void main(String[ ] args) {
21:         JFrame frame = new Swinger( );
22:
23:         WindowListener 1 = new WindowAdapter( ) {
24:             public void windowClosing(WindowEvent e) {
25:                 System.exit(0);
26:             }
27:         };
28:         frame.addwindowListener(l);
29:
30:         frame.pack( );
31:         frame.setVisible(true);
32:     }
33: }
```

A Figura 20.1 mostra a saída desse aplicativo.

Figura 20.1
O *aplicativo Swinger*.

O único material novo na listagem 20.2 são as linhas 10 a 17, nas quais acontece o seguinte:

- Linhas 12 e 13 Um objeto JButton é criado usando-se um string como rótulo. Essa utilização é idêntica à dos construtores da classe Button.

- Linhas 14 e 15 Um objeto JPanel é criado e o botão é adicionado a esse painel.
- Linha 17 O método setContentPane(Container) é utilizado para transformar o painel no painel de conteúdo do quadro.

Uma vez que tenha definido o painel de conteúdo para um quadro, você usa métodos como setLayout*(LayoutManager)* e add*(Component)*, nesse painel. Você não chama esses métodos no próprio quadro.

Isso também vale para os applets, que são implementados no Swing através da classe JApplet. Você deve criar um objeto JPanel, adicionar-lhe componentes e transformar esse painel no painel de conteúdo do applet. Observe que todos os applets que você criar com o Swing e com a linguagem Java 2 não funcionarão nos navegadores da Web que ofereçam suporte apenas a Java 1.02. Você também precisa encontrar um modo de tornar as classes Swing disponíveis para o applet; elas demorarão bastante tempo para serem carregadas por download na página da Web, com os arquivos que compõem o applet.

Trabalhando com o Swing

Existem componentes Swing para cada um dos componentes do Abstract Windowing Toolkit, sobre os quais você aprendeu até agora. Na maioria dos casos, existe um construtor para o componente Swing que combina com seu correspondente AWT; portanto, você não precisa aprender nada de novo para trabalhar com os componentes do Swing.

Também existem novos construtores para muitos componentes que recebem um objeto Icon como argumento. Um *ícone* (icon) é uma pequena imagem gráfica, normalmente em formato GIF, que pode ser usada em um botão, rótulo ou outro elemento de interface, como um modo visual para identificar o componente que representa. Você vê ícones o tempo todo em pastas de arquivo de sistemas operacionais gráficos, como o Windows 95 e o MacOS.

Um objeto Icon é criado de maneira semelhante a um objeto Image. O construtor recebe o nome de arquivo de uma imagem gráfica ou o URL como único argumento. O exemplo a seguir carrega um ícone do arquivo unabom.gif e cria um JButton com o ícone como rótulo. A Figura 20.2 mostra o resultado.

```
ImageIcon una = new ImageIcon("unabom.gif");
JButton button = new JButton(una);

JPanel pane = new JPanel( );
pane.add(button);

setContentPane(pane);
```

Figura 20.2
Um ícone em um *JButton.*

Nota

O ícone Unabomber é proveniente da coleção Pardon My Icons!, de Jeffrey Zeldsman, a qual inclui centenas de ícones que você pode usar em seus próprios projetos. Se você estiver procurando ícones para trabalhar com aplicativos Swing, pode encontrar o Pardon My Icons em seu site na World Wide Web, no seguinte endereço:

http://www.zeldman.com

O código-fonte desse exemplo está disponível no site da Web do livro, no endereço http://www.prefect.com/java21, sob o nome de arquivo UnaButton.java, e a imagem gráfica do ícone está no arquivo unabom.gif.

Rótulos

Os rótulos são implementados no Swing com a classe J Label. A funcionalidade é comparável à dos rótulos AWT, mas agora você pode incluir ícones. Além disso, o alinhamento de um rótulo pode ser especificado com uma de três variáveis de classe da classe SwingConstants: LEFT, CENTER ou RIGHT.

Alguns métodos construtores que você pode usar incluem os seguintes:

- JLabel *(String, int)* Um rótulo com texto e alinhamento especificados.
- JLabel *(String, Icon, int)* Um rótulo com texto, ícone e alinhamento especificados.

Botões

Como você aprendeu, os botões Swing são incorporados pela classe JButton. Eles podem apresentar um rótulo de texto, exatamente como os botões AWT, um rótulo de ícone ou uma combinação de ambos.

Alguns métodos construtores que você pode usar incluem os seguintes:

- JButton *[String)* Um botão com o texto especificado.
- JButton *(Icon)* Um botão com o ícone especificado.
- JButton *(String, Icon)* Um botão com o texto e o ícone especificados.

Compos de texto

Os campos de texto são implementados no Swing com a classe JTextField. Uma diferença entre esses campos de texto e os seus correspondentes AWT é que o método setEchoChar*(char)* não é suportado em JTextField para usar a fim de ocultar entrada de texto.

Os métodos construtores que você pode usar incluem os seguintes:

- JTextField(int) Um campo de texto com a largura especificada.
- JTextField *{String, int)* Um campo de texto com o texto e a largura especificados.

A classe JPasswordField é usada para se criar um campo de texto que possa utilizar um caractere para ocultar entrada. Essa classe possui os mesmos métodos construtores que JTextField: JPasswordField(iní) e JPasswordField *(String, int).*

Uma vez que tenha criado um campo de texto de senha, você pode usar nele o método setEchoChar(char) para ocultar entrada com o caractere especificado.

Áreas de texto

As áreas de texto são implementadas no Swing com a classe JTextArea. Ela recebe os seguintes métodos construtores:

- JTextArea(int, *int)* Uma área de texto com o número especificado de linhas e colunas.
- JTextArea *(String, int, int)* Uma área de texto com texto, linhas e colunas especificadas.

Caixas de seleção e botões de rádio

A classe JCheckBox é a implementação das caixas de seleção no Swing. A funcionalidade é a mesma do AWT, com a inclusão de rótulos de ícone.

Os métodos construtores que você pode usar incluem os seguintes:

- JCheckBox *(String)* Uma caixa de seleção com o rótulo de texto especificado.
- JCheckBox(String, *boolean)* Uma caixa de seleção com o rótulo de texto especificado, que será selecionada se o segundo argumento for true.
- JCheckBox(Jcon) Uma caixa de seleção com o rótulo de ícone especificado.
- JCheckBox(icon, *boolean)* Uma caixa de seleção com o rótulo de ícone especificado, que será selecionada se o segundo argumento for true.
- JCheckBox(String, *Icon)* Uma caixa de seleção com o rótulo de texto e o rótulo de ícone especificados.
- JCheckBox *(String, Icon, boolean)* Uma caixa de seleção com o rótulo de texto e o rótulo de ícone especificados, que será selecionada se o segundo argumento for true.

Os grupos de caixa de seleção são implementados no Swing com a classe ButtonGroup. Conforme você já viu, apenas um componente de um grupo de caixas de seleção pode ser selecionado por vez. Você cria um objeto ButtonGroup e insere caixas de seleção nele, usando o método add *(Component)* para incluir um componente no grupo.

Os botões de rádio são implementados no Swing através da classe JRadioButton. Os métodos construtores são os iguais aos da classe JCheckBox.

A mudança do nome de CheckboxGroup para ButtonGroup reflete a funcionalidade expandida — os botões e botões de rádio também podem ser agrupados.

Listas de escolha

As listas de escolha, que foram criadas no AWT usando-se a classe Choice, representam uma das implementações possíveis com a classe JComboBox.

Uma lista de escolha é criada nas seguintes etapas:

1. O construtor JComboBox ( ) é usado sem nenhum argumento.
2. O método addltem(Object) da caixa de combinação é usado para adicionar itens à lista.
3. O método setEditable*(boolean)* da caixa de combinação é usado com o argumento false.

Este último método transforma a caixa de combinação em uma lista de escolha — as únicas escolhas que um usuário pode fazer são aqueles itens adicionados à lista.

Se a caixa de combinação permite a edição, o usuário pode introduzir texto no campo, como uma alternativa ao uso da lista de escolha para selecionar um item. Essa é a combinação que dá nome às caixas de combinação.

Barras de *rolag*em

As barras de rolagem são implementadas no Swing com a classe JScrollBar. A funcionalidade é idêntica à das barras de rolagem do AWT e você pode usar os seguintes métodos construtores:

- JScrollBar(int) Uma barra de rolagem com a orientação especificada.
- JScrollBar(int, *int, int, int, int)* Uma barra de rolagem com a orientação, valor inicial, tamanho da caixa de rolagem, valor mínimo. e valor máximo especificados.

A orientação é indicada com a variável de classe SwingConstants HORIZONTAL ou VERTICAL.

Um exemplo: o aplicativo SwinaColorTest

Um dos projetos do Dia 14 foi o applet ColorTest, que permitia que uma cor fosse selecionada, usando valores sRGB ou HSB.

O próximo projeto cria uma interface gráfica com o usuário para esse projeto usando Swing e o transforma em uma aplicativo e não em um applet. Amanhã você criará os métodos de tratamento de eventos para ele.

DIGITE **Listagem 20.3.** O texto completo de swingcolortest.java.

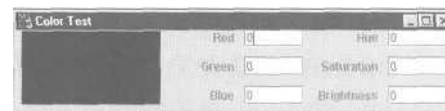
```
1: import java.awt.*;
2: import java.awt.event.*;
3: import java.awt.swing.*;
4:
5: public class SwingColorTest extends JFrame {
6:     SwingColorControls RGBcontrols, HSBcontrols;
7:     JPanei swatch;
8:
9:     public SwingColorTest( ) {
10:         super("Color Test");
11:
12:         JPanel pane = new JPanel( );
13:        pane.setLayout(new GridLayout(l, 3, 5, 15));
14:         swatch = new JPanel( );
15:         swatch.setBackground(Color.black);
16:         RGBcontrols = new SwingColorControls(this, "Red",
17:             "Green", "Blue");
18:         HSBcontrols = new SwingColorControls(this, "Hue",
19:            "Saturation", "Brightness");
20:         pane.add(swatch);
21:         pane.add(RGBcontrols);
22:         pane.add(HSBcontrols);
23:
24:         setContentPane(pane);
25:   }
26:
27:   public static void main(String[ ] args) {
28:         JFrame frame = new SwingColorTest( );
29:
30:         WindowListener 1 = new WindowAdapter( ) {
31:             public void windowClosing(WindowEvent e) {
32:                 System.exit(0);
33:             }
34:        };
35:        frame.addWindowListener(l);
36:
37:        frame.pack( );
38:        frame.setVisible(true);
39:    }
```

Listagem 20.3. Continuação

```
40:
41:     public Insets getInsets( ) {
42:          return new Insets(10, 10, 10, 10);
43:     }
44: }
45:
46: class SwingColorControls extends JPanel {
47:       SwingColorTest frame;
48:       JTextField tfieldl, tfield2, tfield3;
49:
50:       SwingColorControls(SwingColorTest parent,
51:            String 11, String 12, String 13) {
52:
53:            frame = parent;
54:             setLayout(new GridLayout(3,2,10,10));
55:            tfieldl = new JTextField("O");
56:            tfield2 = new JTextField("0");
57:            tfield3 = new JTextField("O");
58:            add(new JLabel(ll, JLabel .RIGHT));
59:             add(tfieldl);
60:           add(new JLabel(12, JLabel.RIGHT));
61:           add(tfield2);
62:           add(new JLabel(13, JLabel.RIGHT));
63:           add(tfield3);
64:       }
65:
66:       public Insets getInsets( ) {
67:            return new Insets(10, 10, 0, 0);
68:       }
69: }
```


A Figura 20.3 mostra a interface desenvolvida para este aplicativo. Embora os botões e outros componentes exibam aparência e comportamento diferentes do applet ColorTest, sobre o que você aprenderá na seção "Definindo a aparência e o comportamento" de hoje, o restante da interface funciona de forma igual ao seu correspondente não-Swing.

Figuro 20.3
O aplicativo SwingColorTest.



O programa SwingColorTest usa a estrutura de aplicativo apresentada anteriormente; portanto, muitas partes do programa foram apresentadas antes.

Esse programa é composto de três classes: a classe principal SwingCo-1 orTest, a classe auxiliar privativa SwingColorControl s e a classe interna que está definida nas linhas 30 a 34.

Tanto a classe Swi ngCol orTest como a classe Swi ngCol orControl s anulam o método getlnsets( ), o que permite que esses componentes sejam inseridos *(inset)* até um número designado de pixels, a partir das bordas de seu contêiner. Assim como em muitos aspectos do Swing, isso é suportado da mesma maneira que acontece para os componentes AWT.

A classe SwingColorControl s é uma subclasse de JPanel. Essa classe foi atualizada para o Swing, mudando os campos de texto e os rótulos dos componentes AWT para componentes do Swing. Nenhuma outra mudança foi necessária.

Na classe Swi ngCol orTest, as seguintes alterações atualizaram o código de janelas para trabalhar com o Swing, em vez do AWT:

- O programa é uma subclasse de JFrame.
- O objeto swatch, que apresenta a cor selecionada no momento, torna-se um objeto JPanel, em vez de Canvas. Não existe objeto Canvas no Swing; portanto, painéis devem ser usados em seu lugar.
- Um objeto JPanel é criado para ser o painel de conteúdo principal do quadro.
- Os componentes swatch, RGBcontrols e HSBcontrols são incluídos no painel de conteúdo, em vez de na janela principal do programa.

Em muitos casos, uma interface criada para o Abstract Windowing Toolkit pode ser implementada usando-se o Swing, com poucas alterações importantes. Se você estiver convertendo um applet escrito para Java 1.02 em um programa Swing escrito para Java 2, encontrará alterações mais significativas nos métodos de tratamento de eventos, que serão discutidos amanhã.

Novos recursos do Swing

Além dos componentes e contêineres que ampliam a funcionalidade do Abstract Windowing Toolkit, o Swing oferece numerosos recursos inteiramente novos, incluindo aparência e comportamento que podem ser definidos, nemônicos de teclado, dicas descritivas e caixas de diálogo padrão.

Definindo a aparência e o comportamento

Você já está familiarizado com o conceito de gerenciadores de layout — classes que controlam a organização dos componentes em uma interface com o usuário.

O Swing possui um gerenciador de interface com o usuário que controla a aparência e o comportamento dos componentes — o modo como os botões, rótulos e outros elementos são renderizados na tela.

O gerenciamento da aparência e do comportamento é manipulado pela classe UIManager do pacote javax.swing.*. As opções de aparência e comportamento variam dependendo do ambiente de desenvolvimento Java que você estiver usando. O seguinte está disponível com o Java 2:

- Aparência e comportamento do Windows 95 ou do Windows NT
- Aparência e comportamento do sistema Motif X-Window
- Metal, nova aparência e comportamento multiplataforma do Swing

A classe UIManager possui um método setLookAndFeel *(LookAndFeel)* que é usado para se escolher a aparência e o comportamento de um programa.

Para obter um objeto LookAndFeel que você possa usar com setLookAndFeel ( ), utilize um dos seguintes métodos UIManager:

- getCrossPlatformLookAndFeelClasslName( ) Esse método retorna um objeto LookAndFeel representando a aparência e o comportamento multiplataforma Metal da linguagem Java.
- getSystemLookAndFeelClassName( ) Esse método retorna um objeto LookAndFeel representando a aparência e o comportamento de seu sistema.

O método setLookAndFeel ( ) levanta uma exceção UnsupportedLookAndFeel Exception, se não puder definir a aparência e o comportamento.

As instruções a seguir podem ser utilizadas em qualquer programa para designar o Metal como aparência e comportamento:

```
try {
    UIManager.setLookAndFeel(
     UIManager.getCrossPlatformLookAndFeelClassName( ));
    } catch (Exception e) {
     System.err.println("Can't set look and feel: " + e);
}
```

Para selecionar a aparência e o comportamento de seu sistema, use getSystemLookAndFeelClassName( ), que está dentro da chamada do método setLookAndFeel ( ) no exemplo anterior. Isso produz resultados diferentes em diferentes sistemas operacionais. Um usuário de Windows 95 obteria uma aparência e comportamento do Windows 95 usando getSystemLookAndFeel - ClassName( ). Um usuário de UNIX obteria a aparência e o comportamento do Motif.

Mnemônicos de teclado

NOVO TERMO

Um *mnemônico de teclado,* também chamado de *acelerador de tecla, é* uma seqüência de teclado que pode ser usada para controlar um componente de interface com o usuário. Esses mnemônicos proporcionam um meio de se usar um programa sem o mouse e fazem parte do suporte do Swing

à acessibilidade — novas classes que tornam mais fácil para cegos e outros usuários com deficiências variadas para que executem um programa Java.

Os mnemônicos de teclado simulam uma ação de mouse quando são utilizados, e o modo de usar um mnemônico varia de acordo com a plataforma que está sendo empregada. Em um computador executando o Windows 95, um mnemônico de teclado está disponível mantendo-se a tecla Alt pressionada em combinação com outra tecla.

Os mnemônicos de teclado são definidos chamando-se o método setMnemonic(char) no componente em que o mnemônico pode ser usado para controlar. O argumento *char* é a tecla que deve ser utilizada como parte do mnemônico. O exemplo a seguir cria um objeto JButton e associa o caractere ' i' ao botão:

```
JButton infoButton = new JButton("Information");
infoButton.setMnemonic('i');
```

Pressionar Alt + I faz com que o componente infoButton seja acionado.

Balões de descrição

Outro modo de tornar um programa mais amigável para o usuário é associar balões de descrição a componentes em uma interface. Talvez você já esteja familiarizado com os balões de descrição — legendas textuais que aparecem em alguns programas, se seu mouse permanecer por alguns segundos sobre um componente.

Os balões de descrição são usados para descrever o objetivo do componente. Quando você está aprendendo a usar um programa pela primeira vez, os balões de descrição representam um excelente recurso de aprendizado, caso tenham sido implementados.

Para definir um balão de descrição para um componente, chame o método setToolTipText(Sírzng) do componente. O string deve ser uma descrição concisa do objetivo do componente.

O exemplo a seguir cria um componente JScrol 1 Bar e associa um balão de descrição a ele.

```
JScrollBar speed = new JScrollBar( );
speed.setToolTipText("Move to set animation speed");
```

O texto do balão de descrição pode ter apenas uma linha de comprimento; portanto, você não pode usar o caractere de newline (' \n ') para dividir o texto em várias linhas.

Descrições e nomes de componente

Outro modo de tornar uma interface mais acessível é fornecer uma descrição textual de um componente Swing. Isso é feito com um processo em duas etapas:

1. Obtenha o objeto AccessibleContext associado ao componente, chamando o método getAccessibleContext( ) no componente.
2. Chame o método setAccessibleDescription(String) nesse objeto AccessibleContext. O argumento de string deve ser a descrição textual do componente.

Por exemplo, o trecho de código a seguir define a descrição de um objeto JButton:

```
JButton quit = new JButton("Quit");
quit.getAccessibleContext( ).setAccessibleDescription(
"When you click this button, the program terminates.");
```

O método setAccessibleName(String) funciona da mesma maneira que setAccessibleDescription(String). Ele pode ser usado para atribuir um nome ao componente, que descreva sucintamente o seu objetivo. "Quit Button" seria um nome apropriado para o objeto quit, no exemplo anterior. O próximo exemplo define o nome de um campo de texto chamado nm para "Name Field":

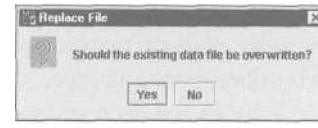
```
JTextField nm = new JTextField( );
nm.getAccessibleContext( ).setAccessibleName("Name Fiel d");
```

Caixas de diálogo padrão

A classe JOptionPane oferece vários métodos que podem ser usados para se criar caixas de diálogo padrão: pequenas janelas que fazem uma pergunta, avisam o usuário ou fornecem uma breve mensagem importante. A Figura 20.4 mostra um exemplo de caixa de diálogo com aparência e comportamento Metal.

Figura 20.4

Uma caixa de diálogo padrão.



Sem dúvida, você já viu caixas de diálogo desse tipo — quando seu sistema falha, uma caixa padrão aparece e divulga a má notícia. Além disso, quando você exclui arquivos, uma caixa de diálogo pode ser usada para garantir que realmente se deseja fazer isso. Essas janelas representam um modo eficiente de se comunicar com um usuário sem a sobrecarga de criar uma nova classe para representar a janela, inserir componentes nela e escrever métodos de tratamento de eventos para receber entrada. Tudo isso é manipulado automaticamente, quando uma das caixas de diálogo padrão oferecidas por JOptionPane é usada.

Existem quatro caixas de diálogo padrão:

- ConfirmDialog Uma caixa de diálogo que faz uma pergunta, com botões para respostas Yes, No e Cancel.
- InputDialog Uma caixa de diálogo que solicita entrada de texto.

- MessageDialog Uma caixa de diálogo que apresenta uma mensagem.
- OptionDialog Uma caixa de diálogo que compreende todos os outros três tipos de caixa de diálogo.

Cada uma dessas caixas de diálogo possui seu próprio método na classe JOptionPane.

Caixas de diálogo de confirmação

O modo mais fácil de criar uma caixa de diálogo do tipo Yes/No/Cancel é com a seguinte chamada de método: showConfirmDiaiog(Component, *Object).* O argumento relativo ao componente especifica o container que deve ser considerado a origem da caixa de diálogo, e essa informação é usada para se determinar onde na tela a janela de diálogo deve ser apresentada. Se nul1 for usado, em vez de um contêiner, ou se o container não for um objeto Frame, a caixa de diálogo ficará centralizada na tela.

O segundo argumento pode ser um string, um componente ou um objeto Icon. Se for um string, esse texto será apresentado na caixa de diálogo. Se for um componente ou um ícone, esse objeto será apresentado no lugar de uma mensagem de texto.

Esse método retorna um de três valores inteiros possíveis e cada um deles é uma variável de classe de JOpti onPane: YES_OPTION, NO_OPTION e CANCEL_OPTION.

A seguir está um exemplo que utiliza uma caixa de diálogo de confirmação com uma mensagem de texto e armazena a resposta na variável response:

```
int response;
response = JOptionPane.showConfirmDialog(nul l,
   "Should I delete all of your irreplaceable personal files");
```

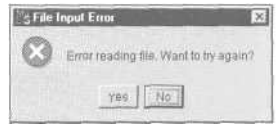
Outro método oferece mais opções para o diálogo de confirmação: showConfirmDiai*og{Component, Object, String, int, int).* Os dois primeiros argumentos são os iguais aos do outro método showConfirmDialog ( ). Os três últimos argumentos são os seguintes:

- Um string que será apresentado na barra de título da caixa de diálogo.
- Um inteiro que indica quais botões de opção serão apresentados. Ele deve ser igual às variáveis de classe YES_NO_CANCEL_OPTION ou YES_NO_OPTION.
- Um inteiro que descreve o tipo dessa caixa de diálogo, usando as variáveis de classe ERROR_MESSAGE, INFORMATION_MESSAGE, PLAIN_ MESSAGE, QUESTION_MESSAGE ou WARNING_MESSAGE. Esse argumento é usado para determinar qual ícone será desenhado na caixa de diálogo junto com a mensagem.

```
int response = JOptionPane.showConfirmDialog(null,
               "Error reading file. Want to try again?",
               "File Input Error",
               JOptionPane.YES_NO_OPTION,
               JOptionPane.ERROR_MESSAGE);
```

A Figura 20.5 mostra a caixa de diálogo resultante, com a aparência e o comportamento do Windows.

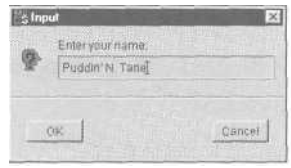
Figura 20.5
Uma caixa de diálogo de confirmação.



Caixas de diálogo de entrada

Uma caixa de diálogo de entrada faz uma pergunta e usa um campo de texto para armazenar a resposta. A Figura 20.6 mostra um exemplo usando a aparência e o comportamento do Motif.

Figura 20.6
uma *caixa de dialogo de entrada.*



O modo mais fácil de criar uma caixa de diálogo de entrada é com uma chamada ao método showInputDialog*(Component, Object)*. Os argumentos são o componente de origem e o string, componente ou ícone a ser apresentado na caixa.

A chamada de método da caixa de diálogo de entrada retorna um string que representa a resposta do usuário. A instrução a seguir cria a caixa de diálogo de entrada ilustrada na Figura 20.6:

```
String response = J0ptionPane.showInputDialog(nu11,
   "Enter your name:");
```

Você também pode criar uma caixa de diálogo de entrada com o método showInputDialog(Component, *Object, String, int)*. Os dois primeiros argumentos são os mesmos da chamada de método mais curta e os dois últimos representam o seguinte:

- O título a ser apresentado na barra de título da caixa de diálogo.
- Uma das cinco variáveis de classe que descrevem o tipo da caixa de diálogo: ERROR_MESSAGE, INFORMATION_MESSAGE, PLAIN_MESSAGE, QUESTION_MESSAGE ou WARNING_MESSAGE.

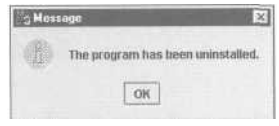
A instrução a seguir cria uma caixa de diálogo de entrada usando esse método:

```
String response = JOptionPane.showInputDialog(null,
   "What is your ZIP code?",
   "Enter ZIP Code",
   JOptionPane.QUESTION_MESSAGE);
```

Caixas de diálogo de mensagem

Uma caixa de diálogo de mensagem é uma janela simples que apresenta informações. A Figura 20.7 mostra um exemplo usando a aparência e o comportamento Metal.

Figura 20.7
Uma caixa de *diálogo de mensagem.*



Uma caixa de diálogo de mensagem pode ser criada com uma chamada ao método showMessageDialog(Component, *Object).* Assim como acontece nas outras caixas de diálogo, os argumentos são o componente de origem e o string, componente ou ícone a ser apresentado.

Diferente das outras, a caixa de diálogo de mensagem não retorna nenhum tipo de valor de resposta. A instrução a seguir cria o diálogo de mensagem ilustrado na Figura 20.7:

```
JOptionPane.showMessageDialog(null,
   "The program has been uninstalled.");
```

Você também pode criar uma caixa de diálogo de entrada de mensagem com o método showMessageDialog *{Component, Object, String, int).* A utilização é idêntica ao método showInputDialog( ) com os mesmos argumentos, exceto que showMessageDialog( ) não retorna um valor.

A instrução a seguir cria uma caixa de diálogo de mensagem usando esse método:

```
String response = JOptionPane.showMessageDialog(null,
   "An asteroid has destroyed the Earth.",
   "Asteroid Destruction Alert",
   JOptionPane.WARNING_MESSAGE);
```

Caixas de diálogo de opção

A mais complexa das caixas de diálogo é a caixa de diálogo de opção, que combina os recursos de todas as outras. Ela pode ser criada com o método showOptionDialog(Component, *Object, String, int, int, Icon, Object[], Object).*

Os argumentos desse método são os seguintes:

- O componente de origem da caixa de diálogo.
- O texto, ícone ou componente a ser apresentado.

- Um string a ser apresentado na barra de título.
- O tipo de caixa, usando as variáveis de classe YES_N0_0PTI0N, YES_N0_CANCEL_0PTI0N ou a literal 0, se outros botões vão ser usados no lugar desses.
- O ícone a ser apresentado, usando as variáveis de classe ERROR_MESSAGE, INFORMATION_MESSAGE, PLAIN_MESSAGE, QUESTION_MESSAGE, WARNING_MESSAGE ou a literal 0, se nenhuma dessas deve ser usada.
- Um objeto Icon a ser apresentado em lugar de um dos ícones do argumento anterior.
- Um array de objetos que contêm os componentes ou outros objetos que representem as escolhas na caixa de diálogo, caso YES_N0_0PTI0N e YES_N0_CANCEL_0PTI0N não estejam sendo usados.
- O objeto representando a seleção padrão, se as opções YES_N0_0PTI0N e YES_NO_CANCEL não estiverem sendo usadas.

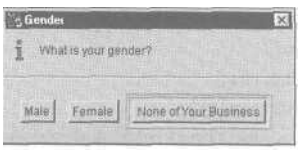
Os dois últimos argumentos permitem que você crie uma ampla gama de escolhas para a caixa de diálogo. Você pode criar um array de botões, rótulos, campos de texto ou mesmo uma mistura de diferentes componentes como um array de objetos. Esses componentes são apresentados usando-se o gerenciador de layout de fluxo — não existe um modo de especificar um gerenciador diferente dentro da caixa de diálogo.

O exemplo a seguir cria uma caixa de diálogo de opção que utiliza um array de objetos JButton para as opções da caixa e o elemento gender[2] como a seleção padrão:

```
JButton[ ] gender = new JButton[3];
gender[0] = new JButton("Male");
gender[l] = new JButton("Female");
gender[2] = new JButton("None of Your Business");
int response = JOptionPane.showOptionDialog(null,
   "What is your gender?",
   "Gender",
   0,
   JOptionPane.INF0RMATI0N_MESSAGE,
   null,
   gender,
   gender[2]);
```

A Figura 20.8 mostra a caixa de diálogo resultante, com a aparência e o comportamento do Motif.

Figura 20.8
Uma caixa de diálogo de opção.



Um exemplo: o aplicativo Info

O próximo projeto proporciona uma chance de ver uma série de caixas de diálogo em um programa em funcionamento. O aplicativo Info utiliza caixas de diálogo para obter informações do usuário, que são então colocadas em campos de texto na janela principal do aplicativo.

Introduza a listagem 20.4 e compile o resultado.

DIGITE

Listagem 20.4. O texto completo de info.java.

```
1: import java.awt.GridLayout;
2: import java.awt.event.*;
3: import java.awt.swing.*;
4:
5: public class Info extends JFrame {
6:     private JLabel titleLabel = new JLabel("Title: ",
7:         SwingConstants.RIGHT);
8:     private JTextField title;
9:     private JLabel addressLabel = new JLabel("Address: ",
10:        SwingConstants.RIGHT);
11:    private JTextField address;
12:    private JLabel typeLabel = new JLabel("Type: ",
13:        SwingConstants.RIGHT);
14:    private JTextField type;
15:
16:    public Info( ) {
17:        super("Site Information");
18:
19:        // Nome do site
20:        String responsei = JOptionPane.showInputDialog(null,
21:            "Enter the site title:");
22:        title = new JTextField(responsel, 20);
23:
24:        // Endereço do site
25:        String response2 = JOptionPane.showInputDialog(null,
26:            "Enter the site address:");
27:        address = new JTextField(response2, 20);
28:
29:       // Tipo do site
30:       String[ ] choices = { "Personal", "Commercial", "Unknown" };
31:        int response3 « J0ptionPane.show0ptionDialog(null,
32:            "What type of site is it?",
33:            "Site Type",
34:           0,
35:            JOptionPane.QUESTIONMESSAGE,
36:            null,
37:            choices,
38:            choices[0]);
39:       type = new JTextField(choices[response3], 20);
```

Listagem 20.4. Continuação

```
40:
41:        JPanel pane = new JPanel( );
42:        pane.setLayout(new GridLayout(3, 2));
43:        pane.add(titleLabel);
44:        pane.add(title);
45:        pane.add(addressLabel);
46:        pane.add(address);
47:        pane.add(typeLabel);
48:        pane.add(type);
49:
50:        setContentPane(pane);
51:    }
52:
53:    public static void main(String[ ] args) {
54:        try {
55:            UIManager.setLookAndFeel(
56:                UIManager.getSystemLookAndFeelClassName( ));
57:        } catch (Exception e) {
58:            System.err.println("Couldn't use the System "
59:                               + "look and feel: " + e);
60:        }
61:
62:        JFrame frame = new Info( );
63:
64:        WindowListener l = new WindowAdapter( ) {
65:            public void windowClosing(WindowEvent e) {
66:                System.exit(0);
67:            }
68:        };
69:        frame.addWindowListener(l);
70:
71:        frame.pack( );
72:        frame.setVisible(true);
73:    }
74: }
```


A Figura 20.9 mostra a janela principal desse aplicativo com a aparência e o comportamento do sistema (Windows, neste instantâneo de tela). Três campos de texto possuem valores fornecidos por caixas de diálogo.

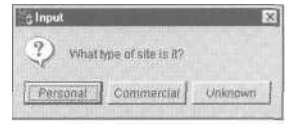
Figura 20.9
A janela principal do aplicativo Info.

A maior parte desse aplicativo é código padronizado que pode ser usado com qualquer aplicativo Swing. As linhas a seguir estão relacionadas às caixas de diálogo:

- Linhas 19 a 22 Uma caixa de diálogo de entrada é usada para pedir ao usuário para que introduza um título de site. Esse título é usado no construtor de um objeto JTextField, que coloca o título no campo de texto.
- Linhas 24 a 27 Uma caixa de diálogo de entrada semelhante à que é usada para solicitar um endereço de site, que é utilizado no construtor de outro objeto JTextField.
- Linha 30 Um array de objetos String chamado choices é criado e três elementos recebem valores.
- Linhas 31 a 38 Uma caixa de diálogo de opção é usada para solicitar o tipo de site. O array choices é o sétimo argumento, que define três botões na caixa de diálogo com os strings do array: Personal, Commercial e Unknown. O último argumento, choices[0], designa o primeiro elemento do array como a seleção padrão da caixa de diálogo. A Figura 20.10 mostra essa caixa de diálogo de opção.
- Linha 39 A resposta à caixa de diálogo de opção, um inteiro identificando o elemento do array que foi selecionado, é armazenada em um componente JTextField chamado type.

Figura 20.10

A *caixa de diálogo de* opção de tipo *de site.*



Resumo

Depois que centenas de milhares de programadores tiveram a chance de usar as primeiras versões da linguagem Java, uma das principais reclamações era com relação ao Abstract Windowing Toolkit. Embora ele permitisse a criação de uma interface funcional, existiam alguns problemas para se fazer todas as interfaces funcionarem em diferentes plataformas e alguns elementos de uma interface gráfica com o usuário não eram suportados pelo AWT.

O Swing é uma resposta efetiva às críticas, oferecendo um sistema de janelas sofisticado que é conveniente para muitos tipos diferentes de programas Java. Se você der uma olhada na documentação incluída com o Swing, encontrará mais de 30 componentes diferentes.

Amanhã, você verá a transformação de uma interface em um aplicativo completo.

Perguntas e respostas

P **Um aplicativo pode ser criado sem o Swing?**

R Certamente. O Swing é apenas uma expansão do Abstract Windowing Toolkit e você pode continuar a usar o AWT para aplicativos com a linguagem Java 2. Você precisará usar as técnicas de tratamento de eventos abordadas amanhã, em vez daquelas disponíveis para applets Java 1.02. Se você deve criar um aplicativo sem o Swing é outra questão. Não há comparação entre os recursos do Swing e aqueles oferecidos pelo AWT. Com o Swing, você pode usar muito mais componentes e controlá-los de maneiras mais sofisticadas.

SEMANA 3

Dia 21

Tratando de eventos de usuário com Swing

Existem dois grandes eventos neste capítulo: a conclusão de sua viagem de três semanas pela linguagem de programação Java e os eventos que você aprende a tratar em programas Swing.

Para transformar uma interface Java em um programa Java que funcione, você precisa tornar a interface receptiva aos eventos de usuário.

Você já tratou de eventos antes, aprendendo a manipular cliques de mouse e outras entradas de usuário com o Abstract Windowing Toolkit. Esse conhecimento foi usado para criar applets compatíveis com Java 1.02.

NOVO TERMO O Swing trata os eventos de forma diferente, pela utilização de um conjunto de classes chamadas *receptores de evento*. Hoje, você aprenderá a adicionar receptores de todos os tipos em seus programas Swing, incluindo aqueles que manipulam eventos de ação, eventos de mouse e outras interações.

Quando terminar, você vai celebrar o evento concluindo um aplicativo Java inteiro, usando o conjunto de classes Swing.

O evento principal

No sistema de tratamento de eventos sobre o qual você aprendeu na última semana, os eventos eram tratados com um conjunto de métodos que estão disponíveis para todos os componentes. Métodos como mouseDown( ), keyDown( ) e action( ) podiam ser anulados por qualquer programa AWT que quisesse tratar desses eventos.

Esse sistema de tratamento de eventos se aplica apenas a Java 1.02, pois uma solução bastante aprimorada para eventos foi oferecida nas versões subseqüentes da linguagem.

Você usará o novo sistema para criar aplicativos Swing.

Receptores de evento

Se uma classe quer responder a um evento de usuário no sistema de tratamento de eventos Java 1.2, deve implementar a interface que trata dos eventos. Essas interfaces são chamadas receptores de evento.

Cada receptor trata de um tipo específico de evento e uma classe pode implementar quantos deles forem necessários.

Os seguintes receptores de evento estão disponíveis:

- ActionListener Eventos de ação, que são gerados por um usuário agindo sobre um componente, como um clique de mouse em um botão
- AdjustmentListener Eventos de ajuste, que são gerados por um componente que está sendo ajustado, como quando uma barra de rolagem é movida
- FocusListener Eventos de foco de teclado, que são gerados quando um componente, como um campo de texto, ganha ou perde o foco
- ItemListener Eventos de item, que são gerados quando um item, como uma caixa de seleção, foi alterado
- KeyListener Eventos de teclado, que ocorrem quando um usuário digita texto no teclado
- MouseListener Eventos de mouse, que são gerados por cliques de mouse, um mouse entrando na área de um componente e um mouse deixando a área de um componente
- MouseMotionListener Eventos de movimento de mouse, que são usados para controlar toda a movimentação de um mouse sobre um componente
- WindowListener Eventos de janela, que são gerados por uma janela, como a janela principal do aplicativo ao ser maximizada, minimizada, movida ou fechada

A classe a seguir é declarada de modo a poder tratar de eventos de ação e de texto:

```
public class Suspense extends JFrame implements ActionListener,
     TextListener {
    // ...
}
```

O pacote java. awt. event contém todos os receptores de evento básicos, bem como os objetos que representam eventos específicos. Para usar essas classes em seus programas, você pode importá-las individualmente ou utilizar uma instrução como a seguinte:

```
import java.awt.event.*;
```

Definindo componentes

Tornando uma classe um receptor de evento, você define um tipo específico de evento a ser captado por essa classe. Isso nunca acontecerá, se você não der prosseguimento a um segundo passo: um receptor correspondente deve ser incluído no componente. Esse receptor gerará os eventos quando o componente for usado.

Depois que um componente for criado, você pode chamar um dos seguintes métodos no componente, para associar um receptor a ele:

- addActionListener( ) Componentes Jbutton, JCheckBox, JComboBox, JTextField e JRadioButton.
- addAdjustmentListener( ) Componentes JScroil Bar.
- addFocusListener( ) Todos os componentes do Swing.
- addItemListener( ) Componentes Jbutton, JCheckBox, JcomboBox e JRadioButton.
- addKeylistener( ) Todos os componentes do Swing.
- addMouseListener( ) Todos os componentes do Swing.
- addMouseMotionListener( ) Todos os componentes do Swing.
- addWindowListener( ) Todos os componentes JWindow e JFrame.

Modificar um componente depois de incluí-lo em um contêiner é um erro fácil de cometer em um programa Java. Você deve inserir receptores em um componente e tratar de qualquer outra configuração antes que ele seja incluído em quaisquer containers, senão esses ajustes serão ignorados quando o programa for executado.

O exemplo a seguir cria um objeto JButton e associa um receptor de evento de ação a ele:

```
JButton zap = new JButton("Zap");
zap.addActionListener(this);
```

Todos os diferentes métodos de adição recebem um argumento: o objeto que está recebendo eventos desse tipo. A utilização de this indica que a classe atual é o receptor de evento. Você poderia especificar um objeto diferente, desde que sua classe implementasse a interface receptora correta.

Métodos de tratamento de eventos

Quando você associa uma interface a uma classe, esta classe deve tratar de todos os métodos contidos na interface.

No caso dos receptores de evento, cada um dos métodos é chamado automaticamente pelo sistema de janelas, quando o evento de usuário correspondente ocorre.

A interface ActionListener possui apenas um método: actionPerformed( ). Todas as classes que implementam ActionListener devem ter um método com uma estrutura como a seguinte:

```
public void actionPerformed(ActionEvent evt) {
    // trata do evento aqui
}
```

Se apenas um componente na interface gráfica com o usuário de seu programa possui um receptor para eventos de ação, esse método actionPerformed( ) pode ser utilizado para responder a um evento gerado por esse componente.

Se mais de um componente possui um receptor de evento de ação, você precisa usar o método para descobrir qual deles foi usado e, então, agir de acordo em seu programa.

No método actionPerformed( ), você pode ter notado que um objeto ActionEvent é enviado como argumento, quando o método é chamado. Esse objeto pode ser usado para se descobrir detalhes a respeito do componente que gerou o evento.

ActionEvent e todos os outros objetos de evento fazem parte do pacote java.awt.event, e eles são subclasses da classe EventObject.

A todo método de tratamento de eventos é enviado um objeto de evento de algum tipo. O método getSource( ) do objeto pode ser usado para se determinar o componente que enviou o evento, como no exemplo a seguir:

```
public void actionPerformed(ActionEvent evt) {
    Object src = evt.getSource( );
}
```

O objeto retornado pelo método getSource( ) pode ser comparado com os componentes, usando-se o operador ==. As instruções a seguir poderiam ser usadas dentro do exemplo de actionPerformed( ) anterior:

```
if (src == quitButton)
    quitProgram( );
else if (src == sortRecords)
    sortRecords( );
```

Esse exemplo chama o método quitProgram( ) se o objeto quitButton gerou o evento, ou o método sortRecords( ), se o botão sortRecords gerou o evento.

Muitos métodos de tratamento de eventos chamarão um método diferente para cada tipo diferente de evento ou componente. Isso torna o método de tratamento de eventos mais fácil de ler. Além disso, se houver mais de um método de tratamento de eventos em uma classe, cada um poderá chamar os mesmos métodos para concluir o trabalho.

Outra técnica útil dentro de um método de tratamento de eventos é usar a palavra-chave instanceof para verificar o tipo de componente que gerou o evento. O exemplo a seguir poderia ser usado em um programa com um botão e um campo de texto, cada um dos quais gerando um evento de ação:

```
void actionPerformed(ActionEvent evt) {
    Object src = evt.getSource( );
    if (src instanceof JTextField)
      calculateScore( );
    else if (src instanceof JButton)
      quitProgramf );
}
```

O programa da listagem 21.1 usa a estrutura de aplicativo para criar um objeto JFrame e inserir componentes nele. O próprio programa apresenta dois componentes JButton que são usados para mudar o texto na barra de título do quadro.

DIGITE

Listagem 21.1. O texto completo de changetitle.java.

```
1: import java.awt.event.*;
2: import java.awt.swing.*;
3: import java.awt.*;
4:
5: public class ChangeTitle extends JFrame implements ActionListener {
6:     JButton b1 = new JButton("Rosencrantz");
7:     JButton b2 • new JButton("Guildenstern");
8:
9:     public ChangeTitle( ) {
10:        super("Title Bar");
11:
12:        bl.addActionListener(this);
13:        b2.addActionListener(this);
14:        JPanel pane = new JPanel( );
15:        pane.add(bl);
16:        pane.add(b2);
17:
```

Listagem 21.1. Continuação

```
18:          setContentPane(pane);
19:      }
20:
21:      public static void main(String[ ] args) {
22:          JFrame frame = new ChangeTitle( );
23:
24:          WindowListener 1 = new WindowAdapter( ) {
25:              public void windowClosing(WindowEvent e) {
26:                  System.exit(0);
27:              }
28:          };
29:          frame.addWindowListener(l);
30:
31:          frame.pack( );
32:          frame.setVisible(true);
33:      }
34:
35:      public void actionPerformed(ActionEvent evt) {
36:          Object source = evt.getSource( );
37:          if (source == b1)
38:              setTitle("Rosencrantz");
39:      else       if (source -• b2)
40:              setTitle("Guildenstern");
41:          repaint( );
42:      }
43: }
```

Esse aplicativo está ilustrado na Figura 21.1.

Figura 21.1
O *aplicativo ChangeTitle.*

Apenas 12 linhas foram necessárias para responder aos eventos de ação nesse aplicativo:

- A linha 1 importa o pacote java.awt.event.
- As linhas 12 e 13 adicionam os receptores de ação aos dois objetos JButton.
- As linhas 35 a 42 respondem aos eventos de ação que ocorrem nos dois objetos JButton. O método getSource( ) do objeto evt é usado para determinar a origem do evento. Se ele for igual ao botão bl, o título do quadro será definido como Rosencrantz; se ele for igual a b2, o título será definido como Guildenstern. Uma chamada a repaint( ) é necessária para que o quadro seja redesenhado depois de qualquer alteração de título que possa ter ocorrido no método.

Trabalhando com métodos

As seções a seguir detalham a estrutura de cada método de tratamento de eventos e os métodos que podem ser usados dentro deles.

Além dos métodos descritos, o método getSource( ) pode ser utilizado para determinar o objeto que gerou o evento.

Eventos de ação

Os eventos de ação ocorrem quando um usuário conclui uma ação utilizando um dos seguintes componentes: Jbutton, JCheckBox, JComboBox, JTextField ou JRadioButton.

Uma classe precisa implementar a interface ActionListener para tratar desses eventos. Além disso, o método addActionLi stener( ) precisa ser chamado em cada componente que deve gerar um evento de ação — a não ser que você queira ignorar os eventos de ação desse componente.

Existe apenas um método na interface ActionListener: actionPer-*formed(ActionEvent).* Ele tem a seguinte forma:

```
public void actionPerfortmed(ActionEvent evt) {
    // •••
}
```

Além do método getSource( ), você pode usar o método getActionCommand ( ) no objeto Act i onEvent, para descobrir mais informações sobre a origem do evento.

Por definição, o comando de ação é o texto associado ao componente, como o rótulo em um JButton. Você também pode definir um comando de ação diferente para um componente, chamando seu método setActionCommand *{String).* O argumento de string deve ser o texto desejado do comando de ação.

Por exemplo, as instruções a seguir criam um JButton e um JTextFi ei d e atribuem a ambos o comando de ação "Sort Files":

```
JButton sort = new JButton("Sort");
JTextField name = new JTextField ( );
sort.setActionCommand("Sort Files");
name.setActionCommand("Sort Files");
```

Os comandos de ação se tornam excepcionalmente úteis quando você está escrevendo um programa em que mais de um componente deve fazer a mesma coisa acontecer. Um exemplo disso seria um programa com um botão Quit e uma opção Quit em um menu suspenso. Atribuindo aos dois componentes o mesmo comando de ação, você pode tratar deles com o mesmo código em um método de tratamento de eventos.

Eventos de ajuste

Os eventos de ajuste ocorrem quando um componente JScrol 1 Bar (para barras de rolagem) é movido com o uso das setas da barra, a caixa ou com um clique de mouse em qualquer parte da barra. Para tratar desses eventos, uma classe precisa implementar a interface AdjustmentListener.

Existe apenas um método na interface AdjustmentListener: adjustmentValueChanged(/lGÍjustmeflt£VeA)t). Ele tem a seguinte forma:

```
public void adjustmentValueChanged(AdjustmentEvent evt) {
    // ...
}
```

Para ver qual é o valor atual do componente JScrollBar dentro desse método de tratamento de eventos, o método getValue( ) pode ser chamado no objeto AdjustmentEvent. Esse método retorna um inteiro representando o valor da barra de rolagem.

Você também pode determinar o modo como o usuário moveu a barra de rolagem, usando o método getAdjustmentType( ) do objeto AdjustmentEvent. Isso retorna um de cinco valores, cada um dos quais sendo uma variável da classe Adjustment:

- UNIT_INCREMENT Um aumento de 1 no valor, que pode ser causado por um clique de mouse em uma seta da barra de rolagem ou pelo uso de uma tecla de cursor.
- UNIT_DECREMENT Uma diminuição de 1 no valor.
- BLOCK_INCREMENT Um aumento maior no valor, causado por um clique de mouse na barra de rolagem, na área entre a caixa e a seta.
- BLOCK_DECREMENT Uma diminuição maior no valor.
- TRACK Uma mudança no valor, causada pela movimentação da caixa.

O programa que aparece na listagem 21.2 ilustra a utilização da interface AdjustmentListener. Uma barra de rolagem e um campo de texto sem capacidade de edição são incluídos em um quadro, e mensagens são apresentadas no campo, quando a barra de rolagem é movida.

DIGITE **Listagem 21.2.** O texto completo de welladjusted. java.

```
1: import java.awt.event.*;
2: import java.awt.swing.*;
3: import java.awt.*;
4:
5: public class WellAdjusted extends JFrame implements AdjustmentListener
  {
6:     BorderLayout bord = new BorderLayout( );
7:     JTextField value = new JTextField( );
8:     JScrollBar bar = new JScrollBar(SwingConstants.HORIZONTAL,
```

Listagem 21.2. Continuação

```
9:            50,  10,  0,  100);
10:
11:     public WellAdjusted( ) {
12:          super("Well  Adjusted");
13:
14:          bar.addAdjustmentListener(this);
15:     value.setHori       zontalAlignment(Swi ngConstants.CENTER);
16:          vaiue.setEditable(false);
17:          JPanel  pane  =  new  JPanel( );
18:          pane.setLayout(bord);
19:          pane.add(value,  "South");
20:          pane.add(bar,  "Center");
21:
22:          setContentPane(pane);
23:      }
24:
25:      public static void main(String[ ] args) {
26:           JFrame frame = new WellAdjusted( );
27:
28:          WindowListener 1 = new WindowAdapter( ) {
29:               public void windowClosing(WindowEvent e) {
30:                    System.exit(0);
31:               }
32:           };
33:           frame.addWindowListener(l);
34:
35:           frame.pack( );
36:           frame.setVisible(true);
37:      }
38:
39:      public void adjustmentValueChanged(AdjustmentEvent evt) {
40:           Object source = evt.getSource( );
41:           if (source == bar) {
42:                int newValue = bar.getValue( );
43:               value.setTextC" + newValue);
44:           }
45:           repaint( );
46:      }
47: }
```

A Figura 21.2 mostra uma captura de tela do aplicativo.

Figura 21.1

A *saída do aplicativo WellAdjusted.*

NOVO TERMO Você pode estar se perguntando por que existe um conjunto de aspas vazias na chamada de setText ( ), na linha 43 desse programa. As aspas vazias são chamadas *de string null* e isso é concatenado ao inteiro newValue para transformar o argumento em um string. Como você deve se lembrar, se um string e um item que não é um string são concatenados, a linguagem Java sempre trata o resultado como um string. O string nul 1 é um atalho quando você quer apresentar algo que ainda não é um string.

Eventos de foco

NOVO TERMO Os *eventos de foco* ocorrem quando qualquer componente ganha ou perde o foco de entrada em uma interface gráfica com o usuário. O foco descreve o componente que está ativo no momento para entrada via teclado. Se um dos campos possuir o foco (em uma interface com o usuário com vários campos de texto que possam ser editados), haverá um cursor intermitente nesse campo. Todo texto digitado será colocado nesse componente.

O foco se aplica a todos os componentes que podem receber entrada. Em um objeto JButton, um contorno pontilhado aparecerá no botão que possui o foco.

Para tratar de eventos de foco, uma classe precisa implementar a interface FocusListener. Existem dois métodos na interface: focusGained(Focus£Ve/7t) e *focusLost(FocusEvent).* Elas têm a seguinte forma:

```
public void focusGained(FocusEvent evt) {
    // •••
}

public void focusLost(FocusEvent evt) {
    // …
}
```

Para se determinar o objeto que ganhou ou perdeu o foco, o método getSource( ) pode ser chamado no objeto FocusEvent enviado como argumento para os métodos focusGained( )e focusLost( ).

Eventos de item

NOVO TERMO *os eventos de item* ocorrem quando um item é selecionado ou perde a seleção em qualquer um dos seguintes componentes: Jbutton, JCheckBox, JComboBox ou JRadioButton. Uma classe precisa implementar a interface ItemListener para tratar desses eventos.

```
void itemStateChanged(ItemEvent evt) {
    // ...
}
```

Para se determinar o item em que o evento ocorreu, o método getltem( ) pode ser chamado no objeto ItemEvent.

Você também pode determinar se o item foi selecionado ou perdeu a seleção, usando o método getStateChange( ). Esse método retorna um inteiro que será igual à variável de classe ItemEvent. DESELECTED ou ItemEvent. SELECTED.

A utilização de eventos de item está ilustrada na listagem 21.3. O aplicativo SelectItem.apresenta a escolha de uma caixa de combinação em um campo de texto.

DIGITE

Listagem 21.3. O texto completo de selectitem.java.

```
1: import java.awt.event.*;
2: import java.awt.swing.*;
3: import java.awt.*;
4:
5: public class SelectItem extends JFrame implements ItemListener {
6:      BorderLayout bord = new BorderLayout( );
7:      JTextField result = new JTextField(27);
8:      JComboBox pick = new JComboBox( );
9:
10:     public Selectltem( ) {
11:         super("Select Item");
12:
13:         pick.addltemListener(this);
14:         pick.addItem("Navigator");
15:         pick.addltem("lnternet Explorer");
16:         pick.addItem("Opera");
17:         pick.setEditable(false);
18:         result.setHorizontalAli gnment(Swi ngConstants.CENTER);
19:         result.setEditable(false);
20:         JPanel pane = new JPanel( );
21:         pane.setLayout(bord);
22:         pane.add(result, "South");
23:         pane.add(pick, "Center");
24:
25:         setContentPane(pane);
26:     }
27:
28:     public static void main(String[ ] args) {
29:         JFrame frame = new Selectltem( );
30:
31:         WindowListener 1 = new WindowAdapter( ) {
32:             public void windowClosing(WindowEvent e) {
33:                 System.exit(0);
34:             }
```

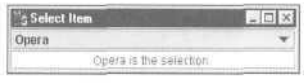
Listagem 21.3. Continuação

```
35:         };
36:         frame.addWindowListener(l);
37:
38:         frame.pack( );
39:         frame.setVisible(true);
40:     }
41:
42:     public void itemStateChanged(ItemEvent evt) {
43:         Object source = evt.getSource( );
44:         if (source == pick) {
45:             Object newPick = evt.getItem( );
46:             result.setText(newPick.toString( ) + " is the selection.");
47:         }
48:         repaint( );
49:     }
50: }
```

A Figura 21.3 mostra esse aplicativo com o item Opera como a seleção atual na caixa de combinação. O método toString( ) do objeto é usado para recuperar o texto do objeto retornado por getItem( ).

Figura 21.3

A *saída do aplicativo SelectItem.*



Eventos de tecla

NOVO TERMO ▶ os *eventos de tecla* ocorrem quando uma tecla é pressionada. Qualquer componente pode gerar esses eventos e uma classe precisa implementar a interface KeyListener para oferecer-lhes suporte.

Existem três métodos na interface KeyListener: keyPressed(keyEvent), keyReleased(*KeyEvent*) e*KeyEvent(KeyEvent).* Eles têm a seguinte forma:

```
public void keyPressed(KeyEvent evt) {
    // ...
}
public void keyReleased(KeyEvent evt) {
    // ...
}
public void keyTyped(KeyEvent evt) {
    // ...
}
```

O método getKeyChar( ) de KeyEvent retorna o caractere da tecla associada ao evento. Se não existir um caractere Unicode que possa ser representado pela

tecla, getKeyChar( )retornará um valor de caractere igual à variável de classe KeyEvent.CHAR_UNDEFINED.

Eventos de mouse

Os eventos de mouse são gerados por vários tipos diferentes de interação do usuário:

- Um clique de mouse.
- Um mouse que entra na área de um componente.
- Um mouse que deixa a área de um componente.

Qualquer componente pode gerar esses eventos, que são implementados por uma classe pela interface MouseListener. Essa interface possui cinco métodos:

mouseCl*icked{MouseEvent)*

mouseEntered(Moivsefi/ení)

mouseExited(*MouseEvent*)

mousePressed(*MouseEvent*)

mouseReleased*(MouseEvent*).

Cada um deles assume a mesma forma básica de mouseRel eased *(MouseEvent):*

```
public void mouseReleased(MouseEvent evt) {
    // ...
}
```

Os seguintes métodos podem ser usados em objetos MouseEvent:

- getClickCount( ) Retorna o número de vezes que o mouse foi acionado, como um inteiro.
- getPoint( ) Retorna as coordenadas x,y dentro do componente em que o mouse foi acionado, como um objeto Point.
- getX( ) Retorna a posição x.
- getY( ) Retorna a posição y.

Eventos de movimento de mouse

Os eventos de movimento de mouse ocorrem quando um mouse é movido sobre um componente. Assim como acontece com os outros eventos de mouse, qualquer componente pode gerar eventos de movimento de mouse. Para oferecer suporte a eles, uma classe precisa implementar a interface MouseMotionLi stener.

Existem dois métodos na interface MouseMotionLi stener: mouse-*Oragged(MouseMotionEvent)* e *mouseMoved(MouseMotionEvent).* Eles têm a seguinte forma:

```
public void mouseDragged(MouseEvent evt) {
    // •••
}

public void mouseMoved(MouseEvent evt) {
    // ...
}
```

Ao contrário das outras interfaces receptoras de evento com que você tratou até agora, MouseMotionListener não possui seu próprio tipo de evento. Em vez disso, são utilizados objetos MouseEvent.

Por isso, você pode chamar os mesmos métodos que chamaria para eventos de mouse: getClick( ),getPoint( ), getX( )egetY( ).

Eventos de janela

Os eventos de janela ocorrem quando um usuário abre ou fecha um objeto janela, como um J Frame ou um JWindow. Qualquer componente pode gerar esses eventos e uma classe precisa implementar a interface WindowListener para oferecer-lhes suporte.

Existem sete métodos na interface WindowListener:

wi ndowActi vated *(WindowEvent)*

windowClose*d(WindowEvent)*

windowClosing*(WindowEvent)*

windowDeactivated*{WindowEvent)*

windowDeiconifie*d(WindowEvent)*

wi ndowIconified*(UindowEvent)*

wi ndowOpened*(UindowEvent)*

Todos eles assumem a mesma forma que o método wi ndowOpened ( ):

```
public void windowOpened(WindowEvent evt) {
    // —
}
```

Os métodos windowClosing( ) e windowClosed( ) são semelhantes, mas um é chamado quando a janela está fechando e o outro é chamado depois que ela está fechada. Na verdade, você pode executar uma ação no método wi ndow-Cl osi ng( ) para interromper o fechamento da janela,

Um exemplo: o aplicativo SwingColorTest

Ontem, você criou uma interface gráfica com o usuário para o programa de conversão de RGB para HSB, usando os componentes do Swing.

Para ter mais experiência no trabalho com o modelo de tratamento de eventos em vigor no Swing, você transformará a interface SwingColorTest em um programa em funcionamento.

Duas classes foram criadas para esse projeto, ontem: SwingColorTest e SwingColorControls. SwingColorTest contém a janela do aplicativo e o método main( ) que é usado para configurá-la. SwingColorControls, uma classe auxiliar, é um painel que contém três rótulos e três campos de texto, utilizados para se escolher uma cor.

Toda a entrada de usuário nesse programa ocorre nos controles de cor — os campos de texto são usados para definir valores sRGB ou HSB.

Por isso, todo o comportamento de tratamento de eventos será incluído na classe SwingColorControls.

A primeira coisa a fazer é conseguir que a classe SwingColorControls trate desses dois tipos de eventos: eventos de ação e eventos de foco. A cláusula extends deve ser incluída na instrução da declaração class, para que as interfaces ActionListener e FocusListener sejam implementadas. Isso está mostrado a seguir:

```
class SwingColorControls extends JPanel
    implements ActionListener, FocusListener {
```

Em seguida, os receptores de ação e foco devem ser incluídos nos três campos de texto da classe: tfieldl, tfield2 e tf i ei d3. Esses receptores devem ser incluídos depois que os campos de texto forem criados, mas antes de serem inseridos em um contêiner. As seguintes instruções podem ser usadas:

```
tfieldl.addFocusListener(this);
tfield2.addFocusListener(this);
tfield3.addFocusListener(this);
tfieldl.addActionListener(this);
tfield2.addActionListener(this);
tfield3.addActionListener(this);
```

Finalmente, você precisa incluir todos os métodos que são definidos nas duas interfaces que essa classe implementa: actionPerformed(actionEvent), focusLost(focusEvent) e focusGained(focusEvent).

Os controles de cor são usados para se introduzir um valor numérico para uma cor e isso faz com que essa cor seja desenhada em um painel. Isso também faz os outros controles de cor serem atualizados para refletir a mudança de cor.

Existem duas maneiras pelas quais um usuário pode finalizar uma nova escolha de cor — pressionando Enter dentro de um campo de texto, o que gera um evento de ação, e saindo do campo para editar outro, o que gera um evento de foco.

As instruções a seguir compõem os métodos actionPerformed( ) e focusLost( ) que devem ser incluídos na classe:

```
public void actionPerformed(ActionEvent evt) {
    if (evt.getSource( ) instanceof TextField)
      frame.update(this);
}
```

```
public void focusLost(FocusEvent evt) {
    frame.update(this);
}
```

Um deles, focusGained( ), não precisa ser tratado. Por isso, uma definição de método vazia deve ser incluída:

```
public void focusGained(FocusEvent evt) { }
```

Os métodos de tratamento de eventos incluídos em SwingColorControls chamam um método em sua classe de origem, *update(SwingColorControls).*

Esse método não contém nenhum comportamento de tratamento de eventos — ele atualiza a amostra de cor e todos os controles para refletir uma mudança de cor. Ele é idêntico à versão criada no Dia 14.

A listagem 21.4 contém o aplicativo, incluindo as duas classes SwingColorTest e SwingColorControls.

DIGITE

Listagem 21.4. O texto completo de swingcolortest.java.

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import java.awt.swing.*;
4:
5: public class SwingColorTest extends JFrame {
6:     SwingColorControls RGBcontrols, HSBcontrols;
7:     JPanel swatch;
8:
9:     public SwingColorTest( ) {
10:         super("Color Test");
11:
12:        JPanel pane • new JPanel( );
13:        pane.setLayout(new GridLayout(l, 3, 5, 15));
14:        swatch = new JPanel( );
15:        swatch.setBackground(Color.black);
16:        RGBcontrols • new SwingColorControls(this, "Red",
17:            "Green", "Blue");
18:        HSBcontrols - new SwingColorControls(this, "Hue",
19:            "Saturation", "Brightness");
20:        pane.add(swatch);
21:        pane.add(RGBcontrols);
22:        pane.add(HSBcontrols);
23:
24:        setContentPane(pane);
25:    }
26:
27:    public static void main(String[ ] args) {
28:        JFrame frame - new SwingColorTest( );
29:
30:        WindowListener 1 = new WindowAdapter( ) {
31:            public void windowClosing(WindowEvent e) {
```

Listagem 21.4. Continuação

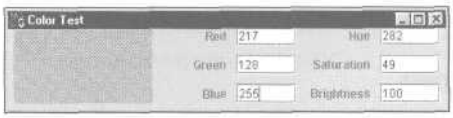
```
32:                 System.exit(0);
33:             }
34:         };
35:         frame.addWindowListener(l);
36:
37:         frame.pack( );
38:         frame.setVisible(true);
39:     }
40:
41:     public Insets getlnsets( ) {
42:        return new Insets(10, 10, 10, 10);
43:     }
44:
45:     void update(SwingColorControls Control Panei) {
46:         Color c;
47:         // obtém os valores de string dos campos de texto, converte em inte
48:         int valuel = Integer.parseInt(controlPanel.tfieldl.getText( ));
49:         int value2 = Integer.parselnt(controlPanei.tfield2.getText( ));
50:         int value3 = Integer.parseInt(controlPanel.tfield3.getText( ));
51:
52:         if (controlPanei == RGBcontrols) {
53:             // RGB mudou, atualiza HSB
54:            c = new Color(valuel, value2, value3);
55:
56:            // converte valores RGB em valores HSB
57:           float[ ] HSB = Color.RGBtoHSB(valuel, value2, value3,
58:                (new float[3]));
59:            HSB[0] *= 360;
60:            HSB[1] *- 100;
61:            HSB[2] *= 100;
62:
63:            // redefine os campos HSB
64:             HSBcontrols.tfieldl.setText(String.valueOf((int)HSB[0]));
65:             HSBcontrols.tfield2.setText(String.value0f((int)HSB[l]));
66:             HSBcontrols.tfieid3.setText(String.vaiueOf((int)HSB[2]));
67:
68:         } else {
69:            // HSB mudou, atualiza RGB
70:            c = Color.getHSBColor((float)valuel / 360,
71:                  (float)value2 / 100, (float)value3 / 100);
72:
73:            // redefine os campos RGB
74:            RGBcontrols.tfieldl.setText(String.valueOf(c.getRed( )));
75:            RGBcontrols.tfield2.setText(String.value0f(c.getGreen( )));
76:            RGBcontrols.tfield3.setText(String.value0f(c.getBlue( )));
77:         }
78:
79:           // atualiza a amostra
```

Listagem 21.4. Continuação

```
80:                swatch.setBackground(c);
81:                swatch.repaint( );
82:        }
83: }
84:
85: class SwingColorControls extends JPanei
86:        implements ActionListener, FocusListener {
87:
88:        SwingColorTest frame;
89:        JTextField tfieldl, tfield2, tfield3;
90:
91:        SwingColorControls(SwingColorTest parent,
92:            String 11, String 12, String 13) {
93:
94:            frame = parent;
95:             setLayout(new GridLayout(3,2,10,10));
96:            tfieldl = new JTextField("0");
97:            tfield2 = new JTextField("0");
98:            tfield3 = new JTextField("0");
99:       tfieldl.addFocusLi       stener(thi s);
100:            tfield2.addFocusListener(this);
101:     tfield3.addFocusLi       stener(thi s);
102:     tfieldl.addActi       onLi stener(thi s);
103:     tfield2.addActi       onListener(thi s);
104:           tfield3.addActionListener(this);
105:           add(new JLabel(ll, JLabel.RIGHT));
106:           add(tfieldl);
107:           add(new JLabel(12, JLabel.RIGHT));
108:           add(tfield2);
109:           add(new JLabel(13, JLabel.RIGHT));
110:           add(tfield3);
111:       }
112:
113:       public Insets getInsets( ) {
114:           return new Insets(10, 10, 0, 0);
115:       }
116:
117:       public void actionPerformed(ActionEvent evt) {
118:            if (evt.getSource( ) instanceof TextField)
119:                frame.update(this);
120:       }
121:
122:       public void focusLost(FocusEvent evt) {
123:           frame.update(this);
124:       }
125:
126:       public void focusGained(FocusEvent evt) { }
127: }
```

A Figura 21.4 mostra o produto final.

Figura 21.4
O *aplicativo SwingColorTest.*



Resumo

Internamente, o sistema de tratamento de eventos utilizado com o Swing é muito mais robusto e facilmente ampliado para tratar de novos tipos de interação com o usuário.

Externamente, o novo sistema também deve fazer mais sentido do ponto de vista da programação. O tratamento de eventos é incluído em um programa com as mesmas etapas:

- Uma interface receptora é adicionada à classe que vai conter os métodos de tratamento de eventos.
- Um receptor é adicionado a cada componente que vai gerar os eventos a serem tratados.
- Os métodos são adicionados, cada um com uma classe EventObject como único argumento do método.
- Os métodos dessa classe EventObject, como getSource( ), são usados para se saber quais componentes geraram o evento e qual era o tipo do evento.

Uma vez que se saiba essas etapas, você pode trabalhar com cada uma das diferentes interfaces receptoras e classes de evento. Você também pode aprender a respeito de novos receptores, à medida que eles forem incluídos no Swing com novos componentes.

Isso nos leva ao evento principal: a conclusão da viagem de 21 dias pela linguagem Java. Agora que você já teve a chance de trabalhar com a sintaxe e com as classes mais importantes que compõem a linguagem Java, está pronto para atacar o material realmente difícil: seus próprios programas.

Este livro possui um site oficial na World Wide Web, no endereço http://www.prefect.com/java21. Ele apresenta respostas às perguntas mais freqüentes dos leitores, todo o código-fonte do livro e, para o evento improvável de que um erro (tipográfico ou não) seja encontrado nesses capítulos, uma página de errata onde podem ser encontradas correções.

Parabéns! Agora que você foi apresentado à linguagem de programação mais extraordinária da última década, fica por sua conta fazer coisas ainda mais notáveis com ela.

Quando passar longas horas em seus próprios programas, aprendendo novos recursos e ampliando as classes Java em seus próprios pacotes, você aprenderá outro motivo no qual o nome da linguagem foi inspirado.

A linguagem Java, assim como o café, que a simboliza, pode viciar.

Nota

"Não deixe isso terminar assim. Diga-lhes que eu falei alguma coisa."

— As últimas palavras de Pancho Villa (1877-1923)

Perguntas e respostas

P **O comportamento de tratamento de eventos para um programa pode ser colocado em sua própria classe, em vez de ser incluído no código que cria a interface?**

R Pode, e muitos programadores irão lhe dizer que essa é uma boa maneira de fazer o design de seus programas. Separar o design da interface de seu código de tratamento de eventos permite que os dois sejam desenvolvidos separadamente — o aplicativo SwingColorTest de hoje mostra a estratégia alternativa. Isso torna mais fácil manter o projeto; o comportamento relacionado é agrupado e isolado do comportamento não-relacionado.

APÊNDICES

A Resumo da linguagem Java

B A biblioteca de classe Java

C Recursos sobre Java na Internet

D Configurando o Java Development Kit

E Usando um editor de textos com o JDK

Apêndice A

Resumo da linguagem Java

Este apêndice contém um resumo da linguagem Java, conforme descrita neste livro.

Esta não é uma visão geral da gramática ou técnica da linguagem em si. Trata-se de uma referência rápida para ser usada depois que você já souber os fundamentos do funcionamento da linguagem. Se você precisar de uma descrição técnica da linguagem, pode visitar a seção de documentação do site da Web da JavaSoft, no endereço http://java.sun.com/docs. Um dos documentos a que ela oferece acesso integral é a especificação oficial da linguagem Java, que inclui a gramática completa.

As palavras-chave e os símbolos da linguagem são apresentados com um tipo diferente. Os argumentos e outras partes a serem substituídas estão em *tipo diferente e em itálico.*

As partes opcionais estão indicadas por colchetes (exceto na seção sobre a sintaxe de array). Se existirem várias opções mutuamente exclusivas, elas serão apresentadas por traços verticais (!), como segue:

[public ! private ! protected] type *vamame*

Palavras reservadas

As palavras a seguir são reservadas para uso pela própria linguagem Java. (Algumas delas são reservadas, mas não utilizadas no momento.) Você não pode usar esses termos para fazer referência a classes, métodos ou nomes de variável:

abstract	do	import	return	void
boolean	double	instanceof	short	volatile
break	else	int	static	while
byte	extends	interface	super	
case	final	long	switch	
catch	finally	native	synchronized	
char	float	new	this	
class	for	package	throw	
const	goto	private	throws	
continue	if	protected	transient	
default	implements	public	try	

As palavras true, false e null também são reservadas para uso como literais.

Comentários

```
/* este é o formato de um comentário
   de várias linhas */
// este é um comentário de uma linha
/** este é um comentário do Javadoc */
```

Literais

number	Tipo int
number[1: L]	Tipo long
Oxhexnumber	Inteiro hexadecimal
oxhexnumber	Inteiro hexadecimal
Ooctalnumber	Inteiro octal
[*number*] *.number*	Tipo double
numberl f ! f]	Tipo float
number[d ! D]	Tipo double
[+ ! -] *number*	Com sinal
number[e ! E] number	Expoente
'character'	Caractere simples
"characters"	String
""	String vazio
\b	Backspace
\t	Tab

\n	Line feed
\f	Form feed
\r	Carriage return
\"	Aspas
\'	Apóstrofo
\\	Barra invertida
uhexnumber	Escape Unicode
null	Null
true	Booleano
false	Booleano

Declararão de variável

[byte ! short ! int ! long] *varname*	Inteiro (Selecione um tipo)
[float ! double] *varname*	Flutuantes (Selecione um tipo)
char *varname*	Caracteres
boolean *varname*	Booleano
classname *varname*	Tipos de classe
type *varname, varname, varname*	Múltiplas variáveis

As opções a seguir estão disponíveis apenas para variáveis de classe e de instância. Todas essas opções podem ser usadas com uma declaração de variável:

t static] variableDeclaration	Variável de classe
[final] variableDeclaration	Constantes
[public ! private ! protected]	Controle de acesso *variableDeclaration*

Atribuicão de variavel

variable = value	Atribuição
variable++	Incremento pós-fixado
++variable	Incremento pré-fixado
variable-	Decremento pós-fixado
-variable	Decremento pré-fixado
variable += value	Soma e atribuição
variable -= value	Subtração e atribuição
*variable *= value*	Multiplicação e atribuição

variable /= value	Divisão e atribuição
variable %= value	Módulo e atribuição
variable &= value	AND e atribuição
variable := value	OR e atribuição
variable ^= value	XOR e atribuição
variable «= value	Deslocamento à esquerda e atribuição
variable »= value	Deslocamento à direita e atribuição
variable <«= value	Preenchimento com zero, deslocamento à esquerda e atribuição

Operadores

arg + *arg*	Adição
arg - *arg*	Subtração
arg * *arg*	Multiplicação
arg / *arg*	Divisão
arg % *arg*	Módulo
arg < *arg*	Menor que
arg > *arg*	Maior que
arg <= *arg*	Menor ou igual a
arg >= *arg*	Maior ou igual a
arg == *arg*	Igual
arg != *arg*	Diferente
arg && *arg*	AND lógico
arg " *arg*	OR lógico
! *arg*	NOT lógico
arg & *arg*	AND
arg ! *arg*	OR
arg ^ *arg*	XOR
arg « org	Deslocamento à esquerda
arg » *arg*	Deslocamento à direita
arg >» arg	Deslocamento à direita com preenchimento com zero
- arg	Complemento
(type)t/)ing	Coerção
arg instanceof class	Instância de
test ? *trueOp : falseOp*	Operador ternário (i f)

Objetos

new *class{*);	Cria nova instância
new *class(arg, arg, arg...)*	Nova instância com parâmetros
new *type(arg, arg, arg...)*	Cria nova instância de uma classe anônima
Primary .new *type(arg, arg, arg...)*	Cria nova instância de uma classe anônima
object. variable	Variável de instância
object.classvar	Variável de classe
Class.cZffssvcrr	Variável de classe
object.method( )	Método de instância (sem args)
object.method(arg, arg, arg...)	Método de instância
object.classmethod( )	Método de classe (sem args)
object.classmethod{arg, arg, arg...)	Método de classe
C\ass.classmethod( )	Método de classe (sem args)
C\ass.classmethod(arg, arg, arg...)	Método de classe

Arrays

Nesta seção, os colchetes fazem parte da criação do array ou das instruções de acesso. Eles não denotam partes opcionais como acontece em outros lugares deste apêndice.

type *varname* [ ]	Variável de array
type[ ] *varname*	Variável de array
new *type[numElements]*	Novo objeto de array
new *type[*] *{initializer}*	Novo objeto de array anônimo
array[index]	Acesso a elemento
array. length	Comprimento do array

Loops e condicionais

if (*test*) *block*	Condicional
if (*test) block*	Condicional com else

else *block*	
switch (*test*) { }	switch (somente com tipos integer ou char) case valor : instruções case valor: instruções default : instrução
for (*initializer; test; change)*	Bloco de loop for
while (*test*) block	while *(test)*
do *block*	Loop do
While (test)	
break [*label*]	Interrupção de loop ou switch from
continue [*label*]	Continua loop
label:	Loops com rótulo

Definições de classe

class *classname block*	Definição de classe simples

Qualquer um dos seguintes modificadores opcionais pode ser incluído na definição de classe:

[final] class *classname block*	Nenhuma subclasse
[abstract] class *classname block*	Não pode ser instanciada
[public] class *classname block*	Acessível fora do pacote
class classname [extends Superclass] block	Define a superclasse
class *classname* [implements *interfaces*] *block*	Implementa uma ou mais interfaces

Definições de método e construtor

O método básico é semelhante ao seguinte, onde returnType é um nome de tipo, um nome de classe ou void:

returnType methodName( ) *block*	Método básico
returnType methodNawe{parameter, parameter...) block	Método com parâmetros

Os parâmetros de método são como segue:

type parameterName

As variações de método podem incluir qualquer uma das seguintes palavras-chave opcionais:

[abstract] *returnType methodName{) block*	Método abstrato
[static] *returnType methodName{) block*	Método de classe
[native] *returnType methodName{) block*	Método nativo
[final] *returnType methodName( ) block*	Método final
[synchronized] *returnType methodName( ) block*	Bloco de thread antes de executar
[public ! private ! protected] *returnType methodName( )*	Controle de acesso

Os construtores são semelhantes ao seguinte:

classnawe{) block	Construtor básico
classname(parameter, parameter, parameter...) block	Construtor com parâmetros
[public ! private ! protected] *classname( ) block*	Controle de acesso

No corpo do método/construtor, você pode usar as seguintes referências e métodos:

this	Refere-se ao objeto atual
classname. this	Refere-se a um objeto de classe interna em particular
super	Refere-se à superclasse
super.*methodName( )*	Chama o método de uma superclasse
this (…)	Chama o construtor da classe
super (…)	Chama o construtor da superclasse
type. class	Retorna objeto da classe do tipo
return [*value*]	Retorna um valor

Importação

import *package.className*	Importa nome de classe específico
import *package.**	Importa todas as classes do pacote
package *packageName*	As classes desse arquivo pertencem ao pacote
interface *interfaceName* [extends *anotherlnterface*] *block*	Definição de interface simples
[public] interface *interfaceName block*	Interface pública
[abstract] interface *interfaceName block*	Interface abstrata

Proteção

synchronized (*object*) *block*	Espera para bloquear o objeto
try *block*	Instruções protegidas
catch (*exception*) *block*	Executado se a exceção for levantada
[finally *block*]	Sempre executado
try *block* [catch (*exception*) *block*] finally *blockl*	O mesmo que o anterior (agora pode utilizar catch ! finally opcional, mas não ambos)

Apêndice B

A biblioteca de classe Java

Este apêndice fornece uma visão geral das classes disponíveis nos pacotes Java padrão, os quais se garante estarem disponíveis em qualquer implementação Java. Para obter informações mais específicas sobre cada classe (como herança, variáveis e métodos), visite a seção Products and APIs do site na Web da Sun (http://java.sun.com).

Algumas classes, métodos e variáveis são listados como desaprovados. Eles estão incluídos na API da Java 2 para manter a compatibilidade com as versões anteriores, mas foram substituídos por soluções melhores. Consulte a documentação da API para conhecer os detalhes sobre as substituições.

- java.applet
- java.awt
- javax.accessibility
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font
- java.awt.geom
- java.awt.im
- java.awt.image
- java.awt.print
- javax.swing
- javax.swing.basic

- javax.swing.beaninfo
- javax.swing.border
- javax.swing.event
- javax.swing.jlf
- javax.swing.motif
- javax.swing.multi
- javax.swing.plaf
- javax.swing.table
- javax.swing.target
- javax.swing.tree
- javax.swing.text
- javax.swing.undo
- java.beans
- java.beans.beancontext
- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.math
- java.net
- java.rmi
- java.rmi.activation
- java.rmi.dgc
- java.rmi.registry
- java.rmi.server
- java.security
- java.security.acl
- java.security.cert
- java.security.interfaces
- java.security.spec
- java.sql
- java.text
- java.util

- java.util.jar
- java.util .mime
- java.util.zip

java.aPDlet

Esse pacote permite que um programa Java seja executado como parte de uma página da Web e reproduza arquivos de som. A linguagem Java 2 inclui recursos de áudio aprimorados, oferecendo suporte a arquivos de 8 e 16 bits, taxas de amostragem de 8 a 48 kHz e os seguintes formatos: AIFF, AU, RMF, WAV e MIDI Tipo 0 e 1. Um novo método na classe AudioClip, newAudioClip, permite que os aplicativos reproduzam arquivos de som.

Interfaces

AppletContext	Métodos relacionados ao ambiente de um applet
AppletStub	Métodos para implementar um visualizador de applet
AudioClip	Métodos para reproduzir arquivos de áudio

Classes

Applet	A classe de applet básica

java.awt

Esse pacote, o Abstract Windowing Toolkit, manipula imagens gráficas e a interface gráfica com o usuário de um programa.

Interfaces

ActiveEvent	Métodos para eventos que podem ser enviados sozinhos (Java 2)
Adjustable	Métodos para objetos com valores numéricos ajustáveis dentro de um intervalo de valores aceitáveis (Java 1.1)
Composite	Métodos para criar uma composição personalizada de uma nova imagem bidimensional e a imagem que está sob ela (Java 2)
CompositeContext	Métodos para definir o ambiente de uma operação de composição de imagem bidimensional personalizada (Java 2)

EventSource	Métodos para objetos que geram eventos (interface beta da Java 1.1 excluída do release final; não presente na Java 2)
ItemSelectable	Métodos para objetos que contêm um conjunto de itens, em que nenhum, um ou mais itens podem ser selecionados (Java 1.1)
LayoutManager	Métodos para organizar componentes de interface em um container
LayoutManager2	Representação mínima de métodos para organizar componentes baseados nas restrições que indicam exatamente onde um compontente deve ficar em um container (Java 1.1)
MenuContainer	Métodos para contêineres relacionados a menus. (Seu método postEvent( ) foi desaprovado a partir da Java 2.)
Paint	Métodos para definir padrões de preenchimento para operações de desenho bidimensionais (Java 2)
PaintContext	Métodos para definir o ambiente de uma operação de desenho de padrão personalizado bidimensional (Java 2)
PrintGraphics	Métodos para fornecer um contexto gráfico para impressão (Java 1.1)
Shape	Métodos para objetos que representam figuras geométricas (introduzidos na Java 1.1 e revisadas como parte da funcionalidade de imagem bidimensional da Java 2)
Stroke	Métodos para representar a área de um pincel de desenho em operações gráficas bidimensionais (Java 2)
Transparency	Métodos para definir modos de transparência (Java 2)

Classes

AWTEvent	A origem de todos os eventos AWT (Java 1.1)
AWTEventMulticaster	Transmissor de eventos de multicoerção (Java 1.1)
AWTPermission	Um modo de criar permissões que controlam o acesso aos recursos do AWT (Java 2)
AlphaComposite	Canal alfa para regras de composição de imagens (Java 2)
BasicStroke	Pincel de desenho representando uma figura de caneta e figura nos finais de linhas (Java 2)

BorderLayout	Gerenciador de layout para organizar componentes de interface em um retângulo central e retângulos nos quatro pontos cardeais. (Seu método addLayoutComponent( ) foi desaprovado a partir da Java 1.1.)
Button	Botão que pode receber um clique de mouse em uma interface com o usuário
Canvas	Área retangular em branco de uma interface para desenhar e receber eventos de usuário
CardLayout	Gerenciador de layout representando componentes como uma pilha de cartas apenas com a que está em cima visível. (Seu método addLayoutComponent( ) foi desaprovado a partir da Java 1.1.)
Checkbox	Uma caixa que pode aparecer marcada ou desmarcada em uma interface com o usuário, com um rótulo de texto identificando-a
CheckboxGroup	Meio de agrupar duas ou mais caixas de seleção de modo que apenas uma possa ser marcada por vez. (Seus métodos getCurrent ( ) esetCurrent( ) foram desaprovados a partir da Java 1.1.)
CheckboxMenuItem	Item em um menu suspenso que pode ser marcado ou desmarcado
Choice	Uma lista instantânea de escolhas, a partir da qual uma única seleção pode ser feita. (Seu método countItems( ) foi desaprovado a partir da Java 1.1.)
Color	Representação de uma cor em uma organização de cores RGB padronizada ou uma organização alternativa definida por um objeto ColorSpace
Component	A origem de todos os componentes de interface com o usuário que não fazem parte de um menu suspenso (numerosos métodos foram desaprovados a partir da Java 1.1: action( ),bounds( ), deliverEvent( ), disable( ),enable( ),getPeer( ),gotFocus( ), handleEvent( ),hide( ), inside( ), keyDown( ), keyUp( ),layout( ),locate( ),location( ), lostFocus( ),minimumSize( ),mouseDown( ), mouseDrag( ),mouseEnter( ),mouseExit( ), mouseMove( ),mousellp( ),move( ),nextFocus( ), postEvent( ), preferredSize( ),reshape( ), resize( ), show( )esize( ))

Container	Um componente de interface com o usuário que pode conter outros componentes e contêineres (vários métodos foram desaprovados a partir da Java 1.1: countComponents( ),deliverEvent( ), insets( ),layout( ), locate( ), minimumSize( ) e preferredSize( ))
Cursor	Um ponteiro de mouse em mapa de bits (Java 1.1)
Dialog	Uma janela para uma interação simples com o usuário
Dimension	Representação da largura e da altura de um componente ou outro objeto
Event	Objeto representando eventos no modelo de tratamento de eventos Java 1.0.2 (substituídos nas versões 1.1 e 2, mas mantidos por questões de compatibilidade retroativa)
EventQueue	Uma lista de eventos esperando para serem processados (Java 1.1)
Filedialog	Janela usada para selecionar um arquivo através da navegação em uma estrutura de diretório
FlowLayout	Gerenciador de layout que organiza objetos da esquerda para a direita e de cima para baixo, como palavras em uma página de papel
Font	Representação de uma fonte
FontMetrics	Exibição das medidas de uma fonte. (Seu método getMaxDecent( ) foi desaprovado a partir da Java 1.1 e substituído por getMaxOescent( ).)
Frame	Janela de nível superior com título e borda. (Seus métodos getCursorType( ) e setCursor( ) foram desaprovados a partir da Java 1.1; numerosas variáveis foram desaprovadas a partir da Java 2: CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, M0VE_CURS0R, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NW_RESIZE_CURSOR, S_RESIZE CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR e WAIT_CURSOR.)
GradientPaint	Comportamento de desenho de um padrão de gradiente dentro de uma figura (Java 2)

Graphics	Comportamento representando um contexto em que objetos podem ser apresentados e operações de desenho podem ocorrer. (Seu método getClipRect( ) foi desaprovado a partir da Java 1.1.)
Graphics2D	Extensão de Graphics que permite rendenzação bidimensional sofisticada (Java 2)
GraphicsConfiguration	Características físicas de um dispositivo de saída gráfico (Java 2)
GraphicsDevice	Dispositivo de saída gráfico, como uma impressora ou um monitor (Java 2)
GraphicsEnvi ronment	Representação de todos os dispositivos de saída gráficos e objetos de fonte que são apresentados em um ambiente (Java 2)
GridBagConstraints	Organização de uma célula específica em uma organização GridBagLayout
GridBagLayout	Gerenciador de layout que organiza os componentes em uma grade de células de tamanhos diferentes, de acordo com os valores de GridBagConstraints
GridLayout	Gerenciador de layout que organiza os componentes em uma grade de células
Image	Representação de uma imagem de mapa de bits
Insets	As medidas das margens de um contêiner, quando elas afetam o layout do componente
Label	Rótulo de texto em uma interface que não pode ser alterado diretamente por um usuário
List	Lista rolante de itens, a partir da qual um ou mais itens podem ser selecionados. (Vários métodos foram desaprovados a partir da Java 1.1: allowsMultipleSelections( ),clear( ), countltems( ),deltems( ), isSelected( ), minimumSize( ), preferredSize( )e setMul tipleSelections( ); Além disso, seus métodos addltem( ) e delltem() foram desaprovados a partir da Java 2.)
MediaTracker	Um modo de monitorar o status de imagens que estão sendo carregadas por download, como arquivos usados por um applet

Menu	Conjunto de menus suspensos, associados à barra de menus de um quadro. (Seu método countltems( ) foi desaprovado a partir da Java 1.1.)
MenuBar	Contêiner para menus associados a um quadro. (Seu método countMenus( ) foi desaprovado a partir da Java 1.1.)
MenuComponent	Origem de todos os elementos de menu. (Seus métodos getPeer( )epostEvent( ) foram desaprovados a partir da Java 1.1.)
Menultem	Um item específico em um menu suspenso. (Seus métodos disable( )eenable( ) foram desaprovados a partir da Java 1.1.)
MenuShortcut	Atalho de teclado para a seleção de um item de menu suspenso (Java 1.1)
Panei	Contêiner cujos componentes podem ser posicionados de acordo com os propósitos do gerenciamento de layout
Point	Representação de um ponto em um espaço de coordenadas x,y
Polygon	Região bidimensional fechada, representada por um conjunto de pontos x,y no espaço de coordenadas. (Seus métodos getBoundingBox( ) e inside( ) foram desaprovados a partir da Java 1.1.)
PopupMenu	Menu de itens que aparecem em uma posição especificada dentro de um componente (Java 1.1)
PrintJob	Comportamento para iniciar e executar uma operação de impressão (Java 1.1)
PrintJob2D	Comportamento para imprimir uma lista de páginas e uma janela permitindo que a operação de impressão seja configurada (Java 2)
Rectangle	Um retângulo representado por quatro pontos x,y que compõem seus cantos em espaço de coordenadas (Seus métodos inside( ),move( ),reshape( ) eresize( ) foram desaprovados a partir da Java 1.1.)
ScrolIPane	Contêiner que controla o rolamento horizontal e vertical, conforme for necessário para um único componente dependente (Java 1.1; seu método layout( ) foi desaprovado a partir da Java 1.1.)

Scrollbar	Barra de rolagem em uma interface com o usuário. (Vários métodos foram desaprovados a partir da Java 1.1: getLineIncrement( ), getPageIncrement( ), getVisible( ), setLineIncrement( ) e setPageIncrement( ).)
SystemColor	As cores utilizadas para os elementos padrão da interface com o usuário de um sistema operacional (Java 1.1)
TextArea	Área retangular em uma interface onde várias linhas de texto podem ser apresentadas, introduzidas ou modificadas pelos usuários. (Vários métodos foram desaprovados a partir da Java 1.1: appendText( ), insertText( ), minimumSize( ), preferredSize( ) e replaceText( ).)
TextComponent	A origem de todos os componentes que permitem a edição de textos
TextField	Área retangular em uma interface onde uma única linha de texto pode ser exibida, introduzida ou modificada. (Seus métodos minimumSize( ), preferredSize( )e setEchoCharacter( ) foram desaprovados a partir da Java 1.1.)
TexturePaint	Comportamento para desenhar um padrão de textura dentro de uma figura (Java 2)
Tool kit	Comportamento que vincula os diversos elementos do AWT às suas implementações nativas em uma plataforma específica
Window	Janela em uma interface com o usuário. (O método postEvent( ) foi desaprovado a partir da Java 1.1.)

javax.accessibil ity

Esse pacote, introduzido na versão 2 da linguagem Java, expande a funcionalidade da interface com o usuário de um programa, para que ela possa utilizar tecnologia auxiliar, como terminais em Braille, leitores de tela e reconhecimento de fala.

B

Interfaces

Accessible	Métodos para tornar um elemento de interface com o usuário mais acessível

AccessibleLayout	Métodos que os contêineres podem usar para indicar as posições relativas dos objetos que eles contêm
AccessibleText	Métodos para determinar o texto apresentado em uma posição de coordenadas em um dispositivo de saída gráfico

Classes

AbstractAccessible	Representação de objetos que não são componentes, mas devem ser tratados como objetos acessíveis
AccessibleEnumeration	Comportamento para manter uma enumeração fortemente tipada
AccessibleRole	Uma descrição precisa do papel que um elemento de uma interface com o usuário desempenha nessa interface
AccessibieState	Uma descrição precisa do estado atual de um elemento de uma interface com o usuário
AccessibleStateSet	O conjunto de todos os objetos AccessibleState, representando o estado integral de um elemento de interface

java.awt.color

Esse pacote, introduzido na versão 2 da linguagem Java, descreve os diferentes sistemas usados para identificar cores e possibilita conversões entre os sistemas.

Classes

ColorSpace	O sistema de cores específico de um objeto Color, uma imagem ou um objeto Graphics
ICC ColorSpace	Um sistema de cores baseado no Profile Format Specification do ICC (International Color Consortium)
ICC_Profile	Representação de dados de perfil de cor da especificação ICC
ICC_ProfileGray	Subconjunto de perfis, como aqueles usados em dispositivos de entrada e saída monocromáticos
ICC_ProfileRGB	Subconjunto de perfis, como aqueles usados em dispositivos de entrada e saída de três componentes RGB

java.awt.datatransfer

Esse pacote, introduzido na linguagem Java 1.1, permite o armazenamento e a recuperação de informações dentro de um programa (ou em diferentes programas) usando-se operações de recortar, copiar e colar em um espaço de armazenamento chamado Área de Transferência.

Interfaces

ClipboardOwner	Métodos para objetos que possuem dados armazenados em uma Área de Transferência
Transferable	Métodos para objetos que podem ser usados para armazenar dados em uma Área de Transferência

Classes

Clipboard	Comportamento para transferir informações dentro de programas ou entre eles, usando operações de recortar, copiar e colar
Clipboard	Comportamento para transferir informações usando uma Área de Transferência
DataFlavor	Representação de um formato de dados que poderiam ser trocados usando-se uma Área de Transferência
StringSelection	Comportamento para transferir strings Java como texto puro usando uma Área de Transferência

java.awt.dnd

Esse pacote, introduzido na linguagem Java 2, manipula os dois lados de uma operação de arrastar e soltar.

Interfaces

DragSourceListener	Métodos para o emissor de uma operação de arrastar e soltar tratar de eventos relacionados
DropTargetListener	Métodos para o receptor de uma operação de arrastar e soltar tratar de eventos relacionados
FlavorMap	Métodos para fazer o mapeamento de tipos MIME em strings que identificam os tipos de dados da plataforma nativa, possibilitando transferências do tipo arrastar e soltar

Classes

DndConstants	Constantes usadas para tratar de uma operação de arrastar e soltar
DragSource	Originador de uma operação de arrastar e soltar
DragSourceContext	Ambiente do originador usado para notificação de eventos e que fornece um estado Transferable
DragSourceDragEvent	Evento usado dentro do originador para reagir à operação de arrastar
DragSourceDropEvent	Evento usado dentro do originador para reagir à operação de soltar
DragSourceEvent	Origem de eventos de fonte de arraste
DropTarget	Receptor de uma operação de arrastar e soltar
DropTargetContext	Ambiente do receptor usado para notificação de eventos e transferência de dados, quando aplicável
DropTargetContext. TransferableProxy	Comportamento para transferir dados
DropTargetDragEvent	Evento usado dentro do receptor para reagir à operação de arrastar
DropTargetDropEvent	Evento usado dentro do receptor para reagir à operação de soltar
DropTargetEvent	Origem dos eventos de destino de soltura

java.awt.event

Esse pacote, introduzido na linguagem Java 1.1, oferece um sistema de tratamento de eventos completamente revisado.

Interfaces

ActionListener	Métodos para responder aos eventos de ação
AdjustmentListener	Métodos para responder aos eventos de ajuste
ComponentListener	Métodos para responder aos eventos de componente
ContainerListener	Métodos para responder aos eventos de contêiner
FocusListener	Métodos para responder aos eventos de foco de teclado
InputMethodListener	Métodos para responder aos eventos de método de entrada (Java 2)
ItemListener	Métodos para responder aos eventos de item

KeyListener	Métodos para responder aos eventos de teclado
MouseListener	Métodos para responder aos eventos de mouse
MouseMotionListener	Métodos para responder aos eventos de movimento de mouse
TextListener	Métodos para responder aos eventos de texto
WindowListener	Métodos para responder aos eventos de janela

Classes

ActionEvent	Um evento de ação definido pelo componente
AdjustmentEvent	Um evento de ajuste gerado por um objeto Adjustable
ComponentAdapter	Classe abstrata para a criação de novas classes receptoras de componente
ComponentEvent	Evento de componente
ContainerAdapter	Classe abstrata para a criação de novas classes receptoras de contêiner
ContainerEvent	Evento de contêiner
FocusAdapter	Classe abstrata para a criação de novas classes receptoras de foco de teclado
FocusEvent	Evento de foco de teclado
InputEvent	Evento de entrada no nível de componente
ItemEvent	Evento de item gerado por um objeto ItemSelectable
KeyAdapter	Classe abstrata para a criação de novas classes receptoras de teclado
KeyEvent	Evento de teclado
MouseAdapter	Classe abstrata para a criação de classes receptoras de mouse
MouseEvent	Evento de mouse
MouseMotionAdapter	Classe abstrata para a criação de novas classes receptoras de movimento de mouse
PaintEvent	Evento de pintura no componente
TextEvent	Evento de texto gerado por um objeto como TextComponent
WindowAdapter	Classe abstrata para a criação de novas classes receptoras de janela
WindowEvent	Evento de janela

java.awt.font

Esse pacote, introduzido na linguagem Java 2, oferece suporte à composição de fonte, exibição e glifos específicos em fontes.

Interfaces

MultipleMaster	Métodos para oferecer suporte a fontes Multiple Master Tipo 1
OpenType	Métodos para oferecer suporte a fontes TrueType e OpenType

Classes

GlyphJustificationInfo	Propriedades de justificação de um glifo
GlyphMetrics	Medidas de glifos
GlyphSet	Representação gráfica de texto com ordenação bidirecional
StyledString	Comportamento exigido para apresentar texto
StyledStringIterator	Implementação do protocolo AttributedCharacterIterator para strings estilizados
TextAttributeSet	Atributos exigidos para layout de texto
TextHitInfo	Posição de caractere para especificar sinal de inclusão (circunflexo) e posições de inserção no texto
TextLayout	Representação gráfica de strings estilizados

java.awt.aeom

Trata-se de um pacote introduzido na linguagem Java 2, que oferece suporte à geometria bidimensional.

Interfaces

PathIterator	Métodos envolvidos na iteração de caminho

Classes

AffineTransform	Transformações Affine bidimensionais. (Seus métodos prepend( ) e append( ) foram desaprovados a partir da Java 2.)
Arc2D	Arcos definidos pelo retângulo envoltório, ângulo inicial, extensão angular e tipo de fechamento
Arc2D.Float	Arcos definidos com precisão float

Área	Uma geometria de área
CubicCurve2D	Segmentos de curva paramétricos cúbicos no espaço de coordenadas x,y
CubicCurve2D.Float	Segmentos de curva paramétricos cúbicos com coordenadas x,y float
Dimension2D	Representação de uma dimensão de largura e altura
Ellipse2D	Elipses definidas por retângulos envoltórios
Ellipse2D.Float	Elipses especificadas com precisão float
Fl atteni ngPathlterator	Vista achatada de outro objeto Pathlterator
GeneralPath	Caminhos geométricos construídos a partir de linhas retas e curvas quadráticas e cúbicas
GeneralPathlterator	Interadores de caminho gerais
Line2D	Segmentos de reta no espaço de coordenadas
Line2D.Float	Segmentos de reta especificados com coordenadas x,y float
Point2D	Pontos no espaço de coordenadas x,y
Point2D.Double	Pontos especificados com coordenadas x,y double
Point2D.Float	Pontos especificados com coordenadas x,y float
QuadCurve2D	Segmentos de curva quadráticos no espaço de coordenadas x,y
QuadCurve2D.Float	Segmentos de curva quadráticos especificados com coordenadas x,y float
Rectangle2D	Retângulos definidos por posições x,y e dimensões
Rectangle2D.Double	Retângulos especificados com coordenadas double
Rectangle2D.Float	Retângulos especificados com coordenadas float
RectangularShape	Figuras inscrevendo um conjunto retangular de limites exteriores
RoundRectangle2D	Retângulos bidimensionais com arcos descrevendo os cantos
RoundRectangle2D.Fl oat	Retângulos bidimensionais com os arcos dos cantos especificados com coordenadas float

java.awt.im

Trata-se de um pacote introduzido na linguagem Java 2, oferecendo suporte aos métodos de entrada, que são usados para introduzir milhares de caracteres diferentes,com várias seqüências de toques de tecla. Os métodos de entrada permitem que os componentes de texto recebam texto em chinês, japonês e coreano.

Interfaces

InputMethodRequests	Métodos que permitem que componentes de edição de texto trabalhem com métodos de entrada

Classes

InputContext	Comportamento para gerenciar a comunicação entre métodos de entrada e componentes de edição de texto
InputMethodHighlight	Estilos de destaque de método de entrada de texto que está sendo composto

java.awt.image

Trata-se de um pacote para o gerenciamento de imagens de mapa de bits.

Interfaces

BufferedlmageOp	Métodos para descrever operações em objetos Bufferedlmage (Java 2)
ItnageConsumer	Métodos para receber imagens criadas por objetos ImageProducer
ImageObserver	Métodos para monitorar o carregamento e a construção de uma imagem
ImageProducer	Métodos para produzir dados de imagem recebidos por objetos ImageConsumer
ImagingLib	Métodos para acessar código de geração de imagens específico de plataforma (Java 2)
RasterlmageConsumer	Métodos para receber imagens de varredura criadas por objetos RasterlmageProducer (Java 2)
RasterOp	Métodos para descrever operações sobre objetos Raster (Java 2)
Renderlmage	Métodos para objetos que produzem ou contêm imagens de varredura (Java 2)

TileChangeListener	Métodos para observar quando os setores de um objeto WritableRenderedlmage podem ser modificados (Java 2)
WriteableRenderedlmage	Métodos para objetos que produzem ou contêm imagens que podem ser modificadas e sobrescritas (Java 2)

Classes

Affi neTransformOp	Transformações affine para realizar mapeamentos lineares de uma imagem ou varredura para outra (Java 2)
AreaAveraging SealeFilter	Filtros para mudar a escala de imagens usando um algoritmo de média de área (Java 2)
BandCombineOp	Combinações lineares arbitrárias de associações de varredura usando matrizes especificadas (Java 2)
BandedSampleModel	Comportamento para fazer amostragem de imagens que armazenam dados associados em diferentes bancos de objetos DataBuffer (Java 2)
BilinearAffine TransformOp	Transformações affine bi-lineares para realizar o mapeamento linear de uma imagem ou varredura para outra (Java 2)
Bufferedlmage	Imagens com um buffer de dados acessível (Java 2; seu método getGraphics( ) foi desaprovado a partir da Java 2.)
BufferedlmageFi1 ter	Filtros que pegam dados de imagem colocados em buffers de objetos ImageProducer e passam versões modificadas para objetos ImageConsumer (Java 2)
ByteLookupTable	Tabelas de pesquisa para canais de setor ou estruturas de componentes de imagem, como RGB (Java 2)
ColorConvertOp	Conversões de cor, pixel por pixel, de imagens de origem (Java 2)
ColorModel	Comportamento para gerenciar informações de cor para imagens
ComponentColorModel	Comportamento para gerenciar informações de cor para imagens usando objetos ColorSpace arbitrários (Java 2)
ComponentSampl eModel	Comportamento para fazer amostragem de imagens que armazenam dados em elementos de array de dados separados no mesmo banco DataBuffer (Java 2)

ConvolveOp	Convoluções de uma imagem em outra (Java 2)
CropImageFil ter	Filtro para cortar imagens
DataBuffer	Encapsulamento de um ou mais arrays de dados em bancos (Java 2)
DataBufferByte	Buffers de dados armazenados como bytes (Java 2)
DataBufferlnt	Buffers de dados armazenados como valores int (Java 2)
DataBufferShort	Buffers de dados armazenados como valores short (Java 2)
DirectColorModel	Modelo de cor no qual os valores de pixel RGB são representados diretamente em cada pixel
Fi1teredlmageSource	Objetos ImageProducer que podem pegar uma imagem e o objeto ImageFi 1 ter e produzir uma imagem para um objeto ImageConsumer
ImageFilter	Filtros que pegam dados de imagem de objetos ImageProducer e passam versões modificadas para objetos ImageConsumer
IndexColorModel	Modelo de cor no qual os valores em pixel são índices para mapas de cor fixos em um objeto ColorSpace
Kernel	Matrizes descrevendo como um pixel e os pixels circunvizinhos afetam seu valor em uma operação de filtragem (Java 2)
LookupOp	Operações de pesquisa de uma imagem em outra (Java 2)
LookupTable	Tabelas de pesquisa contendo dados de um ou mais canais de setor ou estruturas de componente de imagem (Java 2)
MemoryImageSource	Objetos ImageProducer que obtêm imagens da memória
MultibandPacked SampleModel	Comportamento para fazer amostragem de imagens que armazenam pixels em elementos de array de dados específicos no primeiro banco DataBuffer (Java 2)
NearestNei ghborAffi ne TransformOp	Transformações affine do vizinho mais próximo para transformar uma imagem ou varredura em outra (Java 2)
PackedColorModel	Modelo de cor no qual os valores em pixel são armazenados diretamente em bits de um pixel inteiro (Java 2)
PixelGrabber	Objetos ImageConsumer que recuperam subconjuntos dos pixels em uma imagem

RGBImageFilter	Filtros que pegam os dados de imagem RGB de objetos ImageProducer e passam versões modificadas para objetos ImageConsumer
Raster	Representação de varredura de dados de imagem (Java 2)
Repli cateScaleFi1 ter	Filtros que mudam a escala de imagens (Java 1.1)
RescaleOp	Mudança de escala, pixel por pixel, de imagens ou varreduras (Java 2)
ShortLookupTable	Tabelas de pesquisa contendo dados short de um ou mais canais de setor ou estruturas de componente de imagem (Java 2)
SingleBandPacked SampleModel	Comportamento para fazer a amostragem de imagens que armazenam pixels que não abrangem elementos de dados (Java 2)
ThresholdOp	Conversão de limiar de imagens ou varreduras (Java 2)
Ti 1eChangeMulti caster	Comportamento que implementa a interface TileChangeListener (Java 2)
WritableRaster	Representação de varredura gravável de dados image (Java 2)

java.awt.peer

Um pacote oculto que fornece classes de janelas específicas de plataforma com interfaces independentes de plataforma para implementar. Os usuários dessas interfaces não precisam saber qual sistema de janelas da plataforma essas classes não-hierárquicas estão implementando no momento.

Cada classe AWT que herda de Component ou MenuComponent possui uma classe não-hierárquica correspondente. Como elas fornecem um comportamento tão semelhante às suas correspondentes AWT e não se destinam a ser chamadas diretamente pelos desenvolvedores, foram omitidas neste apêndice.

java.awt.print

Pacote de impressão introduzido na linguagem Java 2.

Interfaces

PageContext	Métodos para fornecer informações sobre uma página que faz parte de um Pri nterJob
Pageabl e	Métodos para representar um conjunto de páginas a serem impressas
Pri ntabl e	Métodos para imprimir em conjunto com objetos PaqeFormat

Classes

Book	Lista de páginas a serem impressas
PageFormat	Tamanho e orientação de páginas para impressão
Paper	Características físicas do papel de impressão
PeekGraphics	Classe de suporte à impressão de imagens gráficas bidimensionais
PrinterJob	Um trabalho de impressão
ProxyGraphics2D	Contexto gráfico usado no processo de impressão

javax.swing

Trata-se de um pacote introduzido na linguagem Java 2, que fornece um novo conjunto de componentes de interface gráfica com o usuário e outros aprimoramentos de interface. Os componentes do Swing podem ter, automaticamente, a "aparência e comportamento" de qualquer plataforma (como Windows, Macintosh e Solaris). As classes desse pacote utilizam automaticamente um pacote específico para a aparência e o comportamento escolhidos, incluindo javax.swing.basic, javax.swing. jTf, javax..swing.motif, javax.swing.mui ti e javax.swing.plaf.

O conjunto de componentes do Swing incluem duplicatas de todos os componentes AWT da Java 1.1 e muitos componentes adicionais. Ele compreende um pacote javax.swing e os seguintes subpacotes:

- javax.swing.basic
- javax.swing.beaninfo
- javax.swing.border
- javax.swing.event
- javax.swing.jlf
- javax.swing.motif
- javax.swing.mui ti
- javax.swing.plaf
- javax.swing.table
- javax.swing.target
- javax.swing.tree
- javax.swing.text
- javax.swing.undo

Interfaces

Action	Métodos para associar diferentes componentes com funcionalidade de comando de ação
BoundedRangeModel	Métodos para representar os dados em um componente de barra de rolagem ou controle deslizante
ButtonModel	Métodos para representar o estado de um botão
Cel1 Edi tor	Métodos para editar tabelas em componentes como caixas de combinação e listas de escolha
ComboBoxEditor	Métodos para editar caixas de combinação
ComboBoxModel	Métodos para representar os dados em uma caixa de combinação
DesktopManager	Métodos para implementar comportamento de aparência e comportamento para JDesktopPane
Icon	Pequena imagem para uso em componentes
JComboBox. KeySelecti onManager	Método para definir um KeySelectionManager
ListCellRenderer	Métodos para pintar as células em um JList
ListModel	Métodos para representar os dados em uma lista
ListSelectionModel	Métodos para representar a seleção atual em uma lista
Scrol 1 PaneConstants	Constantes para uso com objetos JScrol 1 Pane
Scrol 1 abi e	Métodos para fornecer informações para um container de rolagem
Si ngl eSel ecti onModel	Métodos para representar a única seleção possível em uma lista
SwingConstants	Constantes para uso com componentes Swing
UIDefaults.ActiveValue	Métodos para armazenar uma entrada na tabela de padrões
UIDefaul ts. LazyVal ue	Métodos para armazenar uma entrada na tabela de padrões que não é construída até ser pesquisada
WindowConstants	Constantes para uso com componentes JWindow

Classes

AbstractAction	Comportamento para objetos de ação
AbstractButton	Comportamento comum para componentes de botão. (Seus métodos getLabel ( ) e setLabel ( ) foram desaprovados a partir da Java 2.)
AbstractButton. ButtonChangeLi stener	Receptor de eventos de alterações no estado do botão
AbstractListModel	Modelo de dados que preenchem um componente de lista
BorderFactory	Produtor de objetos de borda padrão
Box	Contêiner usando o gerenciador de layout de caixa
Box.Filler	Contêiner de caixa sem modo de visualização
BoxLayout	Gerenciador de layout de caixa
ButtonGroup	Classe para permitir que apenas um botão em um grupo seja selecionado
CeilRendererPane	Classe inserida entre renderizadores de célula e componentes para bloquear chamadas de repaint( ) e invalidate( )
DefaultBounded RangeModel	Modelo de intervalo limitado genérico
DefaultButtonModel	Versão padrão do modelo de dados de um botão
DefaultCell Editor	Editor padrão para tabela e células de árvore
DefaultDesktopManager	Gerenciador de área de trabalho genérico
DefaultFocusManager	Gerenciador de foco genérico
DefaultListModel	Versão padrão dos dados de um componente de lista
DefaultList SelectionModel	Versão padrão dos itens selecionados em uma lista
Defaul tSingle SelectionModel	Versão padrão do único item selecionado em uma lista
FocusManager	Gerenciador de foco de componente
GrayFilter	Imagem que desativa um componente, adaptando-o ao sistema de escala de cinza
Imagelcon	Implementação da interface Icon
JApplet	Janela de applet Swing
JButton	Botão Swing em que se pode dar um clique de mouse

JCheckBox	Caixa de seleção Swing
JCheckBoxMenuItem	Item de caixa de seleção Swing em um menu suspenso
JColorChooser	Caixa de diálogo de seleção de cor Swing
JCotnboBox	Caixa de listagem e campo de texto combinados Swing
JComponent	Classe-raiz de componente Swing
JDesktopIcon	ícone Swing para uma área de trabalho
JDesktopPane	Painel de área de trabalho Swmg
JDialog	Janela de diálogo Swing
JDirectoryPane	Painel de diretório Swing
JEditorPane	Painel de editor Swing
JFileChooser	Caixa de diálogo de seleção de arquivos Swii
JFrame	Janela de quadro Swing
JInternalFrame	Quadro interno Swing
JLabel	Rótulo Swing
JLayeredPane	Contêiner de painel em camadas Swing
JList	Lista de escolha Swing
JMenu	Menu suspenso Swing
JMenuBar	Barra de menus Swing em um componente c janela
JMenuItem	Item de menu Swing em um menu suspenso
JOptionPane	Contêiner de painel de opção Swing
JPanel	Contêiner de painel Swing
JPasswordField	Campo de texto Swing com caractere para ocultar senhas
JPopupMenu	Menu instantâneo Swing
JProgressBar	Componente de barra de andamento Swing
JRadioButton	Botão de rádio Swing
JRadioButtonMenuItem	Botão de rádio Swing em um menu suspensc
JRootPane	Contêiner de painel-raiz Swing
JScrollBar	Barra de rolagem Swing
JScrollPane	Contêiner de painel de rolagem Swing
JSeparator	Componente separador de menu Swing
JSlider	Controle deslizante Swing
JSplitPane	Contêiner de painel de divisão Swing

JTabbedPane	Contêiner de painel com guias Swing
JTable	Tabela Swing
JTextArea	Campo de texto de várias linhas Swing
JTextField	Campo de texto de uma linha Swing
JTextPane	Contêiner de painel de texto Swing
JToggleButton	Botão de comutação Swing
JToolBar	Barra de ferramentas Swing
JToolTip	Componente de balão de descrição instantâneo Swing
JTree	Arvore de diretório Swing
JViewport	Contêiner de visor Swing
JWindow	Contêiner de janela Swing
KeyStroke	Uma tecla sendo digitada com o teclado
OverlayLayout	Gerenciador de layout de overlay
ProgressMonitor	Comportamento para monitorar o andamento de uma operação
ProgressMonitor InputStream	Comportamento para monitorar o andamento de um fluxo de entrada
RepaintManager	Gerenciador de exibição de componente
ScrollPaneLayout	Gerenciador de layout de painel de rolagem
SizeRequirements	Medidas de tamanho para uso dos gerenciadores de layout
SpringLayout	Gerenciador de layout spring
SwingUti1i ti es	Métodos utilitários para uso dentro do Swing
Syntheticlmage	Classe auxiliar para computação de imagens sintéticas
Timer	Evento disparado em intervalos definidos
ToolTipManager	Gerenciador de componentes de balão de descrição
UIDefaults	Valores padrão para componentes Swing
UIManager	Classe para controlar a aparência e o comportamento atuais
ViewportLayout	Gerenciador de layout de visor

javax.swing.event

Trata-se de um subpacote Swing introduzido na linguagem Java 2 para oferecer suporte aos novos eventos que são disparados por componentes Swing.

Interfaces

AncestorListener	Métodos para anotar uma alteração em um componente Swing ou em suas superclasses
ChangeListener	Métodos para responder às alterações de texto
DocumentEvent	Métodos para responder às alterações de documento
DocumentListener	Métodos para responder às alterações de documento
TableColumnModelListener	Métodos para responder às alterações em uma coluna de tabela
TableModelListener	Métodos para responder às alterações em uma tabela
TreeExpansionListener	Métodos para responder à expansão de uma árvore de diretório
TreeModelListener	Métodos para responder às alterações em uma árvore de diretório
TreeSelectionListener	Métodos para responder às seleções dentro de uma árvore de diretório

Classes

AncestorEvent	Eventos de quando um JComponent ou suas superclasses mudam
ChangeEvent	Eventos de mudança de texto
DragEvent	Eventos de arraste do mouse
EventListenerList	Lista de receptores de evento
ListSelectionEvent	Eventos de quando uma seleção de lista é feita
MenuEvent	Eventos de menu
TableColumnModelEvent	Eventos de mudança de coluna de tabela
TableModelEvent	Eventos de mudança de tabela
TreeExpansionEvent	Eventos de expansão de tabela
TreeModelEvent	Eventos de mudança de árvore
TreeSelectionEvent	Eventos de seleção de árvore

javax.swing.undo

Trata-se de um subpacote Swing introduzido na linguagem Java 2, que oferece suporte a comandos para desfazer edições.

Interfaces

StateEditable	Métodos para objetos que podem ter seu estado desfeito
UndoableEdit	Métodos para representar uma edição concluída que pode ser desfeita

Classes

AbstractUndoableEdit	Comportamento implementando a interface UndoableEdit
CompoundEdit	Comportamento para combinar várias edições que podem ser desfeitas, em apenas uma, maior
StateEdit	Edição geral de objetos que mudam de estado
UndoManager	Receptor de eventos de edição que pode ser desfeita
UndoableEditEvent	Eventos de edição que pode ser desfeita

java.beans

Esse é um pacote introduzido na linguagem Java 1.1, que cria componentes de software reutilizáveis, chamados beans, e programas que os manipulam.

Interfaces

Appletlnitializer	Métodos para permitir que applets bean sejam inicializados corretamente (Java 2)
Beanlnfo	Métodos para um bean fornecer informações sobre si mesmo
Customizer	Métodos que definem a sobrecarga do fornecimento de um editor de configuração gráfico para um bean
DesignMode	Métodos para oferecer suporte a comportamento bean, quando usado em um ambiente de desenvolvimento (Java 2)
PropertyChangeLi stener	Métodos para responder às alterações em uma propriedade de contorno
PropertyEditor	Métodos para oferecer suporte a interfaces gráficas com o usuário para edição de valores de propriedade de determinado tipo
VetoableChangeLi stener	Métodos para responder às alterações em uma propriedade restrita

Visibility	Métodos para determinar se um bean exige uma interface gráfica com o usuário e se há alguma disponível
VisibilityState	Métodos para determinar se uma interface gráfica com o usuário pode ser usada (Java 2)

Classes

BeanDescriptor	Informações globais sobre um bean
Beans	Comportamento de propósito geral para controlar beans
EventSetDescriptor	Grupo de eventos que um bean é capaz de gerar
FeatureDescriptor	Origem de todas as classes descritoras de bean
IndexedProperty Descriptor	Propriedade que funciona como um array e oferece acesso de leitura/escrita indexada
Introspector	Comportamento para analisar um bean e determinar suas propriedades públicas, métodos e eventos
MethodDescriptor	Comportamento para descrever método pelo qual o bean pode ser acessado externamente
ParameterDescri ptor	Comportamento para fornecer informações adicionais sobre os parâmetros de um bean
PropertyChangeEvent	Evento de quando uma propriedade de contorno ou restrita é alterada
PropertyChangeSupport	Classe auxiliar para gerenciamento de receptores de propriedades de contorno e restritas
PropertyDescri ptor	Propriedade e métodos de acesso que descrevem se um bean é limitado ou restrito
PropertyEditorManager	Comportamento para localizar editores de propriedade de nomes de tipo específicos
PropertyEdi torSupport	Classe auxiliar para a criação de editores de propriedade
SimpleBeanlnfo	Auxiliar para fornecer classes Beanlnfo
VetoableChangeSupport	Classe auxiliar para beans que oferecem suporte a propriedades restritas

java.bean.beancontext

Esse é um pacote introduzido na linguagem Java 2 para fornecer um mecanismo de serviços gerais para beans.

Interfaces

BeanContext	Métodos para fornecer informações sobre o estado do ambiente de um bean
BeanContextChild	Métodos para indicar o ambiente de runtime de um bean
BeanContextMembership Listener	Métodos para responder a eventos de contexto de bean

Classes

BeanContextAddedEvent	Eventos gerados quando um ou mais descendentes são acrescentados
BeanContextEvent	Eventos gerados quando o estado de BeanContext muda
BeanContext MembershipEvent	Eventos enviados a todos os membros quando o estado muda
BeanContextRemovedEvent	Eventos gerados quando um ou mais descendentes são removidos
BeanContextSupport	Implementação da interface BeanContext

java.io

Esse pacote fornece acesso de entrada e saída para arquivos e outra mídia de fluxo.

Interfaces

DataInput	Métodos para ler fluxos de entrada digitados independentes de máquina
DataOutput	Métodos para escrever fluxos de saída digitados independentes de máquina
Externalizable	Métodos para escrever e ler o conteúdo de um objeto com um fluxo (Java 1.1)
FilenameFilter	Métodos para filtrar nomes de arquivo
ObjectInput	Métodos para ler objetos (Java 1.1)
ObjectInputValidation	Métodos para validar um objeto (Java 1.1)
ObjectOutput	Métodos para escrever objetos (Java 1.1)
Replaceable	Métodos de uma classe que pode ser serializada para substituir um objeto por outro, quando escrito em um fluxo (Java 2)
Resolvable	Métodos de uma classe que pode ser serializada para substituir um objeto por outro, quando lido de um fluxo (Java 2)
Serializable	Tag para informar que essa classe pode ser serializada (Java 1.1)

Classes

BufferedlnputStream	Um fluxo de entrada colocado em buffer
BufferedOutputStream	Um fluxo de saída colocado em buffer
BufferedReader	Leitor colocado em buffer (Java 1.1)
BufferedWriter	Escritor colocado em buffer (Java 1.1)
ByteArraylnputStream	Fluxo de entrada de um array de bytes
ByteArrayOutputStream	Fluxo de saída em um array de bytes. (Seu método toString( ) foi desaprovado a partir da Java 1.1.)
CharArrayReader	Leitor de um array de caracteres (Java 1.1)
CharArrayWriter	Escritor em um array de caracteres (Java 1.1)
DatalnputStream	Comportamento para ler tipos primitivos Java (como int, char e boolean) de um fluxo, de modo independente de máquina. (Seu método readLine( ) foi desaprovado a partir da Java 1.1.)
DataOutputStream	Comportamento para escrever tipos de dados primitivos Java (como int, char e boolean) em um fluxo, de modo independente de máquina
File	Representa um arquivo no sistema de arquivos do host
FileDescriptor	Comportamento para conter o descritor de arquivos tipo UNIX de um arquivo ou socket
FilelnputStream	Fluxo de entrada de um arquivo; construído usando-se um nome de arquivo ou um descritor
FileOutputStream	Fluxo de saída de um arquivo; construído usando-se um nome de arquivo ou um descritor
FileReader	Leitor de um arquivo; construído usando-se um nome de arquivo ou um descritor (Java 1.1)
FileWriter	Escritor em um arquivo; construído usando-se um nome de arquivo ou um descritor (Java 1.1)
FilterlnputStream	Classe abstrata que fornece um filtro para fluxos de entrada e para dar funcionalidade de fluxo, como o uso de buffers
FilterOutputStream	Classe abstrata que fornece um filtro para fluxos de saída e para dar funcionalidade de fluxo, como o uso de buffers

FilterReader	Classe que fornece um filtro para leitores e para adicionar funcionalidade, como o uso de buffers (Java 1.1)
FilterWriter	Classe que fornece um filtro para escritores e para a inclusão de funcionalidade, como o uso de buffers (Java 1.1)
InputStream	Classe abstrata que representa um fluxo de entrada de bytes; a origem de todos os fluxos de entrada nesse pacote
InputStreamReader	Leitor ligando fluxos de byte a fluxos de caractere (Java 1.1)
Li neNumberlnputStream	Fluxo de entrada que controla o número de linhas (desaprovado a partir da Java 2)
LineNumberReader	Leitor de fluxo de entrada de caracteres colocado em buffer que controla o número de linhas (Java 1.1)
ObjectlnputStream	Classe que anula a serialização de dados e objetos (Java 1.1; seu método readLine( ) foi desaprovado a partir da Java 2.)
ObjectOutputStream	Classe que serializa dados e objetos (Java 1.1)
ObjectStreamClass	Descritor de classes que podem ser serializadas (Java 1.1)
OutputStream	Classe abstrata que representa um fluxo de saída de bytes; a origem de todos os fluxos de saída nesse pacote
OutputStreamWri ter	Uma ponte entre fluxos de byte e caractere (Java 1.1)
PipedlnputStream	Fluxo de entrada colocado em pipe, que deve estar conectado a um PipedOutputStream para ter utilidade
PipedOutputStream	Fluxo de saída colocado em pipe, que deve estar conectado a um PipedlnputStream para ter utilidade no fornecimento de comunicação segura entre threads
PipedReader	Leitor colocado em pipe, que deve estar conectado a um Pi pedWri ter para ser útil (Java 1.1)
PipedWriter	Escritor colocado em pipe, que deve estar conectado a um Pi pedReader para ser útil (Java 1.1)

PrintStream	Fluxo de saída para impressão, usado por System.out.println( ). (Seus construtores PrintStream( ) foram desaprovados a partir dajava 1.1.)
PrintWriter	Escritor para impressão (Java 1.1)
PushbacklnputStream	Fluxo de entrada com um buffer push-back
PushbackReader	Leitor com buffer push-back (Java 1.1)
RandomAccessFile	Comportamento para fornecer acesso aleatório a um arquivo construído a partir de nomes de arquivo, descritores ou objetos
Reader	Classe abstrata que representa um fluxo de caracteres de entrada; a origem de todos os leitores nesse pacote (Java 1.1)
SequencelnputStream	Comportamento para converter uma seqüência de fluxos de entrada em um único fluxo de entrada
Serial izablePermission	Comportamento para manipular controle de acesso que pode ser serializado Qava 2)
StreamTokenizer	Comportamento para converter um fluxo de entrada em uma série de sinais (tokens) individuais. (Seu construtor StreamTokenizer( ) foi desaprovado a partir dajava 1.1.)
StringBufferlnput Stream	Um fluxo de entrada de um objeto Stri ng (desaprovado a partir da Java 2)
StringReader	Leitor de um objeto String (Java 1.1)
StringWriter	Escritor para um objeto String (Java 1.1)
Writer	Classe abstrata representando um fluxo de caracteres de saída; a origem de todos os escritores nesse pacote (Java 1.1)

java.lang

Esse é o pacote que representa o núcleo da linguagem Java.

Interfaces

Cloneable	Métodos para indicar que um objeto pode ser copiado ou clonado
Comparable	Métodos para impor uma ordem em objetos de uma classe (Java 2)
Runnable	Métodos para executar classes como threads

Classes

Boolean	Wrapper de objeto para valores booleanos
Byte	Wrapper de objeto para valores byte (Java 1.1)
Character	Wrapper de objeto para valores char. (Seus métodos isJavaLetter( ), isJavaLetterOrDigit( )eisSpace( ) foram desaprovados a partir da Java 1.1.)
Class	Representações de runtime de classes
ClassLoader	Comportamento abstrato para a manipulação do carregamento de classes. (Seu método defineClass( ) foi desaprovado a partir da Java 1.1.)
Compiler	Classe de sistema que dá acesso ao compilador Java
Double	Wrapper de objeto para valores doubi e
Float	Wrapper de objeto para valores f 1 oat
Integer	Wrapper de objeto para valores i nt
Long	Wrapper de objeto para valores 1 ong
Math	Classe utilitária para operações matemáticas
Number	Superclasse abstrata de todas as classes numéricas (como Integer e Float)
Object	Classe de objeto genérica no topo da hierarquia de herança
Package	Informações sobre a versão da implementação e da especificação do pacote Java (Java 2)
Process	Comportamento abstrato para processos, como aqueles gerados por métodos da classe System
Runtime	Acesso ao runtime Java. (Seus métodos getLocalizedInputStream( )e getLocal izedOutputStream( ) foram desaprovados a partir da Java 1.1.)
RuntimePermission	Comportamento para fornecer controle de acesso de runtime (Java 2)
SecurityManager	Comportamento abstrato para a implementação de planos de segurança. (A partir da Java 2, sua variável inCheck e vários métodos foram desaprovados: getInCheck( ), classDepth( ), classLoaderDepth( ), inClass( ) e inClassLoader( ).)

Short	Wrapper de objeto para valores short (Java 1.1)
String	Strings de caracteres. (Dois construtores String( ) e um método getBytes( ) foram desaprovados a partir da Java 1.1.)
StringBuffer	Strings mutantes
System	Acesso ao comportamento no sistema da linguagem Java, fornecido de modo independente de plataforma. (Seu método getEnv( ) foi desaprovado a partir da Java 1.1.)
Thread	Métodos para o gerenciamento de threads e classes que são executadas em threads. (Seus métodos stop( ),suspend( ) e resume( ) foram desaprovados a partir da Java 2.)
ThreadDeath	Classe de objetos levantados quando um thread é terminado de forma assíncrona
ThreadGroup	Um grupo de threads. (Seus métodos stop( ), suspend( ) e resume( ) foram desaprovados a partir da Java 2.)
ThreadLocal	Comportamento para fornecer variáveis individualizadas independentemente para threads (Java 2)
Throwable	Classe de exceção genérica
Void	Wrapper de objeto para tipos void (Java 1.1)

java.lana.ref

Trata-se de um pacote introduzido na linguagem Java 2 para permitir que as referências a objeto sejam encapsuladas e examinadas como qualquer outro objeto.

Classes

CachedReference	Objetos de referência fraca colocados em cache
GuardedReference	Objetos de referência protegida
PhantomReference	Objetos de referência fantasma
Reference	Abstração principal para objetos de referência
ReferenceQueue	A fila a que os objetos de referência são atribuídos pelo Garbage Collector
WeakReference	Objetos de referência fraca

JAVA. LANG.REFLECT

Trata-se de um pacote introduzido na linguagem Java 1.1 para oferecer suporte à reflexão, um meio de encontrar informações a respeito das classes carregadas (como seus atributos e comportamento).

Interfaces

Member	Métodos para localizar informações sobre um membro

Classes

AccessibleObject	Classe de base para objetos Fi ei d, Method e Constructor (Java 2)
Array	Métodos para criar e acessar arrays dinamicamente
Constructor	Métodos para localizar informações sobre construtores e acessá-los
Field	Métodos para localizar informações sobre variáveis e acessá-las
Method	Métodos para localizar informações sobre métodos e acessá-los
Modifier	Decodificador para modificadores de acesso a classe e membro
ReflectPermission	Comportamento para fornecer controle de acesso para operações de reflexão (java 2)
[T2java.math	Trata-se de um pacote introduzido na linguagem Java 1.1 para conter números de precisão arbitrária.

Classes

BigDecimal	Um número muito grande em ponto flutuante
Biglnteger	Um inteiro muito grande

java.net

Pacote para realizar operações de rede, como manipulação de sockets e URL.

Interfaces

ContentHandlerFactory	Métodos para criar objetos ContentHandler
FileNameMap	Métodos para fazer o mapeamento entre nomes de arquivo e tipos MIME (Java 1.1)
SocketImplFactory	Métodos para criar implementações de socket
URLStreamHandlerFactory	Métodos para criar objetos URLStreamHandl er

Classes

Authenticator	Objeto que pode obter autenticação de rede (Java 2)
ContentHandler	Comportamento abstrato para ler dados de uma conexão de URL e construir o objeto local apropriado, baseado nos tipos MIME
DatagramPacket	Pacote de datagrama (UDP)
DatagramSocket	Socket de datagrama
DatagramSocketlmpl	Classe de base abstrata para sockets de datagrama e multicoerção (Java 1.1)
HttpURLConnection	Conexão que pode manipular o protocolo HTTP (Java 1.1)
InetAddress	Representação de objeto de um host da Internet (nome do host, endereço IP)
JarURLConnection	Conexão de URL com um arquivo JAR ou entrada em um arquivo JAR (Java 2)
MuiticastSocket	Socket no lado do servidor com suporte a transmissão de dados para vários sockets clientes (Java 1.1)
NetPermissiori	Comportamento para fornecer controle de acesso de rede (Java 2)
PasswordAuthenticatio n	Nome de usuário e senha para uso do Authenticator (Java 2)
ServerSocket	Socket no lado do servidor
Socket	Um socket. (Dois construtores Socket ( ) foram desaprovados a partir da Java 1.1.)
Socketlmpl	Classe abstrata para implementações de socket específicas
SocketPermission	Comportamento para fornecer controle de acesso a uma rede por sockets (Java 2)
URL	Representação de objeto de um URL
URLClassLoader	Carregador de classe e arquivo repositório Jar a partir de um caminho de busca de URL (Java 2)
URLConnection	Comportamento abstrato para um socket que pode manipular vários protocolos baseados n; Web (http, ftp etc.)
URLEncoder	Transforma strings para o formato x-www-form-urlencoded

URLStreamHandler	Classe abstrata para o gerenciamento de fluxos no objeto a que os URLs fazem referência

java.rmi

Pacote introduzido na linguagem Java 1.1 para criar aplicativos distribuídos (de Java para Java), em que os métodos de objetos Java remotos podem ser chamados a partir de outras máquinas virtuais Java, possivelmente em diferentes hosts.

Interfaces

Remote	Métodos para identificar todos os objetos remotos

Classes

MarshalledObject	Fluxo de bytes com uma representação serial de um objeto fornecida para seu construtor Java2) '
Naming	Comportamento para obter referências para objetos remotos baseadas na sintaxe de URL
RMISecuri tyManager	Métodos definindo o plano de segurança RMI Stub para aplicativos (não para applets)

java.rmi.activation

Esse pacote, introduzido na linguagem Java 2, oferece suporte às referências persistentes a objetos remotos e à ativação automática de objetos usando essas referências.

Classes

Activatable	Objetos remotos que exigem acesso persistente com o passar do tempo e podem ser ativados
ActivationDesc	Descritor com as informações necessárias para ativar objetos
ActivationGroup	Comportamento para agrupar e monitorar objetos que podem ser ativados
ActivationGroupDesc	Descritor com informações para criar e recriar grupos de ativação
ActivationGroupID	Identificador para um grupo de ativação registrado
ActivationlD	Identificadores para objetos que podem ser ativados

java.rmi.dgc

Esse pacote, introduzido na linguagem Java 1.1, oferece suporte ao algoritmo de coleta de lixo distribuída.

Interfaces

DGC	Métodos para fazer a limpeza de conexões de clientes não-utilizados

Classes

Lease	Contém um identificador de VM exclusivo e a duração da validade
VMID	Comportamento para manter VMID exclusivo em todas as máquinas virtuais Java

java.rmi.reqistry

Trata-se de um pacote introduzido na linguagem Java 1.1 para manipular os Remote Method Invocation Registries.

Interfaces

Regi stry	Métodos para obter o Registry de diferente hosts
RegistryManager	Métodos para fazer a interface com a implementação privativa. (Seus métodos registryStub( ) e registrylmpl ( ) foram desaprovados a partir da Java 2.)

Classes

LocateRegistry	Comportamento para obter o Registro de partida de um host em particular

java.rmi.server

Trata-se de um pacote introduzido na linguagem Java 1.1 para o lado senador do Remote Method Invocation.

Interfaces

LoaderHandler	Métodos para manipular o carregamento da chamada
RMIFailureHandler	Métodos para manipulação quando o runtime RMI é incapaz de criar um Socket ou um ServerSocket
RemoteCall	Métodos para a implementação de chamadas de um objeto remoto
RemoteRef	Representa uma alça para um objeto remoto
ServerRef	Representa a alça no lado do servidor de uma implementação de objeto remoto
Skeleton	Representa a entidade no lado do servidor que transmite chamadas para a implementação de objeto remoto real
Unreferenced	Métodos para receber notificação de quando não existem mais referências remotas para eles

Classes

LogStream	Fornece um mecanismo de registro de erros que possivelmente têm interesse para quem está monitorando o sistema.
ObjID	Usado para identificar objetos remotos exclusivamente em uma VM.
Operation	Contém uma descrição de um método Java.
RMIClassLoader	Fornece métodos estáticos para o carregamento de classes através da rede.
RMISocketFactory	Usado pelo runtime RMI para obter sockets de cliente e servidor para chamadas RMI.
RemoteObject	Fornece a semântica remota de Object, implementando métodos para hashCode, equals e toString.

RemoteServer	Superclasse para todas as implementações de servidor, que fornece a estrutura para oferecer suporte a uma ampla gama de semânticas de referência remota.
RemoteStub	Objetos de stub são substitutos que oferecem suporte exatamente ao mesmo conjunto de interfaces remotas definidas pela implementação real do objeto remoto.
SocketType	Tipo que pode ser passado para o objeto RMISocketFactory quando da criação de socket ou de socket de servidor (Java 2).
UID	Abstração para a criação de identificadores que são exclusivos com relação ao host em que são gerados.
UnicastRemoteObject	Define um objeto remoto não-duplicado, cujas referências são válidas apenas enquanto o processo servidor está em vigor.

ava.security

Trata-se de um pacote introduzido na linguagem Java 1.1 para implementar certificados e assinaturas digitais em componentes Java.

Interfaces

Certificate	Métodos para gerenciar um certificado, incluindo codificação e decodificação (desaprovado a partir da Java 2)
Guard	Métodos para proteger o acesso a outro objeto (Java 2)
Key	Métodos para definir a funcionalidade compartilhada por todos os objetos-chave (Java 2)
Principal	Representa o componente principal de um certificado
PrivateKey	Métodos para agrupar e fornecer segurança de tipo para todas as interfaces de chave privativa (Java 2)
PublicKey	Métodos para agrupar e fornecer segurança de tipo para todas as interfaces de chave pública (Java 2)

Classes

AccessControlContext	Ambiente usado para tomar decisões de controle de acesso (Java 2)
AccessController	Provedor de acesso controlado baseado em um plano de segurança (Java 2)
AlgorithmParameter Generator	Comportamento para gerar parâmetros usados com determinado algoritmo (Java 2)
Algori thmParameter GeneratorSpi	Interface de provedor de serviços para o gerador de parâmetro de algoritmo (Java 2)
Algori thmParameters	Representação de parâmetros de criptografia (Java 2)
Algori thmParametersSpi	Interface de provedor de serviços para parâmetros de algoritmo (Java 2)
BasicPermission	Comportamento para fornecer as permissões básicas para controle de acesso (Java 2)
CodeSource	Extensão URL para incluir a chave pública exigida para autenticar o código originário (Java 2)
DigestlnputStream	Representa um fluxo de entrada que possui uma compilação de mensagem associada
DigestOutputStream	Representa um fluxo de saída que possui uma compilação de mensagem associada
GuardedObject	Objeto usado para proteger o acesso a outro objeto (Java 2)
Identity	Métodos para gerenciar identidades (podem ser objetos como pessoas, empresas ou organizações) que podem ser autenticadas usando-se uma chave pública. (Os métodos addCertificate( ), removeCertificate( ) ecertifi cates ( ) foram desaprovados a partir da Java 2.)
IdentityScope	Métodos para a definição do escopo de uma identidade, incluindo o nome da identidade, sua chave e certificados associados
Key	Classe abstrata representando uma chave de criptografia
KeyFactory	Conversor de chaves para especificações de chave (Java 2)
KeyFactorySpi	Interface de provedor de serviços para geração de chaves (Java 2)
KeyPair	Um simples portador para um par de chaves
KeyPairGenerator	Gerador para produzir pares de chave (Java 2)

KeyPai rGeneratorSpi	Interface de provedor de serviços para geradores de par de chaves (Java 2)
KeyStore	Conjunto presente na memória de chaves privativas e encadeamentos de certificado associados (Java 2)
MessageDigest	Métodos que fornecem a funcionalidade de um algoritmo de compilação de mensagem
MessageDigestSpi	Interface de provedor de serviços de compilações de mensagem (Java 2)
Permission	Representação de acesso a um recurso do sistema (Java 2)
Permi ssi onCol1ecti on	Conjunto de objetos Permission (Java 2)
Permissions	Conjunto heterogêneo de objetos Permission (Java 2)
Policy	Representação de plano para runtime Java (Java 2)
Provi der	Representa um provedor de pacote de segurança (SPP) para a API JavaSecurity
SecureClassLoader	Extensão de Cl assLoader para classes com fontes de código e conjuntos signer (Java 2)
SecureRandom	Gera um número aleatório
Security	Métodos para gerenciar SPPs. (Os métodos setParameter( ) e TD]getParameter() foram desaprovados a partir da Java 2.)
SecurityPermission	Comportamento para fornecer controle de acesso de segurança (Java 2)
Signature	Fornece o algoritmo para assinaturas digitais
SignatureSpi	Interface de provedor de serviços para objetos Signature (Java 2; seus métodos engineGetParameter( ) e engineSetParameter( ) foram desaprovados a partir da Java 2.)
SignedObject	Objetos de runtime autênticos, cuja integridade não pode ser comprometida sem detecção (Java 2)
Signer	Representa uma identidade que também pode assinar

java.security.acl

Trata-se de um pacote que fornece a interface para uma estrutura de dados que protege o acesso aos recursos.

Interfaces

Ad	Interface que representa uma ACL (Access Control List), que é uma estrutura de dados utilizada para proteger o acesso aos recursos
AclEntry	Métodos que permitem aos programadores incluir, remover ou definir permissões para os objetos Principal de cada ACLEntry na ACL
Group	Métodos que permitem aos programadores incluir ou remover um membro do grupo de objetos Principal
Owner	Representa o proprietário de uma ACL
Permission	Representa o tipo de acesso concedido a um recurso, como um Principal na ACL

java.security.cert

Esse é o pacote introduzido na linguagem Java 2 para certificação de identidade.

Interfaces

X509Extension	Extensões definidas para certificados X.509 v3 e v2 Certificate Revocation Lists

Classes

Certificate	Certificados de identidade com diferentes formatos e utilização comum
RevokedCertificate	Certificado revogado em uma Certificate Revocation List
X509CRL	X.509 Certificate Revocation List
X509Certificate	Certificados X.509

java.security.interfaces

Trata-se de um pacote introduzido na linguagem Java 1.1 para autenticação DSA.

Interfaces

DSAKey	Métodos usados para autenticar componentes, incluindo applets Java e controles ActiveX distribuídos através da Web
DSAKeyPairGenerator	Métodos para gerar pares de chave DSA
DSAParams	Métodos que permitem aos programadores obter a base, principal e secundário

DSAPrivateKey	Interface para uma chave privativa DSA
DSAPublicKey	Interface para uma chave pública DSA

java.security.spec

Este pacote foi introduzido na linguagem Java 2 para oferecer suporte à codificação.

Interfaces

AlgorithmParameterSpec	Métodos para especificar parâmetros de criptografia
KeySpec	Métodos para especificar uma chave de criptografia

Classes

DSAParameterSpec	Parâmetros usados com o algoritmo DSA
DSAPrivateKeySpec	Chave privativa DSA e parâmetros associados
DSAPublicKeySpec	Chave pública DSA e parâmetros associados
EncodedKeySpec	Chave pública ou privativa em formato codificado
PKCS8EncodedKeySpec	Codificação DER de chave privativa para o padrão PKCS #8
X509EncodedKeySpec	Codificação DER de chave pública ou privativa para o padrão X.509

java.sql

Trata-se de um pacote introduzido na linguagem Java 1.1 para estabelecer conexão com bancos de dados de back-end através de SQL (Structured Query Language).

Interfaces

CallableStatement	Métodos usados para executar procedures armazenadas e manipular vários conjuntos de resultados
Connection	Representa uma sessão com o banco de dados
DatabaseMetaData	Interface que permite aos programadores obter informações de alto nível a respeito do banco de dados
Driver	Métodos usados para estabelecer conexão com um banco de dados

PreparedStatement	Métodos para a execução de instruções SQL pré-compiladas
ResultSet	Métodos para a recuperação de valores de uma instrução SQL executada
ResultSetMetaData	Métodos que fornecem informações sobre os tipos e propriedades das colunas em um ResultSet
Statement	Usado para instruções SQL estáticas

Classes

Date	Fornece métodos para formatar e fazer referência a datas. (A partir da Java 2, um construtor Date( ) e vários métodos foram desaprovados: getHours( ), getMinutes( ), getSeconds( ), setHours( ), setMinutes( )e setSeconds( ).)
DriverManager	Permite o gerenciamento de um conjunto de drivers JDBC
DriverPropertyInfo	Fornece métodos para a obtenção de diferentes propriedades de um driver
Time	Fornece métodos para formatar e fazer referência a valores de tempo. (Vários métodos foram desaprovados a partir da Java 2:getDate( ),getDay( ),getMonth( ), getYear( ),setDate( ),setMonth( )e setYear( ).)
Timestamp	Wrapper que contém o valor TIMESTAMP da SQL. (Um construtor TimeStamp( ) foi desaprovado a partir da Java 2.)
Types	Define constantes que são usadas para identificar tipos SQL

java.text

Trata-se de um pacote que cria strings a partir de outros objetos, convertendo-os em outros objetos.

Interfaces

AttributeSet	Métodos para ler um conjunto de atributos (Java 2)
AttributeCharacter Iterator	Métodos para fazer a iteração através do texto e informações de atributo (Java 2)

CharacterIterator	Métodos para percorrer um string e retornar diversas informações a seu respeito
MutableAttributeSet	Métodos para acessar um conjunto de atributos mutantes (Java 2)

Classes

Annotation	Wrapper para valores de atributo de texto (Java 2)
AttributedString	Texto e informações relacionadas sobre os atributos (Java 2)
BreakIterator	Comportamento para localizar os limites de texto (Java 2)
ChoiceFormat	Métodos que permitem a anexação de formatos aos números
CollationElement Iterator	Iterador para percorrer os caracteres em um string internacional (Java 2)
CollatedString	Proporciona um modo de usar strings internacionais em uma tabela com hashing ou conjunto classificado
Collation	Permite a comparação de texto Unicode
`CollationElement Iterator`	Fornece um modo de fazer a iteração através de um string internacional
`CollationKey`	String de acordo com as regras específicas de um objeto Coll ator (Java 2)
`Collator`	Comparações de string específicas da localidade (Java 2)
`DateFormat`	Classe abstrata que inclui várias subclasses de formatação de data e hora
`OateFormatData`	Métodos para definir os dados de formatação de data e hora
`DateFormatSymbols`	Dados de formatação de data e hora locais (Java 2)
`DecimalFormat`	Métodos para formatar números
`DecimalFormatSymbols`	Símbolos utilizados por Decimal Format para formatar números (Java 2)
`FieldPosition`	Comportamento para identificar campos em saída formatada (Java 2)
`Format`	Classe de base para todos os formatos
`FormatStatus`	Usado para alinhar objetos formatados
`MessageFormat`	Métodos para criar mensagens concatenadas

NumberFormat	Classe abstrata para todos os formatos numéricos; inclui subclasses. Você pode usar seus métodos para formatar e analisar números
ParseStatus	Obtém o status da análise, quando você analisa um string com diferentes formatos
RuleBasedCollator	Cotejador de tabela simples, orientado a dados (Java 2)
SimpleDateFormat	Métodos para formatar uma data ou hora em um string
StringCharacter Iterator	Métodos para a iteração bidirecional através de um string

java.util

Trata-se de um pacote de classes e interfaces utilitárias, incluindo números aleatórios, propriedades de sistema e outras classes úteis.

Interfaces

Collection	Métodos para implementar coleções (Java 2)
Comparator	Métodos para ordenar um conjunto de objetos (Java 2)
Enumeration	Métodos para enumerar conjuntos de valores
EventListener	Métodos para captar eventos (Java 1.1)
Iterator	Métodos para fazer a iteração de uma coleção (Java 2)
List	Métodos para enumerar uma coleção ordenada (Java 2)
Listlterator	Métodos para fazer a iteração de listas em ambas as direções (Java 2)
Map	Métodos para fazer o mapeamento de chaves em valores (Java 2)
Observer	Métodos para permitir que as classes sejam objetos Observable
Set	Conjunto sem elementos duplicados (Java 2)

Classes

AbstractCollection	Implementação estrutural de uma coleção (Java 2)
AbstractList	Implementação estrutural de uma lista (Java 2)
AbstractMap	Implementação estrutural de um mapa (Java 2)

AbstractSequentialList	Implementação estrutural de uma lista sustentada por acesso seqüencial (Java 2)
AbstractSet	Implementação estrutural de um conjunto (Java 2)
ArrayList	Implementação de uma lista com array redimensionável (Java 2)
ArrayMap	Implementação de um mapa sustentado por uma lista de array (Java 2)
ArraySet	Implementação de um conjunto sustentado por uma lista de array (Java 2)
Arrays	Comportamento para classificação, pesquisa e outras manipulações de array (Java 2)
BitSet	Um conjunto de bits
Calendar	Calendário genérico (Java 1.1)
Collections	Comportamento para manipular ou retornar objetos Collection (Java 2)
Date	A data atual do sistema, assim como métodos para gerar e analisar datas. (A partir da Java 1.1, quatro construtores Date( ) e vários métodos foram desaprovados: UTC( ), parse( ), getYear( ),setYear( ),getMonth( ),setMonth( ),getDate( ),setDate( ),getDay( ),getHours( ),setHours( ), getMinutes( ), setMinutes( ), getSeconds( ), setSeconds( ), toLocaleString( ), toGMTString( )e getTimeZoneOffset( ).)
Dictionary	Classe abstrata que faz o mapeamento entre chaves e valores (superclasse de HashTable)
EventObject	Objeto de evento associado a outro objeto (Java 1.1)
GregorianCalendar	Representação do calendário gregoriano, o sistema A.C./D.C. utilizado pela maior parte do mundo (Java 1.1)
HashMap	Implementação de um mapa baseado em tabelas com hashing (Java 2)
HashSet	Implementação de um conjunto baseado em tabelas com hashing (Java 2)
Hashtable	Uma tabela com hashing
LinkedList	Implementação de uma lista encadeada com array redimensionável (Java 2)
ListResourceBundle	Fornecedor de recursos para um Local e (Java 1.1)

`Locale`	Descrição de uma localização geográfica (Java 1.1)
`Observable`	Classe abstrata para objetos que podem ser observados
`Properties`	Tabela com hashing que contém comportamento para definir e recuperar propriedades persistentes do sistema ou de uma classe
`PropertyPermissions`	Comportamento para controlar o acesso às permissões de propriedade (Java 2)
`PropertyResourceBundle`	Fornecedor de recursos que utiliza propriedades de um arquivo (Java 1.1)
`Random`	Utilitários para a geração de números aleatórios
`ResourceBundle`	Um conjunto de objetos relacionados a um Locale (Java 1.1)
`SimpleTimeZone`	Um fuso horário simplificado (Java 1.1)
`Stack`	Uma pilha (uma fila last-in-first-out)
`StringTokenizer`	Utilitários para dividir strings em elementos individuais separados por delimitadores comuns
`TimeZone`	Um fuso horário genérico (Java 1.1)
`TreeMap`	Implementação de um mapa baseado em uma árvore red-black (Java 2)
Vector	Um array de Objects que pode crescer

java.util.jar

Trata-se de um pacote introduzido na linguagem Java 2 para manipulação de arquivos Java Archive Resource (JAR).

Classes

`Attributes`	Mapeamento de nomes de atributo Manifest nos valores de string associados
`JarEntry`	Entrada de arquivo JAR
`JarFile`	Comportamento para ler arquivos JAR
`JarInputStream`	Leitura de fluxo de entrada de arquivos JAR
`JarOutputStream`	Escrita de fluxo de saída de arquivos JAR
`Manifest`	Nomes de entrada de manifestos e atributos associados

java.util.mime

Trata-se de um pacote introduzido na linguagem Java 2 para suporte a MIME (Multipurpose Internet Mail Extension).

Classes

MimeType	Um tipo MIME
MimeTypeParameterLi st	Lista de parâmetros de um tipo MIME

java.util.zip

Esse pacote fornece classes para tratar de respositórios de arquivos compactados, usando os algoritmos de compactação Zip e gZip.

Interfaces

Checksum	Métodos para calcular soma de verificação

Classes

Adler32	Calcula uma soma de verificação Adler 32
CRC32	Calcula uma soma de verificação CRC 32
CheckedlnputStream	Fluxo de entrada com uma soma de verificação associada
CheckedOutputStream	Fluxo de saída com uma soma de verificação associada
Deflator	Compactador de arquivos descompactados
Def1atorOutputStream	Fluxo de saída que compacta
GZIPInputSteam	Fluxo de entrada de um arquivo gZip
GZIPOutputStream	Fluxo de saída de um arquivo gZip
Inflater	Descompactador de arquivos compactados
InflaterInputStream	Fluxo de entrada que descompacta
ZipEntry	Uma entrada dentro de um arquivo Zi p
ZipFile	Um arquivo Zip inteiro
ZipInputStream	Fluxo de entrada de um arquivo Zi p
ZipOutputStream	Fluxo de saída em um arquivo Zi p

Apêndice C

Recursos sobre Java na Internet

Este apêndice lista livros, sites da World Wide Web, fóruns de discussão na Internet e outros recursos que você pode usar para aumentar seu conhecimento sobre a linguagem Java.

O site da Web deste livro

Rogers Cadenhead, co-autor deste trabalho, mantém o site oficial na Web deste livro no seguinte endereço:

http://www.prefect.com/java21

Visite o site para obter o código-fonte de todos os projetos do livro, a lista de erratas atuais e esclarecimentos, vínculos atualizados e outras informações.

Outros livros a considerar

A Macmillan Computer Publishing é líder no setor de livros de programação Java, e existem numerosos deles que você deve considerar a leitura ao desenvolver seus conhecimentos. A lista a seguir inclui os números ISBN, os quais as livrarias precisarão caso não possuam o livro que você está orocurando:

- *Developing Intranet Applications with Java* by Jerry Ablan, de Jerry Ablan, Rogers Cadenhead e outros. ISBN: 1-57521-166-1.
- *Java Unleashed 2nd Edition,* de Rogers Cadenhead, Michael Morrison e outros. ISBN: 1-57521-049-5.
- *Developing Professional Java Applets,* de K.C. Hopson e Stephen E. Ingram. ISBN: 1-57521-083-5.
- *Tricks ofthejava Programming Gurus,* de Glenn Vanderburg e outros. ISBN: 1-57521-102-5.

- *Java Dèvelopers Guide,* de Jamie Jaworski. ISBN: 1-57521-069-X.
- *Creating Web Applets withjava,* de David Gulbransen e Ben Rawlings. ISBN: 1-57521-070-3.
- *Sams Teach YourselfInternet Game Progmmmingwith Java m 21* Days, de Michael Morrison. ISBN: 1-57521-148-3.

Vários desses livros estão disponíveis em sua totalidade no site da World Wide Web da Macmillan Computer Publishing, no endereço:

```
http://www.mcp.com
```

O site inclui a Macmillan Online Bookstore e vínculos para sites da Web de autores.

O site da Sun Microsystems

Como você aprendeu no Dia 1, a Sun mantém um site abrangente na Web, no seguinte endereço:

http://java.sun.com

A Sun produz novas versões do Java Development Kit, da biblioteca de classes Java e outros produtos. O site é o primeiro local para se procurar informações relacionadas à linguagem. Novas versões do Development Kit e muitos outros recursos de programação estão disponíveis nesse site.

Quando este livro estava sendo escrito, o site da Sun era dividido nas seguintes áreas:

- **What's** New? Anúncios relacionados a futuras versões de produtos e eventos, como o JavaOne, a conferência semestral para programadores Java. Essa área também contém divulgações à imprensa feitas pela Sun, assim como cronogramas de sessões de treinamento que são oferecidas pela empresa.
- Read **About** Java Artigos sobre a linguagem, destinados às pessoas que a estão estudando pela primeira vez. Esse é um bom lugar para começar, pois ele apresenta os tópicos tendo os iniciantes como alvo.
- **Products** & **APIs** Todos os produtos e a documentação da Sun que podem ser carregados por download, incluindo o Development Kit, documentação da linguagem e outros arquivos.
- **Applets** Uma vitrine de programas Java que são executados na Web, incluindo mais de duas dúzias oferecidos pela Sun, que podem ser adaptados para uso em suas próprias páginas da Web. Há também vários vínculos a diversos diretórios de applets da Internet, incluindo o Developer.Com's Gamelan, no endereço http://www.gamelan.com, e o Java Applet Rating Service (JARS), no endereço http://www.jars.com.

- **For Developers** Indo para o Java Developer Connection, temos um recurso consolidado de suporte oficial da Sun e documentação da linguagem Java no formato HTML. Você pode encontrar informações a respeito de conferências sobre a linguagem, livros oficiais de Java e outros recursos.
- **Java in the Real World** Recursos e cronologia sobre "histórias de sucesso" da linguagem no mercado e configurações diversificadas, variando desde o U.S. Postal Service, UCLA Medicai Center, Ergon Informatik, até a missão Pathfinder para Marte.
- **Business & Licensing** Fornece diretrizes para o licenciamento e marca registrada para uso de produtos Java.
- **Support** & **Services** Listagens de suporte técnico, atendimento ao cliente e serviços de vendas.
- **Marketing** Detalhes sobre o programa "100 Percent Purê Java", Java Solutions Guide e outros trabalhos de marketing e suporte relacionados à linguagem.
- **Employment** Listagens das vagas atuais na Sun, nas áreas de engenharia, redação técnica, marketing/vendas e outros departamentos.
- **Java Store** Um catálogo das mercadorias oficiais da linguagem Java, que podem ser pedidas através da Web, incluindo camisas de algodão, canecas de café, T-shirts e bonés de beisebol.

O site é atualizado continuamente com recursos gratuitos, úteis para programadores Java. Algo de que talvez você queira tirar proveito imediatamente é a página Documentation, no seguinte endereço:

http://java.sun.com/docs/index.html

Outros sites da Web sobre Java

Como grande parte do fenômeno Java foi inspirado pelo seu uso em páginas da Web, um grande número de sites trata da linguagem e da programação Java.

A página de livros sobre Java

Como um meio de classificar as centenas de livros que já foram publicados sobre Java, vários sites de análise foram compilados com listagens de livros novos, atuais e esgotados.

A *JavaWorld,* uma revista online que aborda a linguagem e a tecnologia relacionada, mantém uma lista de livros atuais e futuros sobre Java. Você pode encontrá-la no seguinte endereço:

http://www.javaworld.com/javaworld/books/jw-books-index.html

Outro sumário de livros relacionados à linguagem Java é apresentado por Elliotte Rusty Harold, o autor de vários dos livros descritos na página da Web. A lista de Harold, com análises de muitos livros, está disponível na seguinte página:

http://sunsite.unc.edu/javafaq/books.html

Diretório sobre Java do Gamelan

Como Java é uma linguagem de programação orientada a objetos, é fácil usar recursos criados por outros programadores em seus próprios programas. Antes de começar um projeto Java de qualquer porte, você deve procurar na World Wide Web por recursos que possa utilizar em seu programa.

O lugar para se começar é no Developer.Com's Gamelan, o site da Web que cataloga programas Java, recursos de programação e outras informações. Visite o seguinte endereço:

http://www.gamelan.com

Gamelan é o diretório mais abrangente deste tipo na Web. Supera o próprio site da Sun na profundidade de sua abordagem. Tornou-se o primeiro local em que os programadores Java registram informações sobre um programa quando ele é concluído. Os membros da equipe Gamelan atualizam o site diariamente. O Gamelan também destaca os melhores envios para seu diretório, na seguinte página:

http://www.gamelan.com/special/picks.html

Java Applet Rating Service

Para acessar outro diretório que classifica applets Java, direcione seu navegador da Web para o seguinte endereço:

http://www.jars.com

A maçã, logotipo do Java Applet Rating Service, pode ser vista em diversos applets Java oferecidos nas páginas da Web. O site do JARS foi ampliado recentemente para incluir notícias sobre a linguagem e desenvolvimentos relacionados, análises de ferramentas de desenvolvimento Java e outras informações úteis.

Revista *JavaWorld*

Uma das melhores revistas que apareceram para atender à comunidade de programação Java também é a mais barata. A *JavaWorld* está disponível gratuitamente na World Wide Web, no seguinte endereço:

http://www.javaworl d. com

A *JavaWorld* publica freqüentes artigos tutoriais, junto com notícias sobre desenvolvimento em Java e outras atrações, que são atualizadas mensalmente. O formato exclusivo da Web proporciona uma vantagem em relação a alguns de seus concorrentes impressos (como *a. Java Report)* na área dos artigos "how-to" (como fazer). Enquanto um artigo está ensinando um conceito ou tipo de programação em particular, *a JavaWorld* pode oferecer um applet Java que demonstra a lição.

Perguntas mais freqüentes sobre Java

Como um complemento das listas de FAQ (Frequently Asked Question) sobre a linguagem Java, que estão disponíveis no site da Web da Sun Microsystems, programadores Java que utilizam grupos de discussão na Internet têm colaborado com sua própria lista de perguntas e respostas.

Elliotte Rusty Harold, um dos mantenedores de páginas da Web sobre livros a respeito de Java, também oferece a lista de FAQ Java atual, no seguinte endereço:

http://sunsi te.une.edu/j avafaq/javafaq.html

Outro recurso semelhante, o "Unofficial Obscure Java FAQ", começou com um esforço em responder a algumas perguntas menos freqüentes. Ele se encontra no seguinte site:

http://k2.scl.cwru.edu/-gaunt/j ava/java-faq.html

Newsgroups de Java

Um dos melhores recursos para programadores iniciantes e experientes é a Usenet, a rede internacional de grupos de discussão, que está disponível para a maioria dos usuários da Internet. A seguir estão descrições de alguns dos vários grupos de discussão disponíveis na Usenet:

- comp.1ang.java.mi se Embora esse grupo seja designado como a área de discussão sobre Java para todos os assuntos que não pertençam a nenhum dos outros grupos, ele é mais usado do que qualquer um dos outros. Ele substituiu o comp. 1 ang. j ava, em meados de 1996. Qualquer assunto relacionado à linguagem Java é conveniente para discussão aqui.
- comp. 1 ang. j ava. advocacy Esse grupo é dedicado a todas as discussões sobre Java que provavelmente inspirem debates acalorados ou comparativos. Se você quiser discutir os méritos de Java em relação a outra linguagem, este é o lugar certo. Esse grupo pode ser um bom lugar de consulta, caso você queira ver se a linguagem Java é a escolha certa para um projeto em que esteja trabalhando.

- comp.lang.java.announce Este grupo publica anúncios, propaganda e divulgações à imprensa de interesse para a comunidade de desenvolvimento Java. Ele é moderado; portanto, todas as colocações devem ser submetidas à aprovação, antes de serem enviadas para o grupo.
- comp.lang.java.api Este grupo discute a Application Programming Interface da linguagem Java, os programas completos de biblioteca de classes que acompanham o Java Developer's Kit e outras implementações da linguagem.
- comp.lang.java.programmer Este grupo contém perguntas e respostas sobre a programação Java, o que o torna outro bom lugar para os novos programadores freqüentarem.
- comp.lang.java.security Este grupo de discussão é dedicado às questões de segurança relacionadas à linguagem Java, especialmente com relação à execução de programas Java e outros conteúdos executáveis na World Wide Web.
- comp.lang.java.setup Este grupo fornece um lugar para se discutir problemas de instalação relacionados às ferramentas de programação Java e outras questões similares.
- comp.lang. java. tech O mais avançado dos grupos de discussão Java, este grupo é dedicado a discutir a implementação da linguagem, problemas com a sua mudança para novas máquinas, os detalhes específicos da Java Virtual Machine e assuntos semelhantes.

Oportunidades de emprego

Se você está aprendendo a linguagem Java como um modo de encontrar um emprego, então deve conferir algumas das listagens sobre trabalhos relacionados a Java que são apresentadas na Web. Vários recursos listados neste apêndice possuem uma seção dedicada às oportunidades de emprego.

A *JavaWorld* oferece uma página Career Opportunities que, às vezes, lista vagas para desenvolvedores Java:

```
http://www.javaworld.com/javaworld/common/jw-jobop.html
```

Algo que você pode fazer para que os possíveis empregadores conheçam suas habilidades em Java é registrar-se como um recurso do diretório Gamelan. O Gamelan o incluirá em seu site e essa listagem pode resultar em uma correspondência eletrônica a respeito de indicações para trabalho relacionado à linguagem Java. Para saber como você faz para se registrar, consulte o seguinte endereço na seção Add a Resource do Gamelan:

```
http://www.gamelan.com/submit/submit_persori.shtml
```

Embora essa página da Web não seja especificamente um recurso de emprego relacionado à linguagem Java, o site Career Path da World Wide Web permite que você pesquise os classificados de emprego de mais de vinte jornais dos EUA. Você precisa se registrar para usar o site, mas isso é gratuito e existem mais de 100.000 anúncios classificados que podem ser pesquisados usando-se palavras-chave, como Java ou Internet. Visite o seguinte endereço:

http://www.careerpath.com

Apêndice D

Configurando o Java Development Kit

O Java Development Kit é um conjunto de utilitários de linha de comando, os quais são usados para criar, compilar e executar programas Java. O kit exige o uso de um prompt de comando, pois você executa cada um dos seus programas digitando seus nomes junto com todos os argumentos que forem necessários.

Configuração do Windows 95/98 e do Windows NT

Os usuários de Windows 95/98 e Windows NT podem obter um prompt de comando usando o comando MS-DOS Prompt (Prompt do MS-DOS para Windows), disponível como Start | Programs | MS-DOS Prompt (Iniciar | Programas | Prompt do MS-DOS) a partir da barra de tarefas.

Quando você usa o comando MS-DOS Prompt (Prompt do MS-DOS), uma janela se abre, na qual podem ser digitados comandos e utilizar comandos do MS-DOS para navegar pelas pastas de seu sistema. A pasta inicial será a \WINDOWS, em sua unidade de disco rígido principal (normalmente C:\WINDOWS).

Um exemplo de comando que você pode inserir, quando tiver instalado o JDK, é o seguinte:

```
java -version
```

Esse comando executa o j ava. exe, o interpretador Java que faz parte do JDK. O argumento, -version, faz o interpretador apresentar seu número de versão.

Experimente esse comando. Se você estiver usando o JDK 1.2 — a versão do JDK atualmente usada na linguagem Java 2 — a seguinte mensagem deve ser apresentada em resposta:

```
java version "1.2"
```

As versões posteriores do JDK 1.2 responderão com um número de versão ligeiramente diferente.

Se você obtiver um erro, como o número de versão incorreto ou a mensagem Fi 1 e not found, seu sistema está com problemas para localizar o interpretador Java. Você precisa fazer algumas alterações na configuração de seu sistema para corrigir isso.

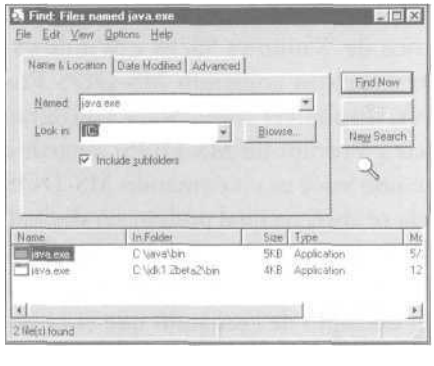
A primeira coisa a fazer ao tratar de um problema de configuração do JDK no Windows é descobrir onde o j ava. exe está localizado em seu sistema. Use o comando Find Files (Localizar Arquivos), através das instruções Start | Find | Files or Folders (Iniciar | Localizar | Arquivos ou Pastas) na barra de tarefas, para procurar java.exe na unidade de disco rígido em que você instalou o JDK.

Se não encontrar o j ava. exe, você precisará reinstalar o JDK e certificar-se de que o arquivo de instalação tenha o tamanho correto, antes de executá-lo novamente.

Se você encontrar o j ava. exe, poderá se deparar com mais de uma versão do arquivo, caso possua versões anteriores do JDK em seu sistema. Olhe na coluna In Folder (Na pasta) da janela Find (Localizar) para determinar os nomes das pastas que contêm versões de j ava. exe. Uma delas deve incluir uma referência à pasta em que você instalou o JDK 1.2.

Na Figura D.l, existem duas versões de java.exe listadas na janela Find (Localizar). A que se refere à instalação atual do JDK nesse exemplo é C:\jdkl.2beta2\bin.

Figura D.1
Resultado de uma busca *de java.exe em Find Files.*



Tome nota dessa pasta exatamente como ela está listada em In Folder (Na pasta). Essa é a pasta Path do JDK e você a usará em breve.

Verificando a instrução PATH

Para certificar-se de que seu sistema está procurando os arquivos do JDK 1.2 no lugar correto, você precisa verificar a configuração PATH de seu sistema. PATH indica onde encontrar um programa, quando você digita seu nome em um prompt do MS-DOS (como no caso de java -version).

Para apresentar a configuração atual de PATH, digite o seguinte em um prompt de comando:

ENTRADA

```
path
```

Você verá uma listagem de todas as pastas em que o Windows procura para encontrar programas MS-DOS; eles são separados porpontos-e-vírgulas. A seguir há um exemplo:

```
PATH=C:\WIND0WS;C:\WIND0WS\C0MMAND;C:\JDK1.2\BIN
```

Nessa listagem de PATH, o texto C:\JDK1.2\BIN se refere à pasta PATH do JDK, que é onde o sistema procurará o arquivo java.exe. Existem ainda duas outras pastas listadas em PATH: C:\WINDOWS e C:\WINDOWS\COMMAND.

Sua configuração de PATH deve incluir uma referência à pasta PATH do JDK. (Nesse caso, a utilização correta de letras maiúsculas e minúsculas não é importante — C:\JDK1.2\BIN seria o mesmo que C:\jdkl.2\bin.)

Se PATH não incluir a pasta PATH do JDK, você precisará editar sua configuração de PATH e incluí-la.

Mudando sua configuração de PATH

A configuração de PATH deve ser alterada pela edição de autoexec.bat, um arquivo de texto na pasta-raiz da unidade de disco rígido principal de seu sistema (normalmente a unidade C).

Para mudar a configuração de PATH em um sistema Windows NT, você não precisa carregar um arquivo de texto para editar. Em vez disso, escolha Settings | Control Panei na barra de tarefas e dê um clique duplo no ícone System.

Para mudar PATH em um sistema Windows 95/98, você deve editar o arquivo de texto autoexec. bat. Dê um clique com o botão direito do mouse no arquivo e selecione o comando Editar para carregar autoexec.bat no Bloco de Notas.

O arquivo autoexec. bat conterá muitos itens de cunho técnico, que serão incompreensíveis para o iniciante em MS-DOS.

Procure uma linha que comece com o texto PATH=, SET PATH= ou PATH, seguido de um espaço em branco.

Problema: Não existe instrução PATH

Se você não encontrar uma instrução PATH, então deve incluir uma no arquivo autoexec.bat. Role para baixo até o final do arquivo e inclua uma linha em branco. Nessa linha, digite o texto PATH=, seguido da pasta PATH do JDK. A linha a seguir poderia ser usada, se seu JDK estiver localizado na pasta C:\jdkl.2 em seu sistema:

```
PATH=C:\jdkl.2\bin
```

Após fazer essa alteração, salve o arquivo. Você deve reinicializar seu sistema, antes que a alteração entre em vigor. Se o arquivo java.exe estiver na pasta indicada na instrução PATH, você deverá ter êxito ao executar a instrução java -version.

Problema: Não existe uma pasta JDK 1.2 na instrução PATH

Se você encontrar uma instrução PATH no arquivo autoexec.bat, que não contenha nenhuma referência à pasta Path do JDK, procure o texto %JAVA_HOME% nessa linha, seguido de um texto como \bin.

Se você encontrar %JAVA_HOME%, exclua-o e também ao texto que o acompanha, até (mas não incluindo) o próximo ponto-e-vírgula. Substitua esse texto pela pasta PATH do JDK. Certifique-se de não excluir nenhum ponto-e-vírgula utilizado para separar nomes de pasta.

Se você não encontrar %JAVA_HOME% na instrução PATH, coloque seu cursor no final da linha que contém a instrução PATH. Nesse ponto, inclua um ponto-e-vírgula (;), seguido da pasta PATH do JDK. Se sua pasta PATH do JDK for C:\jdk1.2\bin, o final da instrução PATH deverá ser semelhante à última parte da linha a seguir:

```
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\jdk12\bin
```

Não deve haver referência a nenhuma outra versão do JDK na instrução PATH. Se você vir uma pasta que faça referência a uma versão anterior do JDK na linha PATH, exclua a referência a essa pasta. Se isso resultar em uma linha PATH que contenha dois caracteres de ponto-e-vírgula seguidos (;;), exclua um deles.

Salve o arquivo, após fazer essas alterações. Você deve reinicializar seu sistema, antes que a nova instrução PATH entre em vigor. Se o arquivo java.exe estiver na pasta indicada na instrução PATH, você deverá ter êxito ao executar o comando java -version.

Corrigindo erros Class not found

Os programas Java são compilados com o JDK executando-se o compilador Java javac com o nome do arquivo-fonte como argumento. Por exemplo, se você estivesse na mesma pasta que o arquivo-fonte HelloDan.java, poderia compilá-lo com o seguinte comando:

```
javac HelloDan.java
```

Dica

Se você ainda não tentou compilar um programa Java para testar o JDK, pode usar o arquivo HelloDan.java do site da Web deste livro, no endereço http://www.prefect.com/java21.

Se você obtiver um erro Class not found ao usar o compilador Java em um arquivo-fonte, a primeira coisa a ser verificada é se o nome foi digitado corretamente, inclusive com as letras maiúsculas e minúsculas certas.

Certifique-se de que você esteja na pasta que contém o arquivo-fonte que está tentando compilar e confira o nome do arquivo.

Os usuários de Windows podem usar o comando dir no prompt do MS-DOS, para listar todos os arquivos de uma pasta. O nome completo de cada arquivo é apresentado na coluna da direita da listagem da pasta, como se vê na Figura D.2. Os nomes de arquivo abreviados, no lado esquerdo da listagem, devem ser ignorados — eles são usados internamente pelo Windows, para gerenciar os arquivos.

Figura D.2

Uma listagem de arquivos em uma janela do MS-DOS.

Se o nome do arquivo-fonte estiver correto e não houver erros no próprio código-fonte Java, seu sistema está com problemas para localizar o arquivo classes.zip, que contém todos os arquivos de classe Java necessários para se compilar e executar com os programas Java 2.

O JDK procura o arquivo classes.zip de duas maneiras. Primeiramente, ele utiliza a configuração de CLASSPATH do sistema (caso uma tenha sido configurada). Em segundo lugar, ele procura o arquivo java.exe e usa a localização desse arquivo para determinar onde o arquivo classes.zip pode ser encontrado.

A maioria dos erros Class not found pode ser corrigida usando-se a configuração de CLASSPATH para indicar a localização de classes.zip.

Um modo de encontrar o arquivo classes.zip é abrindo a pasta onde você instalou o JDK (por exemplo, \jdkl.2). Deve haver uma subpasta chamada 1 ib, que contém o arquivo classes.zip.

No Windows 95 ou no Windows NT, use o comando Find Files (Localizar Arquivos), através de Start | Find Files or Folders | Find Files (Iniciar | Localizar Arquivos ou Pastas | Localizar Arquivos), na barra de tarefas, para procurar classes.zip na mesma unidade de disco em que você instalou o JDK (como se vê na Figura D.3).

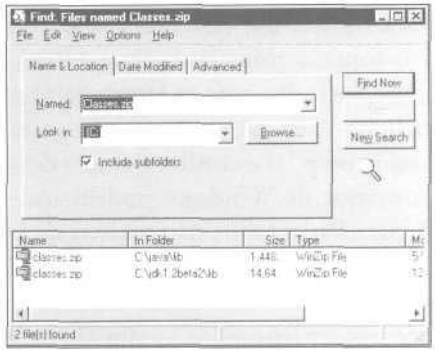


Figura D.3
Resultado de uma busca de classes.zip com Find Files.

Às vezes, haverá mais de um arquivo chamado classes.zip em seu sistema. Alguns deles podem corresponder a versões anteriores do JDK e outras ferramentas de desenvolvimento Java, e outros ainda poderiam ser usados pelos navegadores da Web que oferecem suporte à linguagem Java.

Olhe na coluna In Folder (Na pasta) para ver o nome completo (incluindo a letra da unidade de disco rígido) de cada pasta que contém um arquivo chamado classes.zip. Encontre o arquivo que inclui uma referência à pasta onde você instalou o JDK 1.2. (Na Figura D.3, é a pasta C:\jdkl.2beta2\l ib.)

Tome nota dessa pasta. Essa pasta, seguida de \classes.zip, deve estar em sua configuração de CLASSPATH.

Verificando a instrução CLASSPATH

Para certificar-se de que seu sistema está procurando a versão do JDK 1.2 de classes.zip no lugar certo, você precisa examinara configuração de CLASSPATH de seu sistema.

Os usuários de Windows podem apresentar a configuração atual de CLASSPATH digitando o seguinte em um prompt do MS-DOS:

ENTRADA

```
echo %CLASSPATH%
```

Certifique-se de incluir os sinais de porcentagem (%) antes e depois do termo CLASSPATH. Se seu sistema tiver CLASSPATH configurado, você verá uma listagem de todas as pastas e arquivos em que as ferramentas do JDK procurarão por classes Java. Cada pasta e nome de arquivo é separado por pontos-e-vírgulas. A seguir há um exemplo:

```
.;C:\jdkl.2\lib\classes.zip
```

Nessa listagem de CLASSPATH, o texto C:\jdkl.2\lib\classes.zip é um local a partir do qual as classes Java serão carregadas. Há também um ponto-final (.) listado como o primeiro item — essa referência garante que os utilitários do JDK também procurarão na pasta atual todas as classes que não puderem ser encontradas.

Se CLASSPATH não incluir a referência à cópia do arquivo classes.zip do JDK 1.2, você precisará editar sua configuração de CLASSPATH e incluí-la.

Mudando sua configuração de CLASSPATH

A configuração de CLASSPATH deve ser alterada pela edição de autoexec. bat, um arquivo de texto na pasta-raiz da unidade de disco rígido principal de seu sistema (normalmente a unidade C).

Para mudar a configuração de CLASSPATH em um sistema Windows NT, você não precisa carregar um arquivo de texto para edição. Em vez disso, escolha Control Panei | Settings na barra de tarefas.

Para mudar CLASSPATH em um sistema Windows 95/98, você precisa editar o arquivo de texto autoexec.bat. Dê um clique com o botão direito do mouse no arquivo e selecione o comando Editar para carregar o arquivo autoexec.bat no Bloco de Notas.

O arquivo autoexec.bat conterá muitos itens de cunho técnico que serão incompreensíveis para o iniciante em MS-DOS.

Procure uma linha que comece com o texto CLASSPATH=, SET CLASSPATH= ou CLASSPATH, seguido de um espaço em branco.

Problema: Não existe instrução CLASSPATH

Se você não encontrar uma instrução CLASSPATH, deve incluir uma no arquivo autoexec.bat. Role para baixo, até o final do arquivo de texto e inclua uma linha em branco. Nessa linha, introduza o texto CLASSPATH=, seguido da pasta Path do JDK, como se vê a seguir:

```
CLASSPATH=.;C:\jdkl.2\lib\classes.zip
```

Salve o arquivo após fazer essa alteração. Você deve reinicializar seu sistema, antes que a alteração entre em vigor. Se o arquivo classes.zip estiver na pasta indicada na instrução CLASSPATH, você deverá ter êxito na compilação dos programas.

Problema: Não existe pasta JDK 1.2 na instrução CLASSPATH

Se você encontrar uma instrução CLASSPATH no arquivo autoexec.bat, que não contém nenhuma referência à localização correta de classes.zip, procure o texto %JAVA_HOME% nessa linha.

Você pode encontrar %JAVA_H0ME% seguido do texto \lib\classes, zip, como em CLASSPATH=%JAVA_HOME%\1 i b\classes .zip ou CLASSPATH = %JAVA_HOME%\..\lib\classes.zip.

Se você encontrar %J AVA_H0ME%, exclua-o e também a todo texto que venha após ele, até encontrar o próximo ponto-e-vírgula. Substitua-o pela localização correta de classes.zip. Certifique-se de não excluir nenhum ponto-e-vírgula que seja usado para separar nomes de pasta.

Se você não encontrar %JAVA_HOME% na instrução CLASSPATH, coloque seu cursor no final da linha que contém a instrução CLASSPATH. Nesse ponto, inclua o texto;, seguido da localização correta de classes.zip. Se ela for C:\jdkl.2\lib\classes.zip, o final da instrução CLASSPATH deverá ser semelhante à última parte da linha a seguir:

```
CLASSPATH=.;C:\DEV\CHATSERVER\;C:\jdkl2\lib\classes.zip
```

Não deve haver referência a nenhuma outra versão do JDK na instrução CLASSPATH. Se você vir uma pasta que faça referência a uma versão anterior do JDK na linha CLASSPATH, exclua a referência a essa pasta. Se isso resultar em uma linha CLASSPATH que contenha dois caracteres de ponto-e-vírgula seguidos (;;), exclua um deles. Salve o arquivo, após fazer essas alterações. Você deve reinicializar seu sistema, antes que a nova instrução CLASSPATH entre em vigor. Se o arquivo classes.zip correto estiver na pasta indicada na instrução PATH, você deverá ter êxito ao compilar e executar os programas de exemplo, como o HelloDan.

Configuração de UNIX

Para configurar o JDK em um sistema Solaris, inclua o diretório java/bin ou jdk/bin em seu caminho de execução. Normalmente isso pode ser feito inserindo-se uma linha como a seguinte em seu arquivo .profile, .cshrc ou .login:

```
set path= (-/java/bin/ $path)
```

Essa linha pressupõe que você tenha instalado o JDK no diretório java, fora de seu diretório de base. Uma instalação em outro lugar qualquer exigirá uma mudança no diretório incluído em seu caminho de execução.

Essas alterações não entrarão em vigor até que você se desconecte e conecte novamente ou use o comando source com o nome do arquivo que foi alterado. Se você alterasse o arquivo . 1 ogi n, o comando source seria o seguinte:

```
source -/.login
```

Corrigindo erros Cl ass not found em outras plataformas

Para corrigir todos os erros Cl ass not found em sistemas Solaris, o melhor é certificar-se de que a variável de ambiente CLASSPATH não esteja sendo definida automaticamente no login.

Para ver se CLASSPATH está sendo definida, digite o seguinte em um prompt de comando:

ENTRADA

```
echo $CLASSPATH
```

Se um valor de CLASSPATH foi definido, você pode desfazer sua definição digitando o seguinte comando:

```
unsetenv CLASSPATH
```

Para tornar essa alteração permanente, você deve remover o comando que define CLASSPATH de seu arquivo .profile, .cshrc ou .login.

Essas alterações não surtirão efeito até que você se desconecte e conecte novamente ou use o comando source com o nome do arquivo que foi mudado. Se você alterasse o arquivo .login, o comando source seria o seguinte:

```
source -/.login
```

Apêndice E

Usando um editor de textos com o JDK

Ao contrário das ferramentas de desenvolvimento Java, como Visual Café e SunSoft Java WorkShop, o Java Development Kit não possui um editor de textos para se usar ao criar arquivos-fonte.

Neste apêndice, você aprenderá a selecionar um editor para usar com o JDK e para configurar seu sistema para trabalhar com o editor.

Escolhendo um editor de textos

Para que um editor ou processador de textos funcione com o JDK, ele deve poder salvar arquivos de texto sem formatação.

Essa característica recebe diferentes nomes nos diversos editores. Procure uma opção de formato como as seguintes, quando salvar um documento ou definir as propriedades de um documento:

- Texto puro
- Texto ASCII
- Texto DOS
- Somente texto

Se você estiver usando o Windows 95/98, existem vários editores incluídos com o sistema operacional, que podem ser usados.

O Notepad do Windows (disponível em Programs | Accessories | Notepad, a partir do botão Start), é um editor de textos simples que trabalha apenas com arquivos de texto puro. Ele só consegue manipular um documento por vez.

O WordPad do Windows (Programs | Accessories | WordPad, a partir do botão Start) está um passo acima do Notepad. Ele pode manipular mais de

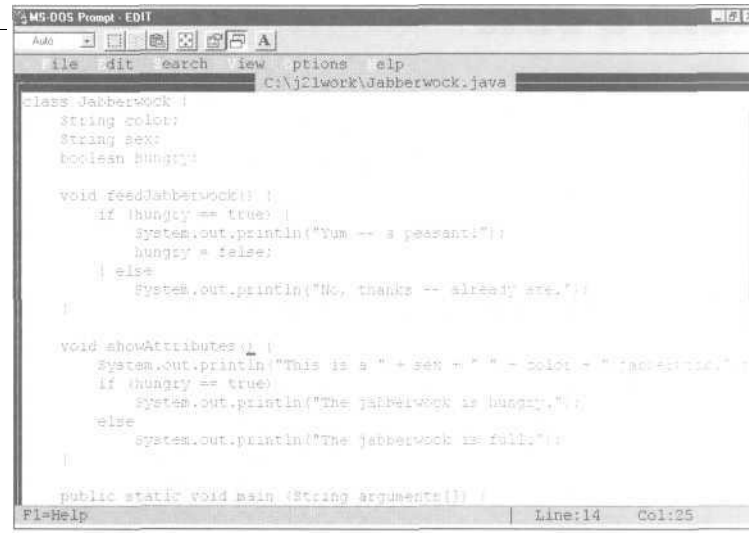
um documento por vez e pode trabalhar com texto puro e com os formatos do Microsoft Word. Ele também se recorda dos últimos documentos com que trabalhou, tornando-os disponíveis no menu suspenso File.

O Edit do DOS, que pode ser executado a partir de um prompt do MS-DOS com o comando edi t, é outro editor simples que manipula documentos de texto sem formatação. Ele parecerá simples demais para um usuário de Windows 95 que não esteja familiarizado com o MS-DOS, mas possui um recurso que o Notepad e o WordPad não têm: o Edit apresenta o número da linha de seu cursor.

A capacidade de apresentar o número da linha que você está editando existe no Edit do DOS e em alguns outros editores de texto. A numeração começa com 1 para a linha superior do arquivo e aumenta à medida que você se move para baixo. A Figura E.l mostra o Edit; o número da linha é indicado no canto inferior direito da janela do programa.

Figura E.1

Um arquivo-fonte Java carregado no Edit do **DOS.**



O número da linha ajuda na programação Java, pois muitos compiladores Java indicam o número da linha em que um erro ocorreu. Dê uma olhada no seguinte erro, gerado pelo compilador JDK:

```
Palindrome.java:2: Class Font not found in type declaration.
```

Esse erro inclui dois-pontos seguido de um número após o nome do arquivo-fonte Java. O número corresponde à linha que causou o erro de compilador.

Com um editor de textos que oferece suporte à numeração, você pode ir diretamente a essa linha e começar a procurar o erro. Normalmente existem maneiras melhores de se depurar um programa com um pacote de programação Java comercial, mas os usuários do JDK devem procurar erros gerados pelo

compilador usando o número de linha indicado pela ferramenta j avac. Por isso, é melhor usar um editor de textos que ofereça suporte à numeração.

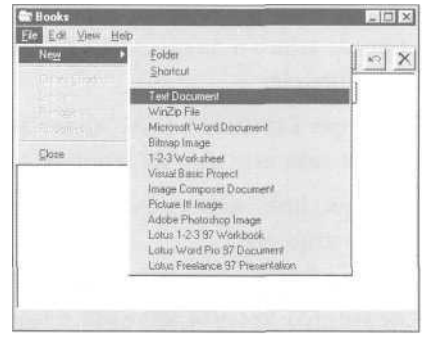
Criando uma associação de arquivo no Windows

Depois que um editor de textos tiver sido selecionado, os usuários de Windows devem associar esse editor à extensão de arquivo .java.Isso torna possível abrir um arquivo-fonte . java dando-se um clique em seu nome em uma pasta; isso também impede que editores, como o Notepad do Windows, incluam incorretamente a extensão de arquivo .txt em arquivos-fonte .java.

Para criar uma associação de arquivo, primeiramente você deve ter um arquivo para trabalhar. Abra uma pasta no Windows e crie um novo documento de texto selecionando File | New | Text Document, a partir da barra de menus (ver Figura E.2).

Figura E.2

Criando um novo ***documento de*** *texto em uma pasta do Windows.*



Um novo documento de texto, chamado New Text Document.txt é criado e você tem a chance de renomeá-lo imediatamente. Mude o nome para Anything.java e confirme o novo nome quando o Windows perguntar se você realmente deseja mudar a extensão de arquivo.

Dê um clique duplo no arquivo, após ter criado o Anything.java. Se seu sistema não associar a extensão de arquivo . j ava a nenhum programa, você verá uma janela Open With. Você pode usar isso para associar a extensão de arquivo . java ao seu editor escolhido. Vá para a seção "Criando uma nova associação", neste capítulo.

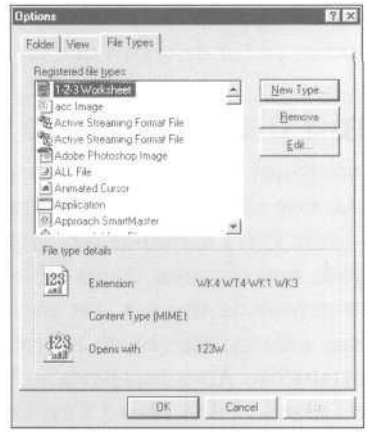
Se algo diferente acontecer, você deverá excluir a associação .java existente, antes de poder criar uma nova.

Exclua uma associação de arquivo existente

Se seu sistema já possui algo associado à extensão de arquivo .java, você pode remover essa associação a partir de qualquer pasta do Windows 95. Selecione View | Options a partir da barra de menus de uma pasta; uma janela Options se abre, contendo três caixas de diálogo com guias. Selecione a guia File Types para ver essa caixa de diálogo (ver Figura E.3).

Figura E.3

A caixa de diálogo com guias de File Types.



A caixa de listagem Registered File Types dessa janela mostra todas as extensões de arquivo que estão associadas a programas em seu sistema. Destaque um tipo de arquivo na caixa de listagem e dois outros campos oferecem informações sobre ele:

- O campo Extension exibe todas as extensões de arquivo que funcionam com esse tipo de arquivo.
- O campo Open With exibe o programa que é utilizado para abrir esse tipo de arquivo.

O tipo de arquivo 1-2-3 Worksheets, na Figura E.4, possui quatro extensões de arquivo: WK4, WT4, WK1 e WK3. Qualquer arquivo com essas extensões pode ser aberto com o programa 123W (que é o aplicativo de planilha eletrônica Lotus 1-2-3).

Role pela lista Registered File Types até encontrar um que inclua JAVA em seu campo Extension. O local mais provável para encontrá-lo é sob um cabeçalho como Java files ou Java programs, mas talvez esse não seja o caso em seu sistema.

Quando encontrar o tipo de arquivo correto, você precisará excluir a associação existente para que possa substituí-la por uma nova. Selecione Remove para excluir a associação existente e dê um clique em Yes para confirmar que você deseja removê-la.

Quando fizer isso, você poderá criar uma nova associação para a extensão de arquivo .java.

Criando uma nova associação

Uma janela Open With se abre, quando você dá um clique duplo em um arquivo que não possui associação conhecida para essa extensão. Isso está ilustrado na Figura E.4.

Figura E.4

Associando uma extensão de arquivo a um programa.

Use os passos a seguir para criar uma associação de arquivo .java:

- Na caixa de texto Description of .java Files, digite Java source file (arquivo-fonte Java) ou algo semelhante.
- Na caixa de listagem Choose the Program You Want to Use, localize o editor ou processador de textos que você deseja utilizar com os arquivos-fonte Java. Se você não encontrá-lo, selecione o botão Other e localize o programa manualmente. Se você estiver usando o Edit do DOS, ele poderá ser encontrado na pasta \Windows\Command na maioria dos sistemas, com o nome de arquivo edit ou edit.exe.
- Certifique-se de que a opção Always Use This Program to Open This File esteja selecionada (marcada).

Quando você der um clique em OK para confirmar essas configurações, o editor escolhido abrirá o arquivo Anything.java e todos os outros arquivos que possuírem a extensão .java.

Associando um ícone a um tipo de arquivo

Após ter associado os arquivos .java ao editor que você escolheu, um ícone é atribuído por padrão a todos os arquivos .java de seu sistema.

Se quiser mudar esse ícone, selecione View | Options | File Types a partir da barra de menus de uma pasta, para ver a caixa de diálogo File Types. Percorra os tipos de arquivos registrados para localizar o que está associado à extensão de arquivo JAVA.

Quando esse tipo de arquivo estiver destacado, selecione Edit para abrir uma janela Edit File Type, que aparece na Figura E.5.

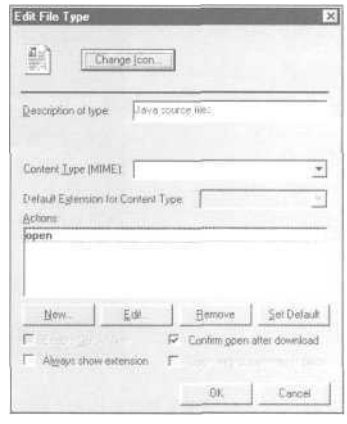
Selecione Change Icon na janela Edit File Type e você poderá escolher um ícone diferente para apresentar todos os seus arquivos .java.

Se você gostar de um dos ícones apresentados na janela Current Icon, destaque esse ícone e dê um clique em OK para fazer a alteração.

Se quiser ver outros ícones, selecione Browse para procurar nos arquivos de seu sistema, para ver os ícones que eles contêm.

Figura E.5

A *janela Edit File Type.*



Você pode abrir qualquer arquivo de ícone, programa Windows 95 ou arquivo .DLL para ver quais ícones eles contêm. Os ícones são apresentados na janela Current Icon, depois que você selecionar um arquivo.

Quando tiver encontrado um ícone que lhe agrade, destaque-o e dê um clique em OK para selecioná-lo.

Índice

Símbolos

& (E comercial), 82
* (asterisco), 391
{ } (chaves), 117
^ (sinal de inclusão), 82
= (sinal de igualdade)
  operador de atribuição, 68
  operador de igualdade, 105
! (ponto de exclamação)
  operador de desigualdade (!=), 105
  operador NOT (!), 82
- (hífen), 80
| (caractere de pipe), 82
+ (sinal de adição)
  operador de concatenação (+), 85
  operador de incremento (++), 80
/*...*/ notação de comentário, 72
/**.„•/ notação de comentário, 73
// notação de comentário, 72

A

abrindo
  conexões de soquete, 463
  conexões na Web, 458-459
  fluxos através da Internet
    exemplo de aplicativo GetFile, 460-461
    método getInputStream(), 459
    objetos BufferedReader, 459
    objetos URLConnection, 459
Abstract Windowing Toolkit, *ver* AWT
acessando
  variáveis de classe, 95
acesso
  acesso padrão, 379
  acesso privativo, 379-380
  acesso protegido, 381-382
  acesso público, 381
  controle de, visão geral, 378-379, 382-383
  herança, 383
  métodos de acesso, 383-384, 408
  pacotes
    acesso padrão, 394
    acesso público, 395-396
ajuste CLASSPATH (configuração do JDK), 393
  alterando, 623
  exibindo, 622-623
  incluindo, 623
  resolução de problemas, 623-624
alinhando
  applets, 185-186
  componentes, 290-291
    inserções, 311-312
    layouts de borda, 294-295
    layouts de carta, 297-300
    layouts de fluxo, 291-292
    layouts de grade de conteúdo, 300-312
    layouts de grade, 292-294
    pineis, 296
alocando memória, 93
alterando
  fontes de rótulo, 269
  valores de variável de classe, 95
ampliando interfaces, 403-404
Andreessen, Marc, 14
animação
  applet Neko
    carregamento da imagem, 246-247
    compilando, 253
    cor de fundo, 250
    listagem do código-fonte, 251-253
    reunião da imagem, 246
    técnica de animação, 247-250
  disposição de objeto gráfico, 253-254
  imagens
    caminhos de arquivo relativos, 241-242
    carregando, 240-241
    desenhando, 242-243
    exemplo de applet Fillmore, 244
    observadores, 243
    recuperando, 240
  pintura/repintura, 230-231
  som
    formatos de arquivo, 258
    loops, 260-261

reproduzindo, 259
threads
definição, 231
exemplo DigitalClock, 233-235
implementando, 232-234
iniciando, 233
interrompendo, 233
tremido
exemplo, 238-239
reduzindo, 238-240
visão geral, 236-237
uso de buffer duplo
criando, 254-255
definição, 254
exemplo de applet Checkers, 257-258
limitações, 254
visão geral, 229-230
ver também imagens gráficas
aninhando rotinas de tratamento de exceção, 423
anulando métodos, 54, 162-165
construtores, 166-168
destroy(), 179
getAppletInfo(), 487
handleEvent(), 337
init(), 178
palavra-chave super, 165
start(), 178
stop(), 179
update( ),237
vantagens, 165
API de interface com o usuário/mesclagem (JavaBeans), 486
API de introspecção (JavaBeans), 486
API persistente de usuário (JavaBeans), 486
aplicativo ReadPrimes, 447
aplicativo Trivia (soquete cliente/servidor), 464-465
cliente
executando, 477-478
listagem do código-fonte, 475-476
design, 465-466
servidor
arquivo QnA.txt, 469
construtor, 466-467
estado WAITFORANSWER, 469
estado WAITFORCLIENT, 469
estado WAITFORCONFIRM, 469
executando, 477-478
listagem do código-fonte, 471-474
método initQnA(), 470
método main(), 470-471
método processlnput( ), 468-469
método run(), 467-468
variáveis de instância, 466
aplicativo WritePrimes, 446-447
aplicativos
AllCapsDemo, 454-455
aplicativos AWT independentes, 360-361
argumentos
passando, 148
ArrayTest
classe ArrayTest, 116
método main(), 116
método printNames(), 116
saída, 116
BaseFramel, 351
ChangeTitle, 533-534
classes auxiliares, 147
cliente/servidor Trivia, 464-465
design, 465-466
executando, 477-478
implementação do cliente, 475-477
implementação do servidor, 466-474
comparados aos applets, 12, 373
CopyArrayWhile
código-fonte, 128
método main(), 129
saída, 129
CopyPaste, 496-497
DoTest, 130
EchoArgs, 149
estrutura Swing, 507-508
EvenSteven, 120
fechando, 373
HelloDan
arquivo-fonte, 26-28
compilando, 28-31
executando, 30-31
método main(), 147
NumberReader, 124
planos de segurança, 430
PopUpWindow, 350-351
SeeMethod (exemplo de reflexão), 108
SumAverage, 149-150
SwingColorTest, 514-516
classes, 543
listagem do código-fonte, 544-546
métodos, 543
receptores de evento, 543
ver também applets; listagens
aplicativos AWT independentes, 360-361
applet ColorTest, 362
atualizando resultados
método getHSBColor( ), 370
método RGBtoHSB(), 369
método update(), 367-368
inicialização, 363-364
layout, 362-364
listagem do código-fonte
classe ColorControls, 372
classe ColorTest, 371

marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
applet Neko
compilando, 253
cor de fundo, 250
imagens
carregando, 246-247
reunindo, 246
listagem do código-fonte, 251-253
técnica de animação, 247-250
applets, 11-13
assinaturas digitais
autoridades de certificado, 426
certificados, 426
específicas do navegador, 429-430
exemplo, 427-429
caixas de diálogo, anexando, 356
Checkers, 255-258
listagem do código-fonte, 257-258
uso de buffer duplo, 256
classe Applet, 177
ColorSwirl, 238-239
ColorTest, 362
atualizando resultados, 367-370
inicialização, 363-364
layout, 362-364
listagem do código-fonte, 371-372
marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
associando informações aos, 487
comparados aos aplicativos, 13, 174, 373
comunicação entre
método getApplet(), 492
método getAppletContext(), 491
método getApplets(), 491
cores de fundo, 216
criando, 177-178
definição, 11
depurando a saída, 197
DigitalClock
listagem do código-fonte, 234-235
loop while(), 235
método paint(), 236
método repaint(), 236
método run(), 236
método sleep(), 236
variáveis de instância, 235
Dominós, 12
exemplo Palindrome
arquivo HTML, 182
listagem do código Java, 180-181
método paint(), 181
exibindo na tela, 180
fazendo upload no servidor da Web, 184
Fíllmore, 243-244
formatos de arquivamento
arquivos CAB, 190
arquivos Zip, 189-190
inicialização, 178
iniciando, 178-179,231
interrompendo, 179, 231
Keys, 331
definição de classe, 331
listagem do código-fonte, 333
marcação HTML, 334
método init(), 331
método keyDown(), 332
método paint(), 332
limitações, 174
Lines
classe Points, 325
listagem do código-fonte, 327-328
marcação HTML, 328
método addLine(), 326
método init(), 324-325
método mouseDown(), 325
método mouseDrag(), 325-326
método mouseUp(), 326
método paint(), 326-327
linha de status, 487
Map2D, 224-227
marcação HTML
alinhamento, 185-186
espaçamento, 186
quebras de linha, 185
tag <APPLET>, 182-184
tag <OBJECT>, 187-188
tag <PARAM>, 191-192
métodos, 178
Neko
compilando, 253
cor de fundo, 250
listagem do código-fonte, 251-253
reunião da imagem, 246
técnica de animação, 247-250
NewPalindrome
arquivo HTML, 194
código-fonte Java, 193-194
passando parâmetros para
exemplo, 193-194
método getParameter(), 192
tag < PARAM >, 192
valores null, 192
planos de segurança, 430
segurança, 177
SetBack, 342
listagem do código-fonte, 345
marcação HTML, 345
método action(), 343-344
método changeColor(), 344

método init(), 343
Spots
definição de classe, 319
listagem do código-fonte, 321-322
marcação HTML, 322
método addSpot(), 320
método init(), 320
método mouseDown(), 320
método paint(), 320-321
método repaint(), 320
testando, 183-184
tratamento de eventos
cliques de mouse, 317-323
eventos de ação, 338-340
eventos de área de texto, 341
eventos de barra de rolagem, 342
eventos de foco, 340-341
eventos de lista de rolagem, 341
eventos de teclado, 328-335
exemplo de applet SetBack, 342-345
método handleEvent(), 316-317, 335-337
movimentos de mouse, 323-328
tipos de evento, 316
vendo, 60
versões de Java, 175-176
vinculação, 488-490
exemplo ButtonLink, 489-490
método linkTo(), 490
objetos URL, 488
ver também aplicativos; listagens
arcos
desenhando
método drawArc(), 207
exemplo, 208-209
preenchendo, 207
Área de Transferência, 494-498
exemplo de aplicativo, 496-497
interface ClipboardOwner, 495
operações de cópia/colagem, 495
áreas de texto
controlando, 280
criando, 279-281
exemplo, 280
Swing, 512
tratamento de eventos, 341
argumentos
passando
para aplicativos, 148
para applets, 191-195
tratando de, 148
exemplo EchoArgs, 149
arquitetura (RMI), 499-501
arquivamentos CAB (cabinet), 190-191
arquivamentos Zip, 190
arquivo QnA.txt (aplicativo TriviaServer), 469
arquivos JAR (arquivamento Java), 189-190
arquivos
class.dat, 438
excluindo, 453
renomeando, 453
retornando o tamanho de, 453
formatos
CAB, 190-191
JAR, 189-190
Zip, 190
objetos File, 453
arrastar e soltar
definição, 16
pacote java.awt.dnd
classes, 570
arrays
exemplo de programa
classe ArrayTest, 116
método main(), 116
método pnntNames(), 116
saída, 116
multidimensionais, 117
arrays multidimensionais, 117
assinatura de código, *ver* assinaturas digitais
assinaturas (digitais)
autoridades de certificado, 426
certificados, 426
específicas do navegador, 429-430
exemplo, 427-429
visão geral, 425
associando
componentes a receptores de evento, 531-532
informações a applets, 487
asterisco (•), 180, 391
ativando
itens de menu, 360
menus, 359
atribuição de nomes
componentes do Swing, 519
pacotes, 393-394
variáveis, 69
atribuindo
restrições a componentes, 304
valores de variável, 68, 86-78
atributos (classe), 38-39
atributos (HTML)
tag < APPLET >
ALIGN, 185-186
CODE, 60, 183, 186-187
CODEBASE, 186-187
HEIGHT, 60, 183
HSPACE, 186
VSPACE, 186
WIDTH, 60,183
tag <PARAM>, 191

CABBASE, 191
atributos de renderização (Java2D)
cor, 218
padrões de preenchimento, 218-219
pincéis de desenho, 220-221
áudio
formatos de arquivo, 258
reproduzindo
loops, 260-261
método play(), 259
AWT (AbstractWindowingToolkit)
animação
applet Neko, 245-253
caminhos de arquivo relativos, 241-242
carregamento de imagem, 240-241
desenhando imagens, 242-243
disposição de objeto gráfico, 253-254
observadores de imagem, 243
pintura/repintura, 230-231
recuperação de imagem, 240
som, 258-261
threads, 231-236
tremido, 236-240
uso de buffer duplo, 254-258
visão geral, 229-230
aplicativos independentes, 360-361
applet ColorTest, 362
atualizando resultados, 367-370
inicialização, 363-364
layout, 362-364
listagem do código-fonte, 371-372
marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
caixas de diálogo
anexando em applets, 356
criando, 352-353
definição, 352
exemplo TextDialog, 354-355
janelas de diálogo, 352-353
modais, 352
objetos de diálogo de arquivo, 357
classes, 266
componentes
áreas de texto, 279-281
barras de rolagem, 283-284
botões, 270-271
caixas de seleção, 271-274
campos de texto, 277-279
canvas, 284-286
incluindo em containers, 267-268
listas de escolha, 275-277
listas de rolagem, 281-282
rótulos, 268-270
diferenças de plataforma, 266
gerenciadores de layout
alternativas aos, 312-313
combinando, 295-296
criando, 290
especificando, 290-291
inserções, 311-312
layout de borda, 294-295
layout de carta, 297-300
layout de fluxo, 291-292
layout de grade de conteúdo, 300-312
layout de grade, 292-294
visão geral, 290
janelas, 348
aviso Unsigned Java Applet Window, 349
eventos, 357-358
exemplo de janela instantânea, 350-351
exibindo, 349
ocultando, 349
ver também imagens gráficas; JFC
menus
ativando, 359
barras de menu, 358-359
desativando, 359
eventos, 360
itens de menu, 359-360
pares, 313
quadros
criando, 348
dimensionando, 349
definição, 348
exemplo BaseFrame, 351
visão geral, 266

B

balões de descrição, 518
barra normal (/), 71
barras de menu, 358-359
barras de rolagem
criando, 283-284
exemplo, 283-284
gerenciando valores de, 284
Swing, 513
tratamento de eventos, 342, 536-538
beans, *ver* JavaBeans
bibliotecas de classe, 38
bitmaps
blocos, 117-118
definição, 66
try...catch, 415-416
blocos try...catch
cláusula finally, 416-418
exemplo, 415-416

Boole, George, 42
botões
  criando, 270-271
  exemplo, 270-271
  rotulando, 270
  Swing, 511
  tratamento de eventos
    eventos de ação, 535
    eventos de item, 538-540
botões de rádio
  criando, 272
  exemplo, 272
  Swing, 513
  tratamento de eventos
    eventos de ação, 535
    eventos de item, 538-540

C

caixas de combinação
  eventos de ação, 535
  eventos de item, 539
caixas de diálogo
  anexando em applets, 356
  criando, 519-520
    diálogos de confirmação, 520-521
    diálogos de entrada, 521-522
    diálogos de mensagem, 522
    diálogos de opção, 522-523
  definição, 352
  exemplo TextDialog, 354-355
  janelas de diálogo, 352
  modais, 352
  objetos de diálogo de arquivo, 357
caixas de diálogo modais, 352
  ver também caixas de diálogo
caixas de seleção
  botões de rádio, 271-274
  criando, 272
  exemplo, 272
  marcando/desmarcando, 272
  não-exclusivas, 271
  retornando o status de, 272
  tratamento de eventos
    eventos de ação, 535
    eventos de item, 538-540
camada de esqueleto (RMI), 500
camada de referência remota (RMI), 500
camada de stubs (RMI), 500
caminhos de arquivo relativos, 241-242
campos de texto
  controlando, 279
  criando, 277
  exemplo, 278
  ocultando caracteres, 277
  Swing, 511-512
  tratamento de eventos
    eventos de ação, 535
    eventos de item, 538-540
canvas, 284-286
caractere de pipe (|), 82
caracteres de final de linha, 450
carregando
  classes, 180
  imagens, 240-241, 246-247
carregando por download
  CABarc, 190
  JDK, 22
  JFC, 505
certificados (assinaturas digitais), 426
chamadas de procedure remotas (RPC), 499
chamando métodos, 160
chaves ({}), 117
chaves privativas, 428-429
chaves públicas, 428-429
círculos, desenhando, 206-207
classe MyRect
  definição de classe, 154
classes, 35-36, 61
  AbstractAccessible, 568
  abstratas, 388-389
  AccessControlContext, 598
  AccessController, 598
  AccessibleEnumeration, 568
  AccessibleObject, 592
  AccessibleRole, 568
  AccessibleState, 568
  AccessibleStateSet, 568
  ActionEvent, 571
  Activatable, 595
  ActivationDesc, 595
  ActivationGroup, 595
  ActivationGroupDesc, 595
  ActivationGroupID, 595
  ActivationID, 595
  AdjustmentEvent, 571
  AffineTransform, 572
  AlgorithmParameters, 598
  AlgorithmParametersSpi, 598
  AlphaComposite, 562
  Applet, 177
  Arc2D, 572
  atributos, 38-39
  Authenticator, 593
  AWTEvent, 562
  AWTEventMulticaster, 562
  AWTPermission, 562
  BasicPermission, 598
  BasicStroke, 562
  BeanContextAddedEvent, 586

BeanContextEvent, 586
BeanContextMembershipEvent, 586
BeanContextRemovedEvent, 586
BeanContextSupport, 586
Button, 563
ButtonGroup, 513
Canvas, 563
CardLayout, 563
Checkbox, 563
CheckboxGroup, 563
CheckboxMenuItem, 563
CheckedlnputStream, 607
CheckedOutputStream, 607
Choice, 275-277
classes auxiliares, 147
classes finais, 388
classes internas, 405-407
  atribuição de nomes, 407
  escopo, 407
  exemplo, 406
  vantagens, 406-407
Clipboard, 569
CodeSource, 598
Color, 563
ColorConvertOp, 575
ColorModel, 575
ColorSpace, 568
compilando, 43-44
criando
  definição de classe, 41
  métodos, 42-43
  variáveis de instância, 41
Component, 563
ComponentAdapter, 571
ComponentColorModel, 575
ComponentEvent, 571
ComponentSampleModel, 575
conflitos de nome, 392
constantes, 137-138
Constructor, 109
Container, 564
ContainerAdapter, 571
ContainerEvent, 571
ContentHandler, 593
ConvolveOp, 576
Countlnstances, 385
CRC32, 607
CropImageFilter, 576
CubicCurve2D, 573
Cursor, 564
Ellipse2D, 573
Error, 412
Event, 564
EventQueue, 564
exemplo Jabberwock, 45-47
Field, 109
FileDialog, 357
FilteredlmageSource, 576
FlatteningPathlterator, 573
FlowLayout, 564
FocusAdapter, 571
FocusEvent, 571
Font, 564
FontMetrics, 213, 564
Frame, 564
GeneralPath, 573
GeneralPathlterator, 573
Glyphjustificationlnfo, 572
GlyphMetrics, 572
GlyphSet, 572
GradientPaint, 564
Graphics, 565
Graphics2D, 565
GraphicsConfiguration, 565
GraphicsDevice, 565
GraphicsEnvironment, 565
GridBagConstraints, 565
GndBagLayout, 565
GridLayout, 565
GuardedObject, 598
GuardedReference, 591
GZIPInputSteam, 607
GZIPOutputStream, 607
herança
  múltipla, 54, 397
  simples, 54, 397
  visão geral, 48-50
hierarquias
  criando, 50-52
  design, 52
  métodos, 53
ICC_ColorSpace, 568
ICC_Profile, 568
ICC_ProfileGray, 568
Identity, 598
Image, 565
ImageFilter, 576
importando, 58, 391
incluindo em pacotes, 394
IndexColorModel, 576
Inflater, 607
InflaterlnputStream, 607
InputEvent, 571
Insets ou inserções, 565
ItemEvent, 571
JButton, 511
JCheckBox, 512
JComboBox, 513
JFrame, 507
JLabel, 511
JPasswordField, 512
JRadioButton, 513

JScrollBar, 513
JTextArea, 512
JTextField, 512
Kernel, 576
KeyAdapter, 571
KeyEvent, 571
Label, 565
Lease, 595
Line2D, 573
List, 565
LocateRegistry, 596
LogStream, 596
LookupOp, 576
LookupTable, 576
MarshalledObject, 594
MediaTracker, 565
MemorylmageSource, 576
Menu, 566
MenuBar, 566
MenuComponent, 566
Menultem, 566
MenuShortcut, 566
Method, 109
Modifier, 109
MouseAdapter, 571
MouseEvent, 571
MouseMotionAdapter, 576
MultibandPackedSampleModel, 576
MyRect
MyRect2, 161-162
NamedPoint, 167
Naming, 594
NearestNeighborAffine, 576
ObjID, 596
Operation, 596
PackedColorModel, 576
PaintEvent, 571
Panei, 566
Person, 159
PhantomReference, 591
PixelGrabber, 576
Point, 566
Polygon, 566
PopupMenu, 566
PrintClass, 163
Printjob, 566
PrintJob2D, 566
PrintSubClass, 163
PrintSubClass2, 164
PropertyChangeEvent, 585
PropertyChangeSupport, 585
PropertyDescriptor, 585
PropertyEditorManager, 585
PropertyEditorSupport, 585
Random, 90
Raster, 577
Rectangle, 566
Reference, 591
ReferenceQueue, 591
ReflectPermission, 592
RemoteObject, 596
RemoteServer, 597
RemoteStub, 597
ReplicateScaleFilter, 577
RGBImageFilter, 577
RMIClassLoader, 596
RMISecurityManager, 594
RMISocketFactory, 596
Scrollbar, 567
ScrollPane, 566
ServerSocket, 463
SimpleBeanlnfo, 585
Socket, 463
Socketlmpl, 464
SocketType, 597
StyledString, 572
StyledStringlterator, 572
subclasses
  criando, 55-61
  definição, 49, 62
superclasses
  definição, 49, 62
  indicando, 136
SystemColor, 567
TextArea, 567
TextAttributeSet, 567
TextComponent, 572
TextDialog, 354-355
TextEvent, 571
TextField, 277
TextHitlnfo, 572
TextLayout, 572
TexturePaint, 567
ThreadLocal, 591
Throwable, 413
Toolkit, 567
TransformOp, 577
UID, 597
UIManager, 517
UnicastRemoteObject, 597
variáveis
  variáveis de classe, 138
variável CLASSPATH, 393
VetoableChangeSupport, 585
VMID, 595
WeakReference, 591
Window, 348
WindowAdapter, 571
WindowEvent, 571
ZipEntry, 607
ZipFile, 607
ZipInputStream, 607

ZipOutputStream, 607
ver também interfaces
classes auxiliares, 147
classes finais, 388
classes internas, 405-407
atribuindo nomes, 407
escopo, 407
exemplo, 406
vantagens, 406-407
clientes
aplicativo Trivia
executando, 477-478
listagem do código-fonte, 475-476
soquetes, 463-464
código de byte, 18
código de máquina, 16
códigos de escape, 75
coerção
objetos
Graphics2D, 218
para interfaces, 401
ver também conversão
comando Prompt do MS-DOS (menu Programas), 25
comandos (MS-DOS)
ed, 28
dir, 29
comentários, 71-72
como a linguagem Java considera letras maiúsculas e minúsculas, 69
como capturar exceções, 414
instrução finally, 416-418
combinando gerenciadores de layout, 295-296
comparando objetos, 105-106
compilando aplicativos
plataformas Windows, 28-30
sistemas Solaris, 30-31
concatenando strings, 85
configurando o JDK (Java Development Kit),
instrução CLASSPATH, 622-624
plataformas UNIX, 624
como editar ajustes de configuração do JDK (Java Development Kit) CLASSPATH, 623
enumerando listas encadeadas, 404-405
enviando formulários, 478
escrevendo em fluxos, 435
fluxos colocados em buffer, 441-445
fluxos de caractere, 451-452
fluxos de saída de arquivo, 439-440
excluindo arquivos, 453
fazendo loop em arquivos de som, 260-261
fazendo upload de applets no servidor da Web, 184
fechando
aplicativos, 373
conexões de soquete, 463
fluxos, 435
filtrando fluxos, 435, 440-441
implementando interfaces
exemplo, 404-405
interfaces múltiplas, 399-400
interfaces simples, 399
importando
classes, 58, 391
pacote AWT, 266
pacotes, 58
receptores de evento, 531
inicializando applets, 178
iniciando
applets, 178,231
threads, 233
instalando o JDK (Java Development Kit)
testando a instalação, 25
sistemas Solaris, 24-25
Windows 95, 23-24
Windows NT, 23-24
interrompendo
applets, 179,231
threads, 233
lendo fluxos, 434
fluxos colocados em buffer, 441
fluxos de caractere, 449-451
fluxos de dados, 445
fluxos de entrada de arquivo, 436-438
levantando exceções,
definição, 411
explícitas, 420
implícitas, 419
palavra-chave throws, 419
problemas de herança, 421
ocultando
caracteres em campos de texto, 278
janelas, 349
organizando objetos gráficos, 253
passando
argumentos
para aplicativos, 148
para applets, 191-195
exceções, 420
publicando applets, 184
renomeando arquivos, 453
reproduzindo arquivos de som
loops, 260-261
método play(), 260
como sair
de aplicativos, 373
testando
applets, 183-184
instalação do JDK, 25

tratando de eventos, *ver* tratamento de eventos
vinculando
applets, 488-490
exemplo ButtonLink, 489-490
método linkTo(), 490
objetos URL, 488
objetos de nó, 110
componentes
alinhando, 290
inserções, 311-312
layouts de borda, 294-295
layouts de carta, 297-300
layouts de fluxo, 291-292
layouts de grade de conteúdo, 300-312
layouts de grade, 292-294
painéis, 296
II = áreas de texto
controlando, 280
criando, 281
exemplo, 278-281
Swing, 512
associando aos receptores de evento, 531-532
atribuição de nomes, 518-519
balões de descrição, 518
barras de rolagem
criando, 283-284
como gerenciar valores de, 284
Swing, 513
botões de rádio, 512-513
botões
criando, 270-271
rotulando, 270
exemplo, 270-271
Swing, 511
caixas de seleção
criando, 272
marcando/desmarcando, 272
retornando o status de, 272
conjunto Swing, 512-513
exemplo, 272
não-exclusivas, 271
campos de texto
controlando, 279
criando, 277
ocultando caracteres, 277
exemplo, 278
Swing, 511-512
canvas, 284-286
criando descrições para, 518-519
incluindo em aplicativos, 267-268, 508-510
definição, 266
gerenciador de interface com o usuário, 516
JavaBeans
API de interface com o usuário/mesclagem, 486
API de introspecção, 486
API de tratamento de eventos, 486
API persistente, 486
arquitetura, 483
comparado à linguagem Java, 484-485
mecanismo de descoberta de classe, 483
persistência, 483
portabilidade, 483
problemas durante o design, 484
serialização, 484
site da Web, 487
suporte à computação distribuída, 484
listas de escolha
controlando, 276-277
criando, 275-277
incluindo em containers, 275
incluindo itens em, 275
conjunto Swing, 513
exemplo, 276
listas de rolagem
controlando, 282
criando, 281-282
incluindo em containers, 281
incluindo itens em, 281
exemplo, 282
mnemônicos de teclado, 517-518
não-visuais, 485
restrições
anchor, 308
atribuindo, 304
fill, 308
gridheight, 304
gridwidth, 304
gridx, 304
gridy, 304
definindo, 302
weightx, 305
weighty, 305
rótulos
criando, 268-270
conjunto Swing, 511
exemplo, 269
fontes, 269
texto, 269
vantagens, 269
componentes de software, *ver* componentes
computação distribuída, 484
comunicação entre applets
método getApplet(), 492
método getAppletContext(), 492
método getApplets(), 491
condicionais,
definição, 118
if
exemplos, 120
palavra-chave else, 119

operador condicional, 121
switch
exemplos, 124
limitações, 122
conexões com a Web, abrindo, 458-459
configuração de PATH (configuração do JDK)
conflitos de nome (classes), 392
conjunto de caracteres Unicode, 69
códigos de escape, 75
site da Web Unicode Consortium, 76
constantes, 137-138
contineres
classes, 571
componentes, incluindo, 267-268
definição, 266
conversão
texto para letras maiúsculas, 453-455
ver também coerção
conversor RGB para HSB, 362
atualizando resultados
método getHSBColor(), 370
método RGBtoHSB(), 369
método update(), 367-368
inicialização, 363-364
layout do applet, 362-364
listagem do código-fonte
classe ColorControls, 372
classe ColorTest, 371
marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
cor
applet ColorTest, 362
atualizando resultados, 367-370
inicialização, 363-364
layout, 362-364
listagem do código-fonte, 371-372
marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
como atributo de renderização, 218
localizando a cor atual, 216
cores de fundo, 216
pacote java.awt.color, 568-569
ponítilhamento, 215
sistema de cores CMYK, 214
sistema de cores RGB, 215
cores de fundo, 216

D

declarando
arrays
arrays de arrays, 117
classes, 556
constantes, 137-138
métodos, 139-141,557
construtores, 557
exemplo, 139
listas de parâmetro, 140
palavra-chave this, 141
variáveis, 67
múltiplas variáveis, 68
variáveis de instância, 136-137
depurando saída de applet, 197
desalocando memória, 93
desativando
itens de menu, 360
menus, 359
desenhando
arcos, 207
método drawArc(), 207
atributos de renderização
cor, 218
padrões de preenchimento, 219-220
pincéis de desenho, 220-221
elipses
método drawOval(), 206
espaços de coordenadas, 217-218
imagens, 242-243
linhas
método drawLine(), 202
polígonos, 206
Java2D, 223-224
método drawPolygon(), 204
retângulos, 203-204
método drawRect(), 203
texto, 200
design
aplicativos soquete, 465-466
classes, 36-38
grades, 301-302
hierarquias de classe, 51-52
dimensionando quadros, 349
diagnóstico e resolução de problemas
erros Class not found
plataformas UNIX, 624
tratamento de erros tradicional, 410-411
ver também erros
DataBuffer, 576
DataBufferByte, 576
DataBufferlnt, 576
DataBufferShort, 576
DataFlavor, 569
definição, 556
Deflator, 607
DeflatorOutputStream, 607
design, 36-38

Dialog, 564
DigestlnputStream, 598
DigestOmputStream, 598
Dimension, 564
Dimension2D, 573
DirectColorModel, 576
DndConstants, 570
DragSource, 570
DragSourceContext, 570
DragSourceDragEvent, 570
DragSourceDropEvent, 570
DragSourceEvent, 570
DropTarget, 570
DropTargetContext, 570
DropTargetDragEvent, 570
DropTargetDropEvent, 570
DropTargetEvent, 570

E

E comercial (&), 82
Edit do DOS, 628
editores de texto, 26
  associações de arquivo
  Bloco de Notas, 627
  Edit do DOS, 628
elipses, desenhando
  método drawOval(), 206-207
encapsulamento, 379
entrada/saída, *ver* I/O
envio de formulário, 478
erros, 431-432
  classe Error, 412
  mensagens
    Bad Command or File Name, 25
    Class not found, 30-31
  tratamento de erros tradicional, 410-411
  ver também exceções
erros Class not found, 30-31, 393
  plataformas UNIX, 624
erros NullPointerException, 503
escopo
  definição, 118
  classes internas, 407
  escopo de variável, 142-143
espaço de coordenadas de usuário, 217-218
espaços de coordenadas (imagens gráficas), 218
espaços de cor, 214
estado WAITFORANSWER (TriviaServer), 469
estado WAITFORCLIENT (TriviaServer), 469
estado WAITFORCONFIRM (TriviaServer), 469
estilos de junção (pincéis de desenho), 220
estilos de terminação (pincéis de desenho), 220
estrutura de pasta (pacotes), 394
eventos
  SCROLL_ABSOLUTE, 342
  SCROLL_LINE_DOWN, 342
  SCROLLJLINEJJP, 342
  SCROLL_PAGE_DOWN, 342
  SCROLL_PAGE UP, 342
  WINDOW_DEICONIFY, 358
  WINDOW_DESTROY, 358
  WINDOW_EXPOSE, 358
  WINDOWJCONIFY, 358
  WINDOW_MOVED, 358
eventos de ação, tratando de, 338-340, 535
eventos de ajuste, tratando de, 536-538
eventos de clique duplo, 322-323
eventos de clique simples de mouse, tratando de
  exemplo de applet Spots, 319-322
  método mouseDown(), 317-318
  método mouseUp(), 318
eventos de item, 538-540
eventos de mouse, 317-319, 541
  cliques duplos, 322-323
  exemplo de applet Lines, 324-328
  exemplo de applet Spots, 319-322
  métodos
    mouseDown(), 317-318
    mouseDrag(), 323
    mouseEnter(), 324
    mouseExit(), 324
    mouseMove(), 323
    mouseUp(),318
  mouse de vários botões, 334-335
exceção ArraylndexOutOfBounds, 412
exceções
  capturando, 414
    instrução finally, 416-418
    rotinas de tratamento aninhadas, 423
  criando, 422-423
  definição, 409
  EOFException, 413
  erros de compilador, 425
  hierarquia de classe
    classe Error, 412
    classe Exception, 413
    classe IOException, 413
    classe Throwable, 412
  levantando,
    definição, 411
    exceções explícitas, 420
    exceções implícitas, 419
    palavra-chave throws, 419
    problemas da herança, 423

limitações, 424
MalformedURLException, 413
passando, 420
quando usar, 424
runtime, 413
verificação de coerência, 413-414
visão geral, 431
exceções explícitas, 420
exceções implícitas, 419
exemplo de aplicativo getFile
construtor GetFile(), 461-462
listagem do código-fonte, 460-461
método getInputStream(), 462
método readLine(), 462
método run(), 462
objeto BufferedReader, 462
tratamento de erros, 462
exemplo de aplicativo HelloDan
arquivo-fonte, 26-27
compilando
plataformas Windows, 28-30
sistemas Solaris, 30-31
executando
plataformas Windows, 29
sistemas Solaris, 30-31
exemplo de aplicativo SwingColorTest, 514-516
classes, 543
listagem do código-fonte, 544-546
métodos, 543
receptores de evento, 543
exemplo de applet ButtonLink, 489-490
exemplo de applet Fillmore, 244
exemplo de applet Keys, 331-334
definição de classe, 331
listagem do código-fonte, 333
marcação HTML, 334
métodos
init(),331
keyDown(), 332
paint(),332
exemplo de applet Lines
classe Points, 325
listagem do código-fonte, 327-328
marcação HTML, 328
métodos
addLine(), 326
init(), 324-325
mouseDown(), 325
mouseDrag(), 325-326
mouseUp(), 325
paint(), 326-327
exemplo de applet NewPalindrome
arquivo HTML, 194-195
código-fonte Java, 193-194
exemplo de applet Palindrome
arquivo HTML, 182
testando, 183-184
listagem do código Java, 180-181
método paint(), 181
exemplo de applet SetBack, 342
listagem do código-fonte, 345
marcação HTML, 345
métodos
action(), 343-344
changeColor(), 344
init(),343
exemplo de applet Spots
definição de classe, 319
listagem do código-fonte, 321-322
marcação HTML, 322
métodos
addSpot(), 320
init(), 320
mouseDown(), 320
paint(), 320-321
repaint(), 320
exemplo de classe Jabberwock
atributos, 38-39
código-fonte, 42-46
compilando, 44
executando, 44
definição de classe, 41
métodos, 39-40
feedjabberwock(), 42-43, 48
main(), 45
showAttributes(), 43, 47
exemplo de classe Palindrome
classes importadas, 58
código-fonte, 59
compilando, 59
definição de classe, 56-57
marcação HTML, 60
método paint(), 57-58
exemplo de programa DoTest, 130
exemplo de programa EvenSteven, 120
exemplo de programa NamesLoop, 127
exemplo de programa NumberReader, 124
exibindo
ajustes de configuração do JDK
CLASSPATH, 622-623
applets
appletviewer, 60
método paint(), 180
janelas, 349
expressões
definição, 76
valores de retorno, 76
ver também operadores

F

ferramenta appletviewer, 60
ferramentas
  CABarc, 190
  jar, 188
  keytool, 427
  policytool, 430
  arcos
    desenhando, 207
    preenchendo, 207
    exemplo, 208-209
  elipses, 206-207
  linhas, 202-203
  polígonos
    desenhando, 204
    incluindo pontos em, 205
  retângulos
    cantos arredondados, 204
    desenhando, 203
    preenchendo, 203
filtros de byte, 441
fluxos
  abrindo através da Internet
    exemplo de aplicativo GetFile, 460-461
    método getInputStream(), 459
    objetos BufferedReader, 459
    objetos de conexão de URL, 459
  colocados em buffer
    criando, 441-442
    escrevendo em, 442-444
    lendo, 441
  criando, 434
  escrevendo em, 435
  fechando, 435
  filtrando, 435, 440-441
  lendo, 434
  definição, 434
  fluxos de byte, 436
    fluxos de entrada de arquivo, 436-438
    fluxos de saída de arquivo, 439-440
  fluxos de caractere, 434, 448-449
    escrevendo arquivos de texto, 451-452
    lendo arquivos de texto, 449-451
  fluxos de dados
    criando, 445
    lendo, 445
    exemplos de aplicativos, 446-447
fluxos de byte, 434-435
  fluxos de entrada de arquivo
    criando, 436
    lendo, 436-438
  fluxos de saída de arquivo
    criando, 439-440
    escrevendo em, 439-440
fluxos de dados
  criando, 445
  lendo, 446
  exemplos de aplicativo
    ReadPrimes, 447
    WritePrimes, 446-447
fluxos de entrada de arquivo
  criando, 436-438
  lendo
    arquivo class.dat, 438
    exemplo de aplicativo, 437
    loops while, 437
    método read(), 436
fluxos de saída de arquivo
  criando, 439-440
  escrevendo em
    exemplo de aplicativo, 439
    método write(), 439
foco
  classe FocusAdapter, 571
  classe FocusEvent, 571
  tratamento de eventos
    FocusListener, 530
    método focusGained(), 538
    método focusLost(), 538
    método gotFocus(), 340-341
    método lostFocus( ),340-341
fontes, 572
  definindo, 211-212
  fontes de rótulo, 269
  objetos Font, 211
formatos de arquivamento
  CAB, 190
  JAR, 189-190
  Zip, 189-190
função membro, *ver* métodos
funções, *ver* métodos
futuro da linguagem Java, 15-16

G

geometria, *ver* figuras
geração de número pseudo-aleatório, 91
geradores de números aleatórios, 91
gerenciador de layout de grade de conteúdo
  criando, 300-301
  comparado ao gerenciador de layout de grade, 300
  componentes, organizando, 307-309
  exemplo de layout, 309-311
  grades
    criando, 302-305
    design, 301-307
    proporções, 305-307
  preenchimento de célula, 311

gerenciadores de layout
alternativas aos, 312-313
combinando, 295-296
criando, 290
especificando, 290-291
definição, 266
inserções, 311-312
layout de borda, 294-295
layout de carta
exibindo cartas, 297
criando, 297
incluindo cartas, 297
exemplo, 298-299
layout de fluxo
construtor FlowLayout(), 292
exemplo, 291
layout de grade, 292-294
criando, 292
comparado ao gerenciador de layout de grade de conteúdo, 300
exemplo, 293
layout de grade de conteúdo
criando grades, 302-305
determinando proporções, 305-307
organizando componentes, 307-309
design de grades, 301-302
exemplo, 309-310
preenchimento de célula, 311
visão geral, 300-301
gerenciamento de memória, 92-93
Gosling, James, 13
grades
criando, 302-305
gerenciador de layout de grade, 292-294
gerenciador de layout de grade de conteúdo, 302-305
design, 301-302
preenchimento de célula, 311
proporções de fileira/coluna, 305-307
gradientes acíclicos, 219
gradientes cíclicos, 219
GUIs (interfaces gráficas com o usuário), 266, 505
aplicativos AWT independentes, 360-361
caixas de diálogo
anexando a applets, 356
criando, 519-520
definição, 352
caixas de diálogo de confirmação, 520-521
caixas de diálogo de entrada, 521-522
caixas de diálogo de mensagem, 522
caixas de diálogo de opção, 522-523
exemplo TextDialog, 354-355
janelas de diálogo, 352
modais, 352
objetos de diálogo de arquivo, 357
componentes
áreas de texto, 279-281
barras de rolagem, 283-284
botões, 270-271
caixas de seleção, 271-274
campos de texto, 277-279
canvas, 284-286
incluindo em containers, 267-268
listas de escolha, 275-277
listas de rolagem, 281-282
rótulos, 268-270
diferenças de plataforma, 266
exemplo de applet ColorTest, 362
atualizando resultados, 367-370
inicialização, 363-364
layout, 362-364
listagem do código-fonte, 371-372
marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
gerenciadores de layout
alternativas aos, 312-313
combinando, 295-296
criando, 290
especificando, 290-291
inserções, 311-312
layout de borda, 294-295
layout de carta, 297-300
layout de fluxo, 291-292
layout de grade de conteúdo, 300-312
layout de grade, 292-294
visão geral, 290
janelas
aviso Unsigned Java Applet Window, 349
classe Window, 348
eventos, 357-358
exemplo de janela instantânea, 350-351
exibindo, 349
ocultando, 349
menus
ativando, 359
barras de menu, 358-359
desativando, 359
eventos, 360
itens de menu, 359-360
quadros
criando, 348
definição, 348
dimensionando, 349
exemplo BaseFrame, 351
recursos de acessibilidade, 567.-568
Swing, 505
aplicativo SwingColorTest, 514-516

áreas de texto, 512
balões de descrição, 518
barras de rolagem, 513
botões, 511
botões de rádio, 512-513
caixas de seleção, 512-513
campos de texto, 511-512
descrições de componente, 518
estrutura de aplicativo, 507-508
gerenciador de interface com o usuário, 516
listas de escolha, 513
mnemônicos de teclado, 517-518
nomes de componente, 518-519
painéis de conteúdo, 508
rótulos, 511
vantagens, 506
tratamento de eventos
aplicativo SwingColorTest, 542-547
cliques de mouse, 317-323, 541
configuração de componente, 531-532
eventos de ação, 338-340, 535
eventos de ajuste, 536-538
eventos de área de texto, 341
eventos de barra de rolagem, 342
eventos de foco, 340-341, 538
eventos de item, 538-540
eventos de janela, 542
eventos de lista de rolagem, 341
eventos de teclado, 328-335, 540-541
exemplo de aplicativo ChangeTitle, 533-534
exemplo de applet SetBack, 342-345
método handleEvent(), 316-317, 335-337
métodos, 532-534
movimentos de mouse, 323-328, 541-542
receptores de evento, 530-531
tipos de evento, 316

H

herança
controle de acesso, 383
definição, 48
exceções, 421
hierarquias de classe
criando, 50-52
design, 52
métodos, 53
múltipla, 54, 397
simples, 54, 397
subclasses, 55-61
visão geral, 48-50
hífen (-), 80
história do desenvolvimento da linguagem Java
endosso da Netscape, 14
navegador Hotjava, 14
projeto Green, 13-14
versões de Java, 14-15
HTML (Hypertext Markup Language)
envio de formulário, 478
tags
<APPLET>, 182-185

, 185
<OBJECT>, 187-188
<PARAM>, 191-192

I

I/O (entrada/saída)
fluxos
colocados em buffer, 441-445
criando, 434
escrevendo em, 435
definição, 434
fechando, 435
filtrando, 435, 440-441
fluxos de byte, 436
fluxos de caractere, 434, 448-452
fluxos de dados, 445-448
fluxos de entrada de arquivo, 436-438
fluxos de saída de arquivo, 439-440
lendo, 434
interfaces, 586-587
ícones
criando, 510
definição, 510
site da Web Pardon My Icons!, 511
imagens
caminhos de arquivo relativos, 241-242
carregando, 240-2411, 246-247
desenhando, 242-243
observadores, 243
recuperando, 240
ver também imagens gráficas
imagens gráficas
animação
applet Neko, 245-253
caminhos de arquivo relativos, 241-242
carregamento de imagem, 240-241
desenhando imagens, 242-243
disposição de objeto gráfico, 253-254
observadores de imagem, 243
pintura/repintura, 230-231
recuperação de imagem, 240
threads, 231-237
tremido, 236-240

uso de buffer duplo, 254-258
visão geral, 229-230
arcos
desenhando, 207
exemplo, 209
preenchendo, 207
atributos de renderização
cor, 218
padrões de preenchimento, 219-220
pincéis de desenho, 220-221
componentes de GUI
áreas de texto, 279-281
barras de rolagem, 283-284
botões, 273-271
caixas de seleção, 271-274
campos de texto, 277-279
canvas, 284-286
listas de escolha, 275-277
listas de rolagem, 281-282
rótulos, 268-270
espaços de coordenadas, 219
exemplo Map2D, 224-226
funções de cópia/colagem, 210-211
ícones
criando, 510
definição, 510
site da Web Pardon My Icons!, 511
linhas, 202-203
objetos Graphics2D, 218
ovais, 206
polígonos
desenhando, 204
incluindo pontos em, 205
retângulos
cantos arredondados, 204
desenhando, 203
preenchendo, 203
texto, 200
visão geral, 199
ver também AWT; GUIs
imagens gráficas bidimensionais, 200
applet Map2D, 224-227
arcos
desenhando, 207
preenchendo, 207
exemplo, 208-209
atributos de renderização
cor, 218
padrões de preenchimento, 218-220
pincéis de desenho, 220-221
espaços de coordenadas, 217-218
funções copiar/colar, 210-211
linhas, 202-203
objetos Graphics2D, 218
ovais, 206
polígonos
desenhando, 204
inserindo pontos em, 205
retângulos
cantos arredondados, 204
desenhando, 203
preenchendo, 203
implementação de soquete de camada de transporte, 464
independência de plataforma
máquina virtual, 17-18
no código-fonte, 18
problemas de desempenho, 19-20
vantagens, 16-20
inserções
classe Insets, 565
método insets(), 311
instâncias, *ver* objetos
instruções
blocos, 117-118
caractere de terminação, 66
condicionais,
definição, 118
operador condicional, 121
switch, 121-125
expressões, 76
finally, 416-418
ICC_ProfileRGB, 568
import, 391
loops
do, 129-130
valores de índice, 126
package, 394
instruções if
exemplos, 120
palavra-chave else, 119
instruções switch
exemplos, 122-124
limitações, 122
visão geral, 121-125
interfaces, 62, 396
Accessible, 567
AccessibleLayout, 568
AccessibleText, 568
Activationlnstantiator, 594
ActivationMonitor, 594
ActivationSystem, 594
Activator, 594
ActiveEvent, 561
Adjustable, 561
ampliando, 403-404
AppletContext, 561
AppletStub, 561
AudioClip, 561
BeanContext, 586
BeanContextChild, 586
Certificate, 597

ClipboardOwner, 495
coerção de objetos para, 401
como tipo de variável, 400
comparadas às classes, 398
Composite, 518
CompositeContext, 561
ContentHandlerFactory, 592
Datalnput, 586
DataOutput, 586
DGC, 595
DragSourceListener, 569
DropTargetListener, 569
EventSource, 562
Externalizable, 586
FilenameFilter, 586
FileNameMap, 592
FlavorMap, 569
Guard, 597
ImageObserver, 243
implementando
exemplo, 404-405
interfaces múltiplas, 399-400
interfaces simples, 398-399
ItemSelectable, 562
Key, 597
LayoutManager, 562
LayomManager2, 562
LoaderHandler, 596
MenuContainer, 562
MultipleMaster, 572
Objectlnput, 586
ObjectlnputValidation, 586
ObjectOutput, 586
OpenType, 572
Paint, 562
PaintContext, 562
Pathlterator, 572
Principal, 597
PrintGraphics, 562
Registry, 595
RegistryManager, 595
RemoteCall, 596
RemoteRef, 596
Replaceable, 586
Resolvable, 586
RMIFailureHandler, 596
Runnable, 232
Serializable, 586
ServerRef, 596
Shape, 562
Skeleton, 596
SocketlmplFactory, 592
Stroke, 562
Transferable, 569
Transparency, 562
Unreferenced, 596
URLStreamHandlerFactory, 592
WindowConstants, 579
interfaces com o usuário, *ver* GUIs
interfaces gráficas com o usuário, *ver* GUIs
interligação em rede
aplicativo Trivia, 464-465
design, 465-466
executando, 477-478
implementação do cliente, 475-477
implementação do servidor, 466-474
conexões na Web, 458-459
definição, 457
fluxos
exemplo de aplicativo GetFile, 460-461
rtiêtodo getInputStream(), 459
olpjetos BufferedReader, 459
objetos URLConnection, 459
pacote java.net
interfaces, 592-593
soquetes
implementação da camada de transporte, 464
no lado do cliente, 463
no lado do servidor, 463-464
Internet Explorer, 427
introspecção, *ver* reflexão

J

janelas
aviso Unsigned Java Applet Window, 349
classe Window, 348
eventos, 357-358
exemplo de janela instantânea, 350-351
exibindo, 349
ocultando, 349
quadros
criando, 348
definição, 348
dimensionando, 349
exemplo BaseFrame, 351
tratamento de eventos, 542
JARS (Java Applet Rating Service), 612
Java Development Kit, *ver* JDK
Java Plug-in, 19
Java2D, 217
atributos de renderização
cor, 218
exemplo de applet, 224-227
padrões de preenchimento, 218-220
pincéis de desenho, 220-221
espaços de coordenadas, 217-218
objetos Graphics2D, 218

JavaBeans
API de interface com o usuário/mesclagem, 486
API de intropecção, 486
API de tratamento de eventos, 486
API persistente, 486
classes,
comparado à linguagem Java, 484-485
definição, 483
mecanismo de serviços, 585-586
site da Web, 487
vantagens
arquitetura compacta, 483
mecanismo de descoberta de classe, 483
persistência, 483
portabilidade, 483
propriedades em tempo de design, 484
senahzação, 484
suporte à computação distribuída, 484
javac
plataformas Windows, 28
sistemas Solaris, 30
JavaScript, 32
JDBC (Java Database Connectivity), 502-503
JDK (Java Development Kit)
carregando por download, 22-23
instalando
sistemas Solaris, 24-25
testando a instalação, 25
Windows 95, 23-24
Windows NT, 23-24
configuração para UNIX, 624
configuração para Windows 95/Windows NT,
instrução CLASSPATH, 622-624
disponibilidade, 15
Setup Wizard, 23-24
visão geral, 22-23
JFC (Java Foundation Classes)
acessibilidade, 567-568
arrastar e soltar, 16
carregando por download, 505
Swing,
aplicativos, 507-508, 514-516
caixas de diálogo, 519-526
classes, 580-582
componentes, 508-513
definição, 16
gerenciador de interface com o usuário, 517
mnemômcos de teclado, 517-518
vantagens, 506
visão geral, 505-506
ver também AWT; imagens gráficas

L

letras maiúsculas, convertendo texto para, 453-455
linha de status (applets), 487
linhas, desenhando
método drawLine(), 202
listagens de código, *ver* listagens
listagens de código-fonte, *ver* listagens
listagens de programa, *ver* listagens
listagens
aplicativo AllCapsDemo, 454-455
aplicativo BaseFramel, 351
aplicativo CopyPaste, 496-497
aplicativo EchoArgs, 148
aplicativo GetFile, 460-461
aplicativo HelloDan, 27
aplicativo LinkedList, 404-405
aplicativo PopUpWindow, 350-351
aplicativo SumAverage, 149-150
aplicativo Trivia
cliente, 475-476
servidor, 471-474
aplicativo WritePrimes, 446-447
applet ButtonLink, 489-490
applet Checkers, 257-258
applet ColorSwirl, 238-239
applet ColorTest
classe ColorControls, 372
classe ColorTest, 371
applet DigitalClock, 234-235
applet Fillmore, 244
applet Keys, 333
applet Map
arquivo HTML, 200
código-fonte Java, 200, 209
applet Map2D
arquivo HTML, 226
código-fonte Java, 224-225
applet Neko, 251-253
applet SetBack, 345
applet Spots, 321-322
blocos try...finally, 417
classe CountInstances, 385
classe MyRect2, 161-162
classe NamedPoint, 167
classe PrintClass, 163
classe PnntSubClass, 163
classe PrintSubClass2, 164
classe TextDialog, 354-355
classes internas, 406
componentes de GUI
áreas de texto, 280
barras de rolagem, 283-284

botões de rádio, 274
botões, 270-271
caixas de seleção, 272
campos de texto, 278
canvas, 284-285
listas de escolha, 276
listas de rolagem, 282
rótulos, 269
controle de acesso de pacote, 395-396
exemplo de applet NewPalindrome
arquivo HTML, 194-195
código-fonte Java, 193-194
exemplo de applet Palindrome
arquivo HTML, 182
código-fonte Java, 180-181
exemplo de aritmética simples, 77-78
exemplo de classe FontMetrics, 213
exemplo de classe Palindrome
código-fonte, 59
marcação HTML, 60
exemplo de método construtor, 159
exemplo Jabberwock
aplicativo, 45-46
classe, 42-44
fluxos
aplicativo BufferDemo, 442-444
aplicativo ReadBytes, 437
aplicativo ReadPrimes, 447
aplicativo ReadSource, 450-451
aplicativo WnteBytes, 439
aplicativo WntePrimes, 446-447
gerenciadores de layout
layout de borda, 295
layout de carta, 298-299
layout de fluxo, 291
layout de grade, 293
layout de grade de conteúdo, 309-310
instruções
if, 120
switch, 124
loops de áudio, 260-261
loops
do...while, 130
while, 128
método printMe(), 165
métodos
chamando, 96-97
definindo, 140
objetos
comparando, 105-106
criando, 91
programa ScopeTest, 142
reflexão, 107
Swing
aplicativo Info, 524-525
aplicativo SwingColorTest, 514-515
aplicativo Swinger, 509
estrutura de aplicativo, 507
tagOBJECT, 187-188
tratamento de eventos
aplicativo ChangeTitle, 533-534
aplicativo SelectItem, 539-540
aplicativo SwingColorTest, 544-546
aplicativo WellAdjusted, 536-537
listas
listas de escolha
controlando, 276-277
criando, 275-277
incluindo em containers, 275
incluindo itens em, 275
exemplo, 276
itens, incluindo, 275
Swing, 513
listas de rolagem
controlando, 282
criando, 281-282
incluindo em containers, 281
incluindo itens em, 281
exemplo, 282
listas encadeadas, enumerando, 404-405
listas suspensas
controlando, 276-277
criando, 275-277
exemplo, 276
incluindo em containers, 275
itens, incluindo, 275
literais,
booleanas, 74
códigos de escape, 75
definição, 72
inteiros, 73-74
strings, 75-76
literais de caractere, 74-75
ver também conjunto de caracteres Unicode
literais inteiras, 70, 73-74
literais numéricas, 73
livros (relacionados à linguagem Java), 609-610
loops,
do...while, 129-130
for
erros comuns, 126
exemplo, 127
instruções vazias, 126
visão geral, 125-127
valores de índice, 126

M

máquina virtual
  suporte de navegador a, 19-20
  visão geral, 17-18
mensagem de aviso Unsigned Java Applet Window, 349
mensagem de erro Bad command or file name, 26
menus
  ativando, 359
  barras de menu, 358-359
  classes, 566
  desativando, 359
  definição, 358
  eventos, 360
  itens de menu
    criando, 359-360
    tipos, 359
métodos, 40, 62
  action()
    argumentos, 339
    definindo, 338
    valor de retorno, 339
  actionPerformed(), 532, 535
  addActionListener(), 535
  addlteni(),275
  addLine(), 326
  addPoint(), 205
  addSpot(), 320
  adjustmentValueChanged(), 536
  anulando, 53, 162-165
    palavra-chave super, 165
    vantagens, 165
  ArgStream(), 444
  assinaturas, 153
  Borderl_ayout(), 294
  BufferedInputStream(), 441
  BufferedOutputStream(), 442
  BufferedReader(),450
  BufferedWriter(),452
  buildConstraints(), 302-307
    sobrecarregando, 156
  button(), 270
  CardLayout(),297
  changeColor(),344
  charWidthQ, 213
  checkbox(),272
  evenCheck (), 120
  clearRectQ, 210
  close(), 435
  closePath (),223
  definindo,
    exemplo, 139
    listas de parâmetros, 140
    palavra-chave this, 141
construtores, 92, 158
  anulando, 166-168
  chamando a partir de outro construtor, 160
  definindo, 557
  exemplo, 159
  nomes, 159
  sobrecarregando, 160-162
controle de acesso, 378-384
  acesso padrão, 379
  acesso privativo, 379-380
  acesso protegido, 381-382
  acesso público, 381
  herança, 383
convertlt(), 123
convertNum(), 124
copyArea(),210
countltems(), 276
DataFlavor(),494
DataInputStream(), 445
DataOutputStream(), 445
delete(),453
deleteOnExit(), 453
destroy(), 179
Dialog(), 353
disable(), 359
dispose(), 253
drawArc(),207
drawlmage(), 242-243
drawLine(), 202
drawOval(), 206
drawPolygon(), 204-205
drawRectO, 203
drawRoundRect(), 203
drawString(), 200
enable(), 359
equals(), 105
exists(), 453
feedjabberwock(), 42
File(), 453
FileDialog(), 357
FileInputStream(), 436
FileOutputStream(), 449
FileReader(), 449
FileWriter(), 451
fillArc(), 207
fillOval(),206
fillPolygon(), 204-205
fillRect(), 203
fillRoundRect(), 203
finalize(), 179
flowLayout(), 291-292
focusGained(), 538

focusLost(), 538
Font(),212
Framework(), 507
getAccessibleContext(), 519
getActionCommand(), 535
getAdjustmentType(), 536
getAppletContext(), 487, 491
getAppletlnfoQ, 487
getAudioClip(),259
getClassQ, 107
getClickCount(), 541
getCodeBase(),242
getColor(),217
getCrossPlatformLookAndFeel
ClassName(),517
getCurrem(), 274
getDocumentBase(), 242
GetFile(),461
getFontMetrics(), 213
getHeight(),213, 243
getHSBColor(), 370
getInputStream(), 459
getlnsetsQ, 311
getltem(), 276
getKeyChar(), 541
getLabel(), 270
getParameter(), 192
getParent(), 356
getPoint(), 541
getSelectedIndex(), 279
getSelectedIndexes(), 282
getSelectedItem(), 277
getSelectedItems(),282
getSize(),211
getSource(),532, 535
getState(), 272
getStateChange(), 539
getSystemClipBoard(), 494
getSystemLookAndFeelClassName(),
517
getText(), 269, 279
getToolkit(),494
getValue(),284
getWidth(),243
getX(), 541
getY(), 541
gotFocus(), 340
handleDeselect(), 341
handleEvent(), 316-317, 335-337
handleSelect(), 341
hide(), 349
init()
applet Keys, 331
applet Lines, 324
anulando, 178
exemplo de applet Spots, 320
initQnA(), 470
insertText(), 280-281
insets(),311
isEditable(),279
itemStateChanged(), 539
JButton(), 511
JCheckBox(), 512
JComboBox(), 513
JLabelQ, 511
JScrollBar(), 513
JTextArea(), 512
JTextField(),512
keyDown(), 330
keyPressed(),540
keyReleased(), 540
keyTyped(), 540
keyUp(), 329
label(),269
lineTo(),223
linkTo(),490
list(),281
loop(), 260
lostFocus(), 340
lostOwnership(), 495
main ()
aplicativo TriviaServer, 470
assinatura, 146
exemplo Jabberwock.java, 45-46
makeRange(), 140-141
métodos abstratos, 388-389
métodos de acesso, 383-384
métodos finais, 387
mkdir(), 453
mouseDown()
applet Lines, 325
applet Spots, 320
eventos de clique duplo, 322-323
eventos de clique simples, 317
mouseDrag(), 323
mouseDragged(), 541
mouseEnter(), 324
mouseExit(), 324
mouseMove(), 323
mouseMoved(), 541
mouseReleased(), 541
mouseUp(),318
moveTo(), 223
nekoRun(),247
nekoScratch(), 249
nekoSleep(),250
newAudioClip(), 259
newFrame(), 348
newLine(), 452
nextDouble(),91

pack(), 349
paint(), 57, 230-231
  applet Keys, 332
  applet Lines, 326-327
play(),259
POST, 478
printMe(), 165
primNames(), 116
processlnput(), 468-469
read(), 434
readFloat(),445
readLine(), 450, 462
readStream(), 444
readUnsignedByte(), 445
readUnsignedShort(), 445
renameTo(), 453
repaint(), 230-231
replaceText(), 281
requestFocus(), 332
reshape(), 313
resize(),349
RGBtoHSB(), 369
run()
  aplicativo TriviaServer, 4667-468
  applets com threads, 233
scrollbar(), 283
select(),277
setAccessibleDescription(), 519
setAccessibleName(), 519
setActionCommand(), 535
setBackground(), 216
setColor(),216, 218
setConstraints(), 304
setContentPane(), 508
setCurrent(),274
setEchoChar(),512
setEchoCharacter(), 277
setEditable(),279
setFont(), 269
setForeground(), 216
setHelpMenu(),359
setLabel(),270
setLayout(),290, 296
setLookAndFeel(),517
setMenuBar(), 358
setMnemonic(), 518
setPaint(),218
setState(),272
setStroke(),220
setText(),279, 538
setToolTipText(),518
setValue(), 284
setVisible(),508
show(),298
showAttributes (), 43
showConfirmDialog(), 520
showInputDialog(), 521
showMessageDialog(), 522
showOptionDialog(), 522
showStatus(), 487
size(),211
sobrecarregados
  criando, 154-158
  construtores, 160-162
  vantagens, 153
start(), 178,231
  anulando, 178
static, 384-386
stop(), 179,231
  anulando, 179
stringWidth(),213
testBlock(), 118
textArea(), 279
textField(),277
tipos de retorno, 139
update(), 237, 368
URL(),488
wmdowClosed(), 542
windowClosing(), 542
windowOpened(), 542
write(), 435, 451
writelnt(), 448
writeStream(), 444
métodos abstratos, 388-389
métodos de acesso, 383-384, 408
métodos de entrada,
métodos de instância, *ver* métodos
métodos estáticos, 384-386
métodos finais, 387
métodos finalizadores, 153
modificador abstract, 388
modificador private, 380
modificador protected, 381
modificadores, *ver* palavras-chave
MS-DOS
  comandos
    ed, 28
    dir, 29
  prompt de comando, 25
multimídia
  animação
    applet Neko, 245-253
    caminhos de arquivo relativos, 241-242
    carregamento de imagens, 240-241
    desenhando imagens, 242-243
    disposição de objetos gráficos, 253-254
    observadores de imagem, 243
    pintura/repintura, 230-231
    recuperação de imagem, 240
    threads, 231-236

tremido, 236-240
uso de buffer duplo, 254-258
visão geral, 229-230
som
formatos de arquivo, 258
loops, 260-261
reproduzindo, 259
ver também imagens gráficas; GUIs
multitarefa, 231

N

Naughton, Patrick, 14
navegador Hotjava, 14
navegadores
aprimoramento de navegador Plug-in, 19
assinaturas específicas do navegador, 429-430
Hotjava, 14
Java Plug-in, 176
Netscape Navigator, 429-430
Netscape Signing Tool, 430
newsgroups, 32, 613-614
newsgroups da Usenet, 32, 613-614
notação exponencial, 73
NotePad, 628
NT
aplicativos Java
compilando, 28-30
executando, 29
configuração do JDK, 23-24
erros Class not found,
instrução CLASSPATH, 622-624
instrução PATH,
números em ponto flutuante, 70
números hexadecimais, 73
números negativos, representando como literais, 73
números octais, 73

O

objetos,
CheckboxGroup, 273
coerção
para interfaces, 105-106
comparando, 105-106
criando
argumentos, 90
construtores, 92
gerenciamento de memória, 92-93
operador new, 90-92
definição, 35, 61
determinando a classe de, 107
File, 453
Font, 211
GeneralPath, 223
Graphics2D, 218
nós, vinculando, 110
Random, 91
transferíveis
definição, 493
strings de texto, 494
versões de dados, 493-494
URL, 488
ver também classes
opção -f (comando jar), 189
operações de colagem
Área de Transferência, 494-498
exemplo de aplicativo, 496-497
interface ChpboardOwner, 495
objetos transferíveis
definição, 493
strings de texto, 494
versões de dados, 493-494
suporte a, 492-493
operações de cópia/colagem
Área de Transferência, 494-498
exemplo de aplicativo, 496-497
interface ClipboardOwner, 495
objetos transferíveis
definição, 493
versões de dados, 493-494
strings de texto, 494
suporte a, 492-493
operações de recorte/colagem
Área de Transferência, 494-498
exemplo de aplicativo, 496-497
interface ClipboardOwner, 495
objetos transferíveis
definição, 493
strings de texto, 494
versões de dados, 493-494
suporte a, 492-493
operador de concatenação (+), 85
operador de decremento (—), 80
operador de desigualdade (! =), 105
operador de igualdade (= =), 105
operador de incremento (+ +), 80
operador instanceof, 107, 533
operador new,
criando objetos, 90-92
operador NOT, 82
operador ternário, 121
operador XOR, 82
operadores,
comparação, 81
concatenação (+), 85
condicionais, 121
decremento (—), 80
desigualdade (! =), 105

igualdade (= =), 105
incremento (+ +), 80
instanceof, 107
new,
criando objetos, 90-92
operadores AND, 82
operadores OR, 82
oportunidades de emprego (Java), 614-615
ovais, desenhando
método drawOval(), 206-207
overflow (atribuição de variável), 86-87

P

pacote de acessibilidade
classes, 568
interfaces, 567
pacotes
criando
classes, 394
estrutura de pasta, 394
nomes, 393-394
fazendo referência
nomes de classe, 390
nomes de pacote, 391
importando, 58, 391
controle de acesso
acesso padrão, 394
acesso público, 394-396
definição, 62
vantagens, 389-390
visão geral, 55, 389-390
ver também classes; interfaces
pacotes personalizados
atribuição de nomes, 393-394
classes, incluindo, 394
controle de acesso
acesso padrão, 394
acesso público, 395-396
estrutura de pasta, 394
padrões de preenchimento
arcos, 207
elipses, 206
Java2D, 218-220
polígonos, 205
retângulos, 203
painéis
criando, 296
exemplo de applet ColorTest, 365
inserções, 311
palavras-chave
abstract, 388
break, 131
else, 119
extends, 403
final, 137
implements, 398
instanceof, 533
null, 126
private, 380
protected, 381
public, 56, 381
static, 138, 384
super, 165
this, 141, 160, 233
throws, 419
parâmetros, passando para applets
exemplo, 193-195
método getParameter(), 192
tag < PARAM >, 192
valores nulos, 192, 197
ver também argumentos
pares, 313
pastas, criando/excluindo, 453
permissões, 427
persistência, 483
Plug-in, 19
polígonos
desenhando
método drawPolygon(), 204
incluindo pontos em, 205
pontilhamento, 215
ponto de exclamação (!), 82
portabilidade da linguagem Java
código-fonte, 18
máquina virtual, 17-18
problemas de desempenho, 19
vantagens, 16-20
possibilidades de carreira, 614
preenchimento de célula, 311
preenchimentos de gradiente, 219
problemas no uso de várias plataformas, 11
programa CopyArrayWhile, 128-129
programa Personal Bookshelf, 275
programa rmiregistry, 502
programa ScopeTest, 146
projeto Green, 13-14
prompt de comando, 25

Q-R

quadros
criando, 348
dimensionando, 349
quebras de linha, marcação HTML, 185
receptores de evento, 530-531
ActionListener, 535
AdjustmentListener
exemplo de programa, 536-537

método adjustmentValueChanged(), 536
associando componentes aos, 531-532
definição, 529
FocusListener, 538
importando, 531
ItemListener
exemplo de programa, 539-540
método itemStateChanged(), 539
KeyListener, 540-541
MouseListener, 541
MouseMotionListener, 541-542
WindowListener, 542
recursos da Internet (relacionados à linguagem Java)
newsgroups, 613-614
sites da Web
Gamelan, 612
JARS, 612
Java FAQs, 613
JavaSoft, 610-611
JavaWorld, 612-613
livros relacionados à linguagem Java, 611-612
Macmillan Computer Publishing, 610
oportunidades de emprego, 614
Sams Teach Yourself Java 2, 609
recursos da linguagem Java (visão geral), 11
reflexão,
classes, 592
exemplo, 107-108
interfaces, 592
pacote java.lang.reflect, 109
Remote Method Invocation, *ver* RMI
restrições
anchor, 308
atribuindo a componentes, 303
fill, 308
gridheight, 304
gridwidth, 304
gridx, 304
gridy, 304
weightx, 305
weighty, 305
retângulos
cantos arredondados, 204
desenhando
método drawRect(), 203
preenchendo, 203
RMI (Remote Method Invocation)
aplicativos, 501-502
arquitetura, 499-501
pacote java.rmi.activation, 595
pacote java.rmí.dgc, 595-596
pacote java.rmí.registry, 596-597
pacote java.rmi.server
classes, 596
interfaces, 500
serialização, 502
site da Web, 502
visão geral, 498-499
rótulos
criando, 268-270
exemplo, 269
fontes, 269
rótulos de botão, 270
Swing, 511
texto, alterando, 269
vantagens, 269
RPC (chamadas de procedure remotas), 499

S

saída, *ver* I/O
segurança
assinaturas digitais
autoridades de certificado, 426
certificados, 426
específicas do navegador, 429-430
exemplo, 427-429
newsgroups, 613-614
pacote java.security
interfaces, 597-598
planos, 430
separadores (menus), 360
serialização (RMI), 484, 500
servidores (aplicativo Trivia)
arquivo QnA.txt, 469
construtor, 466-467
estado WAITFORANSWER, 469
estado WAITFORCLIENT, 469
estado WAITFORCONFIRM, 469
executando, 477-478
listagem do código-fonte, 471-474
método initQnA(), 470
método main(), 470-471
método processlnput(), 468-469
método run(), 467-468
variáveis de instância, 466
Signing Tool (Netscape), 430
simplicidade da linguagem Java, 20-21
sinal de adição (+)
operador de concatenação (+), 85
operador de incremento (+ +), 80
sinal de igualdade (=)
operador de atribuição, 68
operador de igualdade, 105

sinal de inclusão (^), 82
sistema de cores CMYK, 214
sistema de cores HSB (matiz, saturação, brilho), convertendo para, 362
atualizando resultados
método getHSBColor(), 362
método RGBtoHSB(), 370
método update(), 368
inicialização, 363-364
layout de applet, 362-364
listagem do código-fonte
classe ColorControls, 372
classe ColorTest, 371
marcação HTML, 372
subpainéis, 364-367
tratamento de eventos, 367
sistema de cores RGB (vermelho, verde, azul), 214
sistema de numeração de base 8, 73
sistema de numeração de base 16, 73
sistema operacional Star7,13
sistemas Solaris
aplicativos Java, compilando/executando, 30-31
instalação do JDK, 24-25
sistemas SPARC, instalação do JDK, 24
sites da Web
applet Dominós, 12
Sams Teach Yourself Java 2, 609
documentação da linguagem Java 2, 72
FAQs sobre a linguagem Java, 613
Gamelan, 32
JARS, 612
JavaWorld, 612-613
JDK, 22
livros relacionados à linguagem Java, 611-612
Macmillan Computer Publishing, 275
Microsoft, 190
Neil/Fred's Gigantic List of Palindromes, 61
Netscape Signing Tool, 430
oportunidades de emprego, 614-615
página de recursos Code Signing, 430
Pardon My Icons!,511
Unicode Consortium, 76
Unofficial Obscure Java FAQ, 613
som
formatos de arquivo, 258
reproduzindo
loops, 260-261
método play(), 260
soquetes TCP (Transmission Control Protocol)
aplicativo Trivia
design, 465-466
executando, 477-478
implementação do cliente, 475-477
implementação do servidor, 466-474
implementação da camada de transporte, 464
no lado do cliente, 463
no lado do servidor, 463
strings, 75-76
classes
StringSelection, 494
concatenando, 84
nulos, 538
Stroustrup, Bjarne, 13
subclasses, 50
criando, 55-61
definição, 498 62
superclasses
definição, 49, 62
Swing,
aplicativos
estrutura, 507-508
SwingColorTest, 514-516
caixas de diálogo
caixas de diálogo de confirmação, 520-521
caixas de diálogo de entrada, 521-522
caixas de diálogo de mensagem, 522
caixas de diálogo de opção, 522-523
criando, 519-520
classes, 580-582
componentes
áreas de texto, 512
balões de descrição, 518
barras de rolagem, 513
botões de rádio, 512-513
botões, 511
caixas de seleção, 511-513
campos de texto, 511-512
atribuindo nomes, 518-519
criando descrições para, 518-519
incluindo em aplicativos, 508-512
listas de escolha, 513
rótulos, 511
visão geral, 510-511
definição, 16
gerenciador de interface com o usuário, 516
mnemônicos de teclado, 517-518
tratamento de eventos, 529
configuração de componente, 531-532
eventos dé ação, 535
eventos de ajuste, 536-538
eventos de foco, 538
eventos de item, 538-540
eventos de janela, 542
eventos de mouse, 541
eventos de tecla, 540-541
exemplo ChangeTitle.java, 533-534
exemplo de aplicativo SwingColorTest, 542-547

métodos, 532-534
palavra-chave instanceof, 533
receptores de evento, 530-531
vantagens, 506
visão geral, 505-506

T

tags (HTML)
<APPLET>
atributo ALIGN, 185-186
atributo CODE, 183, 186-187
atributo CODEBASE, 186-187
atributo HSPACE, 186
atributo VSPACE, 186
exemplo, 182-183

, 185
<OBJECT>, 187-188
<PARAM>, 191-192
teclado
mnemônicos, 517-518
tratamento de eventos, 328-329, 540-541
exemplo de applet Keys, 331-334
método keyDown(), 329-330
método keyUp(), 329-330
teclas modificadoras, 334-335
teclas padrão, 330
teclas modificadoras, tratamento de eventos, 334-335
texto
convertendo para letras maiúsculas, 453-455
desenhando, 200
rótulos, 268-269
Thawte Certification, 426
threads
iniciando, 233
interrompendo, 233
definição, 231
programas de animação, 231-232
exemplo DigitalClock, 234-235
implementação, 232-234
tipo de dados
boolean, 42, 70
byte, 70
char, 70
double, 70
float, 70
int, 70
long, 70
short, 70
tipos de dados, 70-71,110
booleanos, 42
convertendo em objetos, 104-105
tipos primitivos, 70,110
convertendo em objetos, 104-105
tratamento de eventos, 529, 570-571
aplicativo SwingColorTest
classes, 543
listagem do código-fonte, 544-546
métodos, 543
receptores de evento, 543
applet ColorTest, 367
cliques de mouse, 541
cliques duplos, 322-323
exemplo de applet Spots, 319-322
método mouseDown(), 317-318
método mouseUp(), 318-319
eventos de ação, 535
eventos de ajuste, 536-538
eventos de foco, 538
eventos de item, 538-540
eventos de janela, 357-358, 542
eventos de tecla, 540-541
exemplo ChangeTitle.java, 533-534
JavaBeans, 485-487
menus, 360
métodos
actionPerformed(), 532
getSource(), 532
palavra-chave instanceof, 533
método handleEvent(), 316-317
movimentos de mouse, 541-342
exemplo de applet Lines, 324-328
método mouseDrag(), 323
método mouseEnter(), 324
método mouseExit(), 324
método mouseMove(), 323
receptores de evento
ActionListener, 535
AdjustmentListener, 536-538
associando componentes a, 531-532
definição, 529
FocusListener, 530, 538
importando, 531
ItemListener, 538-540
KeyListener, 540-541
MouseListener, 541
MouseMotionListener, 541-542
WindowListener, 542
teclado, 328-329
exemplo de applet Keys, 331-334
método keyDown(), 329
método keyUp(), 329-330
teclas modificadoras, 334-335
teclas padrão, 330
tipos de evento, 316
tremido (animação)
reduzindo
anulando o método update(), 238-240
uso de buffer duplo, 254-258

exemplo, 238
visão geral, 237

U-V

UNIX, configuração do JDK, 624
uso de buffer duplo
criando, 254-255
definição, 254
exemplo de applet Checkers, 255-258
limitações, 254
utilitário keytool, 427
utilitário policytool, 430
utilitários, *ver* **ferramentas**
variáveis
atribuição de nomes, 69
classe, 39, 62, 67
acessando valores, 95
alterando valores, 95
CLASSPATH, 3973
atribuindo, 67
controle de acesso, 378-384
acesso padrão, 379
acesso privativo, 379-380
acesso protegido, 381-382
acesso público, 381
declarando, 67
múltiplas variáveis, 68
decrementando, 80-81
definição, 67
escopo, 118
programa ScopeTest, **142**
estáticas, 384-386
globais, 67
incrementando, 80-81
instância, 39, 62, 67
locais, 67
overflow, 87
tipo de interface, 400
tipos de classe, 71
tipos de dados, 70-71
booleanos, 42
variáveis constantes
declarando, 137
definição, 137
variáveis de classe, 138
variáveis finais, 387
variáveis de classe, 39, 62, 67
valores
acessando, 95
alterando, 95
valores iniciais, 68
variáveis de instância, 39, 62, 67
valores
valores iniciais, 68
variáveis de objeto, *ver* **variáveis de instância**
variáveis estáticas, 384-386
variáveis finais, 387
variáveis globais, 67
variáveis locais, 67
declarando, 68
valores, 68
variável clickCount, 322
venda, 405
verificação de coerência (exceções), 413-414
VeriSign, 426
versões (dados), 493-494
versões de dados
criando, 493-494
definição, 493

W

Windows 95/Windows NT
aplicativos Java
compilando, 28-30
executando, 30
configuração do JDK, 23-24