1.12 Basic, Visual Basic, Visual C++, C# e .NET

A linguagem de programação **Basic** (Beginner's All-Purpose Symbolic Instruction Code) foi desenvolvida em meados da década de 1960, no Dartmouth College, como um meio de escrever programas simples. O principal propósito de Basic foi familiarizar os iniciantes com as técnicas de programação.

A linguagem **Visual Basic** da Microsoft foi introduzida no início de 1990 para simplificar o desenvolvimento de aplicativos Microsoft Windows e é uma das linguagens de programação mais populares do mundo.

As últimas ferramentas de desenvolvimento da Microsoft fazem parte de sua estratégia de escopo corporativo para integrar a Internet e a Web em aplicativos de computador. Essa estratégia é implementada na **plataforma .NET** da Microsoft, que fornece aos desenvolvedores as capacidades necessárias para criar e executar aplicativos de computador que possam executar em computadores distribuídos através da Internet. As três principais linguagens de programação da Microsoft são **Visual Basic** (baseada no Basic original), **Visual C++** (baseada no C++) e **C#** (baseada no C++ e no Java, e desenvolvida especificamente para a plataforma .NET). Os desenvolvedores que utilizam .NET podem escrever componentes de software na linguagem com que se sentem à vontade, então formar aplicativos combinando esses componentes com outros escritos em qualquer linguagem de .NET.

1.13 Ambiente típico de desenvolvimento Java

Agora explicamos os passos comumente seguidos na criação e execução de um aplicativo Java que utiliza um ambiente de desenvolvimento Java (ilustrado na Figura 1.1).

Em geral, programas Java passam por cinco fases — edição, compilação, carregamento, verificação e execução. Discutimos essas fases no contexto do Java SE Development Kit 6 (JDK6) da Sun Microsystems, Inc. Você pode baixar o JDK mais recente e sua documentação de `java.sun.com/javase/downloads/`. *Siga atentamente as instruções de instalação do JDK fornecidas* na *Seção* "Antes de você começar" *deste livro a fim de garantir a correta configuração do computador para compilar e executar programas Java.* Também é recomendável visitar o New to Java Center da Sun em:

```
java.sun.com/new2java/
```

[*Nota:* esse site da Web fornece instruções de instalação para Windows, Linux e Mac OS X. Se você não estiver utilizando um desses sistemas operacionais, consulte a documentação do ambiente Java do seu sistema ou pergunte a seu instrutor como realizar essas tarefas com base no sistema operacional do seu computador. Se tiver problemas com esse link ou com quaisquer outros referidos neste livro, verifique erratas e nos informe por e-mail em `deitel@deitel.com`.]

Fase 1: criando um programa

A Fase 1 consiste em editar um arquivo com um **programa editor** (em geral, conhecido simplesmente como um **editor**). Você digita um programa Java (em geral referido como **código-fonte**) utilizando o editor, faz quaisquer correções necessárias e salva o programa em um dispositivo de armazenamento secundário, como sua unidade de disco. Um nome de arquivo que termina com a **extensão `.java`** indica que o arquivo contém o código-fonte Java. Supomos que você saiba como editar um arquivo.

Dois editores amplamente utilizados em sistemas Linux são o `vi` e o `emacs`. No Windows, o Notepad será suficiente. Também há muitos editores freeware e shareware disponíveis online, incluindo EditPlus (`www.editplus.com`), TextPad (`www.textpad.com`) e jEdit (`www.jedit.org`).

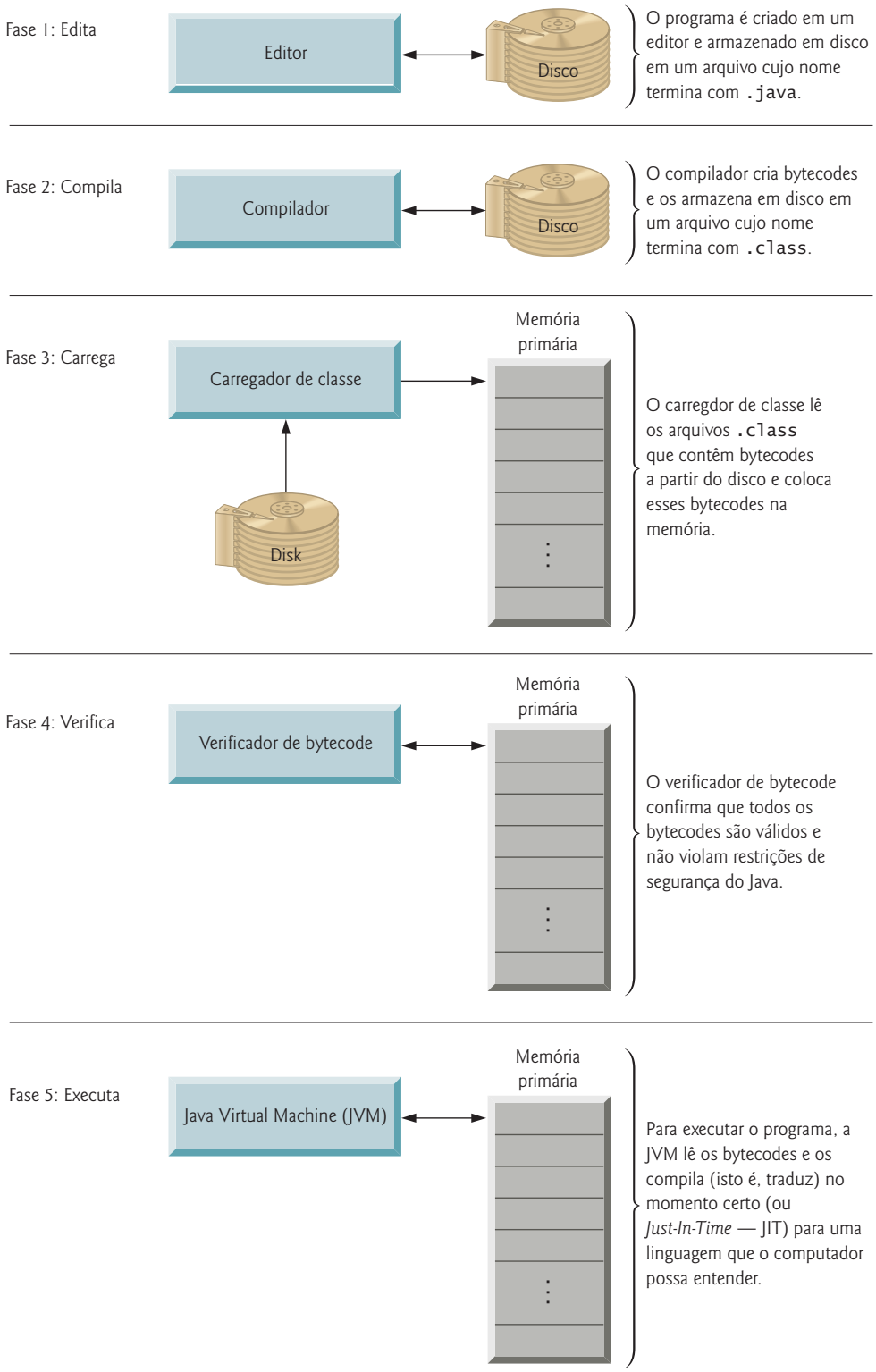

Figura 1.1 | Ambiente típico de desenvolvimento Java.

Para organizações que desenvolvem sistemas de informações substanciais, **ambientes de desenvolvimento integrado** (***Integrated Development Environments* — IDEs**) estão disponíveis a partir de muitos fornecedores de software importantes. IDEs fornecem ferramentas que suportam o processo de desenvolvimento de software, incluindo editores para escrever e editar programas e depuradores para localizar **erros de lógica** — erros que fazem os programas executar incorretamente. IDEs populares são Eclipse (`www.eclipse.org`), NetBeans (`www.netbeans.org`), JBuilder (`www.codegear.com`), JCreator (`www.jcreator.com`), BlueJ (`www.blueJ.org`) e jGRASP (`www.jgrasp.org`).

Fase 2: compilando um programa Java em bytecodes

Na Fase 2, utilize o comando (o **compilador Java**) para **compilar** um programa. Por exemplo, para compilar um programa chamado `Welcome.java`, você digitaria:

```
javac Welcome.java
```

na janela de comando do seu sistema (isto é, o **Prompt de Comando** no Windows, o **prompt de shell** no Linux ou **aplicativo Terminal** no Mac OS X). Se o programa compilar, o compilador produz um arquivo **.class** chamado `Welcome.class` que contém a versão compilada do programa.

O compilador Java converte o código-fonte Java em **bytecodes** que representam as tarefas a serem executadas na fase de execução (Fase 5). Os bytecodes são executados pela **Java Virtual Machine** (**JVM**) — uma parte do JDK e a base da plataforma Java. A **máquina virtual** (**Virtual Machine — VM**) é um aplicativo de software que simula um computador, mas oculta o sistema operacional e o hardware subjacentes dos programas que interagem com ela. Se a mesma VM for implementada nas várias plataformas de computador, os aplicativos que ela executa podem ser utilizados em todas essas plataformas. A JVM é uma das máquinas virtuais mais amplamente utilizadas. O .NET da Microsoft utiliza uma arquitetura de máquina virtual semelhante.

Ao contrário da linguagem de máquina, que é dependente do hardware específico de computador, os bytecodes são independentes de plataforma — eles não dependem de uma plataforma de hardware particular. Portanto, os bytecodes do Java são **portáveis** — sem recompilar o código-fonte, os mesmos bytecodes podem executar em qualquer plataforma contendo uma JVM que entende a versão do Java em que os bytecodes foram compilados. A JVM é invocada pelo comando `java`. Por exemplo, para executar um aplicativo Java chamado `Welcome`, você digitaria o comando:

```
java Welcome
```

em uma janela de comando para invocar a JVM, que então iniciaria os passos necessários para executar o aplicativo. Isso inicia a Fase 3.

Fase 3: carregando um programa na memória

Na Fase 3, a JVM armazena o programa na memória para executá-lo — isso é conhecido como **carregamento**. O **carregador de classe** da JVM pega os arquivos `.class` que contêm os bytecodes do programa e transfere-os para a memória primária. O carregador de classe também carrega qualquer arquivo `.class` fornecido pelo Java que seu programa utiliza. Os arquivos `.class` podem ser carregados a partir de um disco em seu sistema ou em uma rede (por exemplo, sua unidade local ou rede corporativa ou a Internet).

Fase 4: verificação de bytecode

Na Fase 4, enquanto as classes são carregadas, o **verificador de bytecode** examina seus bytecodes para assegurar que eles são válidos e não violam restrições de segurança do Java. O Java impõe uma forte segurança para certificar-se de que os programas Java que chegam pela rede não danificam os arquivos ou o sistema (como vírus e vermes de computador).

Fase 5: execução

Na Fase 5, a JVM executa os bytecodes do programa, realizando assim as ações especificadas pelo programa. Nas primeiras versões do Java, a JVM era simplesmente um interpretador para bytecodes Java. Isso fazia com que a maioria dos programas Java executasse lentamente porque a JVM interpretava e executava um bytecode por vez. Em geral, as JVMs atuais executam bytecodes utilizando uma combinação de interpretação e a chamada **compilação Just-In-Time (JIT)**. Nesse processo, a JVM analisa os bytecodes à medida que eles são interpretados, procurando **hot spots (pontos ativos)** — partes dos bytecodes que executam com frequência. Para essas partes, um **compilador Just-In-Time** (**JIT**) — conhecido como **compilador Java HotSpot** — traduz os bytecodes para a linguagem de máquina de um computador subjacente. Quando a JVM encontra novamente essas partes compiladas, o código de linguagem de máquina mais rápido é executado. Portanto, os programas Java na realidade passam por duas fases de compilação — uma em que código-fonte é traduzido em bytecodes (para a portabilidade entre JVMs em diferentes plataformas de computador) e uma segunda em que, durante a execução, os bytecodes são traduzidos em linguagem de máquina para o computador real em que o programa é executado.

Problemas que podem ocorrer em tempo de execução

Os programas podem não funcionar na primeira tentativa. Cada uma das fases anteriores pode falhar por causa de vários erros que discutiremos por todo este livro. Por exemplo, um programa executável talvez tente realizar uma operação de divisão por zero (uma operação ilegal para a aritmética de número inteiro em Java). Isso faria o programa Java imprimir uma mensagem de erro. Se isso ocorresse, você teria de retornar à fase de edição, fazer as correções necessárias e passar novamente pelas demais fases para determinar se as correções resolveram o(s) problema(s). [*Nota:* a maioria dos programas Java realiza entrada ou saída de dados. Quando dizemos que um programa exibe uma mensagem, normalmente queremos dizer que ele exibe essa mensagem na tela do computador. As mensagens e outros dados podem ser enviados para outros dispositivos de saída, como discos e impressoras ou até mesmo uma rede para transmissão para outros computadores.]

Erro comum de programação 1.1

Erros como divisão por zero ocorrem durante a execução de um programa, por isso são chamados de ***erros de runtime*** *ou* ***erros de tempo de execução****.* ***Erros de tempo de execução fatais*** *fazem com que os programas sejam imediatamente encerrados sem terem realizado seu trabalho com sucesso.* ***Erros de tempo de execução não fatais*** *permitem que os programas executem até sua conclusão, produzindo frequentemente resultados incorretos.*

1.14 Notas sobre o Java e este livro

O Java é uma poderosa linguagem de programação. Os programadores experientes às vezes sentem-se orgulhosos por criarem algum uso exótico, distorcido e complexo de uma linguagem. Essa é uma prática de programação pobre. Ela torna programas mais difíceis de ler, mais propensos a se comportar estranhamente, mais difíceis de testar e depurar e mais difíceis de adaptar em caso de alteração de requisitos. Este livro enfatiza a *clareza.* Eis a seguir a nossa primeira dica de Boa prática de programação.

Boa prática de programação 1.1

Escreva seus programas Java de uma maneira simples e direta. Isso é às vezes chamado KIS (“Keep It Simple”, ou seja, “mantenha a coisa simples”). Não “estenda” a linguagem tentando usos bizarros.

Você viu antes que o Java é uma linguagem portável e que programas escritos em Java podem executar em muitos computadores diferentes. Em geral, a *portabilidade é um objetivo ilusório.*

Dica de portabilidade 1.2

Embora seja mais fácil escrever programas portáveis em Java do que na maioria das outras linguagens de programação, diferenças entre compiladores, JVMs e computadores podem tornar a portabilidade difícil de alcançar. Simplesmente escrever programas em Java não garante portabilidade.

Dica de prevenção de erro 1.1

Sempre teste os programas Java em todos os sistemas em que você quiser executá-los, para assegurar que funcionarão corretamente para o público-alvo.

Você pode encontrar uma versão baseada na Web da documentação da Java API em `java.sun.com/javase/6/docs/api/index.html`, ou fazer o download dessa documentação no seu computador em `java.sun.com/javase/downloads/`. Para detalhes técnicos adicionais sobre muitos aspectos do desenvolvimento Java, visite `java.sun.com/reference/docs/index.html`.

Boa prática de programação 1.2

Leia a documentação da versão do Java que você está utilizando. Consulte-a frequentemente para certificar-se de que você está ciente da rica coleção de recursos Java e de que está utilizando esses recursos corretamente.

[1] Em `www.deitel.com/books/jhtp8/` fornecemos uma versão do Linux para esse test-drive. Também fornecemos links para vídeos que o ajudam a começar a trabalhar com vários ambientes de desenvolvimento integrados populares (IDEs), incluindo o Eclipse e NetBeans.