

### 1.1 Introduction



*The central theme of this book is to learn how to solve problems by writing a program.*

what is programming?  
programming  
program

This book is about programming. So, what is programming? The term *programming* means to create (or develop) software, which is also called a *program*. In basic terms, software contains the instructions that tell a computer—or a computerized device—what to do.

Software is all around you, even in devices that you might not think would need it. Of course, you expect to find and use software on a personal computer, but software also plays a role in running airplanes, cars, cell phones, and even toasters. On a personal computer, you use word processors to write documents, Web browsers to explore the Internet, and e-mail programs to send and receive messages. These programs are all examples of software. Software developers create software with the help of powerful tools called *programming languages*.

This book teaches you how to create programs by using the Java programming language. There are many programming languages, some of which are decades old. Each language was invented for a specific purpose—to build on the strengths of a previous language, for example, or to give the programmer a new and unique set of tools. Knowing that there are so many programming languages available, it would be natural for you to wonder which one is best. But, in truth, there is no “best” language. Each one has its own strengths and weaknesses. Experienced programmers know that one language might work well in some situations, whereas a different language may be more appropriate in other situations. For this reason, seasoned programmers try to master as many different programming languages as they can, giving them access to a vast arsenal of software-development tools.

If you learn to program using one language, you should find it easy to pick up other languages. The key is to learn how to solve problems using a programming approach. That is the main theme of this book.

You are about to begin an exciting journey: learning how to program. At the outset, it is helpful to review computer basics, programs, and operating systems. If you are already familiar with such terms as CPU, memory, disks, operating systems, and programming languages, you may skip Sections 1.2–1.4.

### 1.2 What Is a Computer?



*A computer is an electronic device that stores and processes data.*

hardware  
software

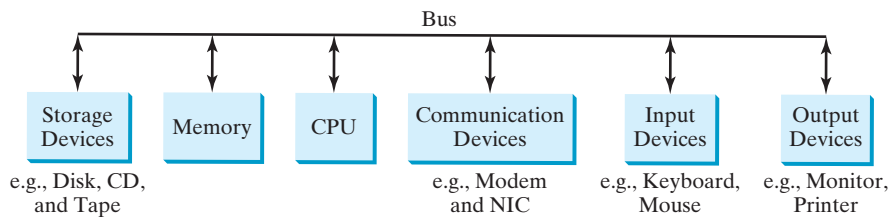
A computer includes both *hardware* and *software*. In general, hardware comprises the visible, physical elements of the computer, and software provides the invisible instructions that control the hardware and make it perform specific tasks. Knowing computer hardware isn’t essential to learning a programming language, but it can help you better understand the effects that a program’s instructions have on the computer and its components. This section introduces computer hardware components and their functions.

A computer consists of the following major hardware components (Figure 1.1):

- A central processing unit (CPU)
- Memory (main memory)
- Storage devices (such as disks and CDs)
- Input devices (such as the mouse and keyboard)
- Output devices (such as monitors and printers)
- Communication devices (such as modems and network interface cards)

bus

A computer’s components are interconnected by a subsystem called a *bus*. You can think of a bus as a sort of system of roads running among the computer’s components; data and power travel along the bus from one part of the computer to another. In personal computers,



**FIGURE 1.1** A computer consists of a CPU, memory, storage devices, input devices, output devices, and communication devices.

the bus is built into the computer's *motherboard*, which is a circuit case that connects all of the parts of a computer together. motherboard

### 1.2.1 Central Processing Unit

The *central processing unit (CPU)* is the computer's brain. It retrieves instructions from memory and executes them. The CPU usually has two components: a *control unit* and an *arithmetic/logic unit*. The control unit controls and coordinates the actions of the other components. The arithmetic/logic unit performs numeric operations (addition, subtraction, multiplication, division) and logical operations (comparisons). CPU

Today's CPUs are built on small silicon semiconductor chips that contain millions of tiny electric switches, called *transistors*, for processing information.

Every computer has an internal clock, which emits electronic pulses at a constant rate. These pulses are used to control and synchronize the pace of operations. A higher clock *speed* enables more instructions to be executed in a given period of time. The unit of measurement of clock speed is the *hertz (Hz)*, with 1 hertz equaling 1 pulse per second. In the 1990s, computers measured clocked speed in *megahertz (MHz)*, but CPU speed has been improving continuously; the clock speed of a computer is now usually stated in *gigahertz (GHz)*. Intel's newest processors run at about 3 GHz. speed  
hertz  
megahertz  
gigahertz

CPUs were originally developed with only one core. The *core* is the part of the processor that performs the reading and executing of instructions. In order to increase CPU processing power, chip manufacturers are now producing CPUs that contain multiple cores. A multicore CPU is a single component with two or more independent cores. Today's consumer computers typically have two, three, and even four separate cores. Soon, CPUs with dozens or even hundreds of cores will be affordable. core

### 1.2.2 Bits and Bytes

Before we discuss memory, let's look at how information (data and programs) are stored in a computer.

A computer is really nothing more than a series of switches. Each switch exists in two states: on or off. Storing information in a computer is simply a matter of setting a sequence of switches on or off. If the switch is on, its value is 1. If the switch is off, its value is 0. These 0s and 1s are interpreted as digits in the binary number system and are called *bits* (binary digits). bits

The minimum storage unit in a computer is a *byte*. A byte is composed of eight bits. A small number such as 3 can be stored as a single byte. To store a number that cannot fit into a single byte, the computer uses several bytes. byte

Data of various kinds, such as numbers and characters, are encoded as a series of bytes. As a programmer, you don't need to worry about the encoding and decoding of data, which the computer system performs automatically, based on the encoding scheme. An *encoding scheme* is a set of rules that govern how a computer translates characters, numbers, and symbols into data the computer can actually work with. Most schemes translate each character encoding scheme

into a predetermined string of bits. In the popular ASCII encoding scheme, for example, the character **C** is represented as **01000011** in one byte.

A computer’s storage capacity is measured in bytes and multiples of the byte, as follows:

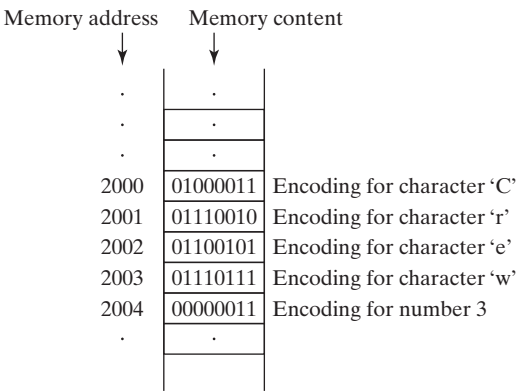
- A *kilobyte (KB)* is about 1,000 bytes.
- A *megabyte (MB)* is about 1 million bytes.
- A *gigabyte (GB)* is about 1 billion bytes.
- A *terabyte (TB)* is about 1 trillion bytes.

A typical one-page word document might take 20 KB. Therefore, 1 MB can store 50 pages of documents and 1 GB can store 50,000 pages of documents. A typical two-hour high-resolution movie might take 8 GB, so it would require 160 GB to store 20 movies.

1.2.3 Memory

A computer’s *memory* consists of an ordered sequence of bytes for storing programs as well as data that the program is working with. You can think of memory as the computer’s work area for executing a program. A program and its data must be moved into the computer’s memory before they can be executed by the CPU.

Every byte in the memory has a *unique address*, as shown in Figure 1.2. The address is used to locate the byte for storing and retrieving the data. Since the bytes in the memory can be accessed in any order, the memory is also referred to as *random-access memory (RAM)*.



**FIGURE 1.2** Memory stores data and program instructions in uniquely addressed memory locations.

Today’s personal computers usually have at least 4 gigabyte of RAM, but they more commonly have 6 to 8 GB installed. Generally speaking, the more RAM a computer has, the faster it can operate, but there are limits to this simple rule of thumb.

A memory byte is never empty, but its initial content may be meaningless to your program. The current content of a memory byte is lost whenever new information is placed in it.

Like the CPU, memory is built on silicon semiconductor chips that have millions of transistors embedded on their surface. Compared to CPU chips, memory chips are less complicated, slower, and less expensive.

1.2.4 Storage Devices

A computer’s memory (RAM) is a volatile form of data storage: any information that has been stored in memory (i.e., saved) is lost when the system’s power is turned off. Programs and data are permanently stored on *storage devices* and are moved, when the computer

storage devices

actually uses them, to memory, which operates at much faster speeds than permanent storage devices can.

There are three main types of storage devices:

- Magnetic disk drives
- Optical disc drives (CD and DVD)
- USB flash drives

*Drives* are devices for operating a medium, such as disks and CDs. A storage medium drive physically stores data and program instructions. The drive reads data from the medium and writes data onto the medium.

## Disks

A computer usually has at least one hard disk drive. *Hard disks* are used for permanently storing data and programs. Newer computers have hard disks that can store from 500 gigabytes to 1 terabytes of data. Hard disk drives are usually encased inside the computer, but removable hard disks are also available.

## CDs and DVDs

*CD* stands for compact disc. There are two types of CD drives: CD-R and CD-RW. A *CD-R* is for read-only permanent storage; the user cannot modify its contents once they are recorded. A *CD-RW* can be used like a hard disk; that is, you can write data onto the disc, and then overwrite that data with new data. A single CD can hold up to 700 MB. Most new PCs are equipped with a CD-RW drive that can work with both CD-R and CD-RW discs.

*DVD* stands for digital versatile disc or digital video disc. DVDs and CDs look alike, and you can use either to store data. A DVD can hold more information than a CD; a standard DVD's storage capacity is 4.7 GB. Like CDs, there are two types of DVDs: DVD-R (read-only) and DVD-RW (rewritable).

## USB Flash Drives

*Universal serial bus (USB)* connectors allow the user to attach many kinds of peripheral devices to the computer. You can use a USB to connect a printer, digital camera, mouse, external hard disk drive, and other devices to the computer.

A *USB flash drive* is a device for storing and transporting data. A flash drive is small—about the size of a pack of gum. It acts like a portable hard drive that can be plugged into your computer's USB port. USB flash drives are currently available with up to 256 GB storage capacity.

## 1.2.5 Input and Output Devices

Input and output devices let the user communicate with the computer. The most common input devices are *keyboards* and *mice*. The most common output devices are *monitors* and *printers*.

### The Keyboard

A keyboard is a device for entering input. Compact keyboards are available without a numeric keypad.

*Function keys* are located across the top of the keyboard and are prefaced with the letter *F*. Their functions depend on the software currently being used.

A *modifier key* is a special key (such as the *Shift*, *Alt*, and *Ctrl* keys) that modifies the normal action of another key when the two are pressed simultaneously.

The *numeric keypad*, located on the right side of most keyboards, is a separate set of keys styled like a calculator to use for entering numbers quickly.

*Arrow keys*, located between the main keypad and the numeric keypad, are used to move the mouse pointer up, down, left, and right on the screen in many kinds of programs.

## 6 Chapter 1 Introduction to Computers, Programs, and Java

Insert key  
Delete key  
Page Up key  
Page Down key

The *Insert*, *Delete*, *Page Up*, and *Page Down* keys are used in word processing and other programs for inserting text and objects, deleting text and objects, and moving up or down through a document one screen at a time.

### The Mouse

A *mouse* is a pointing device. It is used to move a graphical pointer (usually in the shape of an arrow) called a *cursor* around the screen or to click on-screen objects (such as a button) to trigger them to perform an action.

### The Monitor

The *monitor* displays information (text and graphics). The screen resolution and dot pitch determine the quality of the display.

screen resolution  
pixels

The *screen resolution* specifies the number of pixels in horizontal and vertical dimensions of the display device. *Pixels* (short for “picture elements”) are tiny dots that form an image on the screen. A common resolution for a 17-inch screen, for example, is 1,024 pixels wide and 768 pixels high. The resolution can be set manually. The higher the resolution, the sharper and clearer the image is.

dot pitch

The *dot pitch* is the amount of space between pixels, measured in millimeters. The smaller the dot pitch, the sharper the display.

## 1.2.6 Communication Devices

Computers can be networked through communication devices, such as a dial-up modem (*modulator/demodulator*), a DSL or cable modem, a wired network interface card, or a wireless adapter.

dial-up modem

- A *dial-up modem* uses a phone line and can transfer data at a speed up to 56,000 bps (bits per second).

digital subscriber line (DSL)

- A *digital subscriber line (DSL)* connection also uses a standard phone line, but it can transfer data 20 times faster than a standard dial-up modem.

cable modem

- A *cable modem* uses the cable TV line maintained by the cable company and is generally faster than DSL.

network interface card (NIC)

local area network (LAN)

million bits per second  
(mbps)

- A *network interface card (NIC)* is a device that connects a computer to a *local area network (LAN)*. LANs are commonly used in universities, businesses, and government agencies. A high-speed NIC called *1000BaseT* can transfer data at 1,000 million bits per second (mbps).

- Wireless networking is now extremely popular in homes, businesses, and schools. Every laptop computer sold today is equipped with a wireless adapter that enables the computer to connect to a local area network and the Internet.



### Note

Answers to checkpoint questions are on the Companion Website.



- 1.1 What are hardware and software?
- 1.2 List five major hardware components of a computer.
- 1.3 What does the acronym “CPU” stand for?
- 1.4 What unit is used to measure CPU speed?
- 1.5 What is a bit? What is a byte?
- 1.6 What is memory for? What does RAM stand for? Why is memory called RAM?
- 1.7 What unit is used to measure memory size?

- 1.8** What unit is used to measure disk size?
- 1.9** What is the primary difference between memory and a storage device?

## 1.3 Programming Languages

*Computer programs, known as software, are instructions that tell a computer what to do.*



Computers do not understand human languages, so programs must be written in a language a computer can use. There are hundreds of programming languages, and they were developed to make the programming process easier for people. However, all programs must be converted into the instructions the computer can execute.

### 1.3.1 Machine Language

A computer's native language, which differs among different types of computers, is its *machine language*—a set of built-in primitive instructions. These instructions are in the form of binary code, so if you want to give a computer an instruction in its native language, you have to enter the instruction as binary code. For example, to add two numbers, you might have to write an instruction in binary code, like this:

machine language

**1101101010011010**

### 1.3.2 Assembly Language

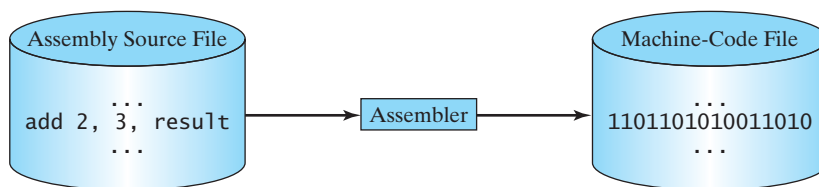
Programming in machine language is a tedious process. Moreover, programs written in machine language are very difficult to read and modify. For this reason, *assembly language* was created in the early days of computing as an alternative to machine languages. Assembly language uses a short descriptive word, known as a *mnemonic*, to represent each of the machine-language instructions. For example, the mnemonic **add** typically means to add numbers and **sub** means to subtract numbers. To add the numbers **2** and **3** and get the result, you might write an instruction in assembly code like this:

assembly language

**add 2, 3, result**

Assembly languages were developed to make programming easier. However, because the computer cannot execute assembly language, another program—called an *assembler*—is used to translate assembly-language programs into machine code, as shown in Figure 1.3.

assembler



**FIGURE 1.3** An assembler translates assembly-language instructions into machine code.

Writing code in assembly language is easier than in machine language. However, it is still tedious to write code in assembly language. An instruction in assembly language essentially corresponds to an instruction in machine code. Writing in assembly requires that you know how the CPU works. Assembly language is referred to as a *low-level language*, because assembly language is close in nature to machine language and is machine dependent.

low-level language



1.3.3 High-Level Language

high-level language      In the 1950s, a new generation of programming languages known as *high-level languages* emerged. They are platform independent, which means that you can write a program in a high-level language and run it in different types of machines. High-level languages are English-like and easy to learn and use. The instructions in a high-level programming language are called *statements*. Here, for example, is a high-level language statement that computes the area of a circle with a radius of 5:

statement

```
area = 5 * 5 * 3.14159;
```

There are many high-level programming languages, and each was designed for a specific purpose. Table 1.1 lists some popular ones.

TABLE 1.1 Popular High-Level Programming Languages

Language	Description
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner’s All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	C++ is an object-oriented language, based on C.
C#	Pronounced “C Sharp.” It is a hybrid of Java and C++ and was developed by Microsoft.
COBOL	COmmon Business Oriented Language. Used for business applications.
FORTRAN	FORmula TRANslation. Popular for scientific and mathematical applications.
Java	Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications.
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming.
Python	A simple general-purpose scripting language good for writing short programs.
Visual Basic	Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop graphical user interfaces.

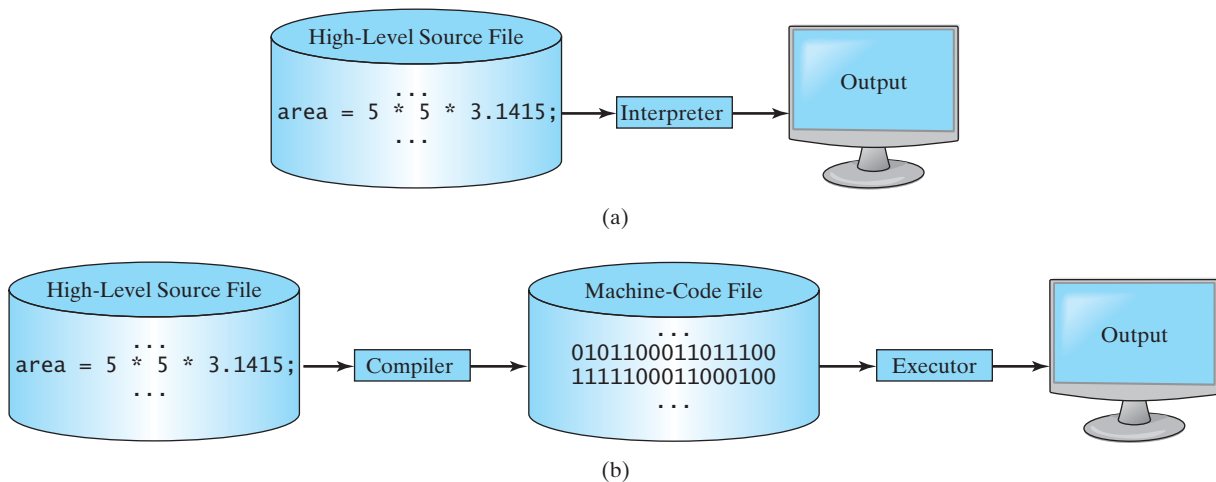
source program      A program written in a high-level language is called a *source program* or *source code*.  
source code      Because a computer cannot execute a source program, a source program must be translated  
interpreter      into machine code for execution. The translation can be done using another programming tool  
compiler      called an *interpreter* or a *compiler*.

- An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away, as shown in Figure 1.4a. Note that a statement from the source code may be translated into several machine instructions.
- A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in Figure 1.4b.



- 1.10 What language does the CPU understand?
- 1.11 What is an assembly language?
- 1.12 What is an assembler?
- 1.13 What is a high-level programming language?
- 1.14 What is a source program?

- I.15** What is an interpreter?  
**I.16** What is a compiler?  
**I.17** What is the difference between an interpreted language and a compiled language?



**FIGURE 1.4** (a) An interpreter translates and executes a program one statement at a time. (b) A compiler translates the entire source program into a machine-language file for execution.

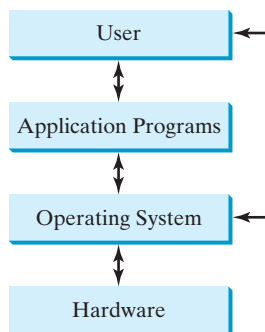
## I.4 Operating Systems

*The operating system (OS) is the most important program that runs on a computer. The OS manages and controls a computer's activities.*



operating system (OS)

The popular *operating systems* for general-purpose computers are Microsoft Windows, Mac OS, and Linux. Application programs, such as a Web browser or a word processor, cannot run unless an operating system is installed and running on the computer. Figure 1.5 shows the interrelationship of hardware, operating system, application software, and the user.



**FIGURE 1.5** Users and applications access the computer's hardware via the operating system.

The major tasks of an operating system are as follows:

- Controlling and monitoring system activities
- Allocating and assigning system resources
- Scheduling operations



### 1.4.1 Controlling and Monitoring System Activities

Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the monitor, keeping track of files and folders on storage devices, and controlling peripheral devices, such as disk drives and printers. An operating system must also ensure that different programs and users working at the same time do not interfere with each other. In addition, the OS is responsible for security, ensuring that unauthorized users and programs are not allowed to access the system.

### 1.4.2 Allocating and Assigning System Resources

The operating system is responsible for determining what computer resources a program needs (such as CPU time, memory space, disks, input and output devices) and for allocating and assigning them to run the program.

### 1.4.3 Scheduling Operations

The OS is responsible for scheduling programs' activities to make efficient use of system resources. Many of today's operating systems support techniques such as *multiprogramming*, *multithreading*, and *multiprocessing* to increase system performance.

multiprogramming

*Multiprogramming* allows multiple programs to run simultaneously by sharing the same CPU. The CPU is much faster than the computer's other components. As a result, it is idle most of the time—for example, while waiting for data to be transferred from a disk or waiting for other system resources to respond. A multiprogramming OS takes advantage of this situation by allowing multiple programs to use the CPU when it would otherwise be idle. For example, multiprogramming enables you to use a word processor to edit a file at the same time as your Web browser is downloading a file.

multithreading

*Multithreading* allows a single program to execute multiple tasks at the same time. For instance, a word-processing program allows users to simultaneously edit text and save it to a disk. In this example, editing and saving are two tasks within the same application. These two tasks may run concurrently.

multiprocessing

*Multiprocessing*, or *parallel processing*, uses two or more processors together to perform subtasks concurrently and then combine solutions of the subtasks to obtain a solution for the entire task. It is like a surgical operation where several doctors work together on one patient.



**1.18** What is an operating system? List some popular operating systems.

**1.19** What are the major responsibilities of an operating system?

**1.20** What are multiprogramming, multithreading, and multiprocessing?

## 1.5 Java, the World Wide Web, and Beyond



*Java is a powerful and versatile programming language for developing software running on mobile devices, desktop computers, and servers.*

This book introduces Java programming. Java was developed by a team led by James Gosling at Sun Microsystems. Sun Microsystems was purchased by Oracle in 2010. Originally called *Oak*, Java was designed in 1991 for use in embedded chips in consumer electronic appliances. In 1995, renamed *Java*, it was redesigned for developing Web applications. For the history of Java, see [www.java.com/en/javahistory/index.jsp](http://www.java.com/en/javahistory/index.jsp).

Java has become enormously popular. Its rapid rise and wide acceptance can be traced to its design characteristics, particularly its promise that you can write a program once and run it anywhere. As stated by its designer, Java is *simple*, *object oriented*, *distributed*,

*interpreted, robust, secure, architecture neutral, portable, high performance, multi-threaded, and dynamic.* For the anatomy of Java characteristics, see [www.cs.armstrong.edu/liang/JavaCharacteristics.pdf](http://www.cs.armstrong.edu/liang/JavaCharacteristics.pdf).

Java is a full-featured, general-purpose programming language that can be used to develop robust mission-critical applications. Today, it is employed not only for Web programming but also for developing standalone applications across platforms on servers, desktop computers, and mobile devices. It was used to develop the code to communicate with and control the robotic rover on Mars. Many companies that once considered Java to be more hype than substance are now using it to create distributed applications accessed by customers and partners across the Internet. For every new project being developed today, companies are asking how they can use Java to make their work easier.

The World Wide Web is an electronic information repository that can be accessed on the Internet from anywhere in the world. The Internet, the Web's infrastructure, has been around for more than forty years. The colorful World Wide Web and sophisticated Web browsers are the major reason for the Internet's popularity.

Java initially became attractive because Java programs can be run from a Web browser. Such programs are called *applets*. Applets employ a modern graphical interface with buttons, text fields, text areas, radio buttons, and so on, to interact with users on the Web and process their requests. Applets make the Web responsive, interactive, and fun to use. Applets are embedded in an HTML file. *HTML (Hypertext Markup Language)* is a simple scripting language for laying out documents, linking documents on the Internet, and bringing images, sound, and video alive on the Web. Today, you can use Java to develop rich Internet applications. A rich Internet application (RIA) is a Web application designed to deliver the same features and functions normally associated with desktop applications.

Java is now very popular for developing applications on Web servers. These applications process data, perform computations, and generate dynamic Web pages. Many commercial Websites are developed using Java on the backend.

Java is a versatile programming language: you can use it to develop applications for desktop computers, servers, and small handheld devices. The software for Android cell phones is developed using Java.

- 1.21** Who invented Java? Which company owns Java now?
- 1.22** What is a Java applet?
- 1.23** What programming language does Android use?



## 1.6 The Java Language Specification, API, JDK, and IDE

*Java syntax is defined in the Java language specification, and the Java library is defined in the Java API. The JDK is the software for developing and running Java programs. An IDE is an integrated development environment for rapidly developing programs.*



Computer languages have strict rules of usage. If you do not follow the rules when writing a program, the computer will not be able to understand it. The Java language specification and the Java API define the Java standards.

The *Java language specification* is a technical definition of the Java programming language's syntax and semantics. You can find the complete Java language specification at <http://docs.oracle.com/javase/specs/>.

The *application program interface (API)*, also known as *library*, contains predefined classes and interfaces for developing Java programs. The API is still expanding. You can view and download the latest version of the Java API at <http://download.java.net/jdk8/docs/api/>.

Java language specification

API  
library

Java is a full-fledged and powerful language that can be used in many ways. It comes in three editions:

Java SE, EE, and ME

- *Java Standard Edition (Java SE)* to develop client-side applications. The applications can run standalone or as applets running from a Web browser.
- *Java Enterprise Edition (Java EE)* to develop server-side applications, such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
- *Java Micro Edition (Java ME)* to develop applications for mobile devices, such as cell phones.

Java Development  
Toolkit (JDK)  
JDK 1.8 = JDK 8

This book uses Java SE to introduce Java programming. Java SE is the foundation upon which all other Java technology is based. There are many versions of Java SE. The latest, Java SE 8, is used in this book. Oracle releases each version with a *Java Development Toolkit (JDK)*. For Java SE 8, the Java Development Toolkit is called *JDK 1.8* (also known as *Java 8* or *JDK 8*).

Integrated development  
environment

The JDK consists of a set of separate programs, each invoked from a command line, for developing and testing Java programs. Instead of using the JDK, you can use a Java development tool (e.g., NetBeans, Eclipse, and TextPad)—software that provides an *integrated development environment (IDE)* for developing Java programs quickly. Editing, compiling, building, debugging, and online help are integrated in one graphical user interface. You simply enter source code in one window or open an existing file in a window, and then click a button or menu item or press a function key to compile and run the program.



- 1.24** What is the Java language specification?
- 1.25** What does JDK stand for?
- 1.26** What does IDE stand for?
- 1.27** Are tools like NetBeans and Eclipse different languages from Java, or are they dialects or extensions of Java?

## 1.7 A Simple Java Program



*A Java program is executed from the **main** method in the class.*

what is a console?  
console input  
console output

Let's begin with a simple Java program that displays the message **Welcome to Java!** on the console. (The word *console* is an old computer term that refers to the text entry and display device of a computer. *Console input* means to receive input from the keyboard, and *console output* means to display output on the monitor.) The program is shown in Listing 1.1.

### LISTING 1.1 Welcome.java

class  
main method  
display message



**VideoNote**  
Your first Java program

```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         // Display message Welcome to Java! on the console  
4         System.out.println("Welcome to Java!");  
5     }  
6 }
```



Welcome to Java!

line numbers

Note that the line numbers are for reference purposes only; they are not part of the program. So, don't type line numbers in your program.

Line 1 defines a class. Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is `Welcome`.

Line 2 defines the `main` method. The program is executed from the `main` method. A class may contain several methods. The `main` method is the entry point where the program begins execution.

A method is a construct that contains statements. The `main` method in this program contains the `System.out.println` statement. This statement displays the string `Welcome to Java!` on the console (line 4). *String* is a programming term meaning a sequence of characters. A string must be enclosed in double quotation marks. Every statement in Java ends with a semicolon (`;`), known as the *statement terminator*.

*Reserved words*, or *keywords*, have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word `class`, it understands that the word after `class` is the name for the class. Other reserved words in this program are `public`, `static`, and `void`.

Line 3 is a *comment* that documents what the program is and how it is constructed. Comments help programmers to communicate and understand the program. They are not programming statements and thus are ignored by the compiler. In Java, comments are preceded by two slashes (`//`) on a line, called a *line comment*, or enclosed between `/*` and `*/` on one or several lines, called a *block comment* or *paragraph comment*. When the compiler sees `//`, it ignores all text after `//` on the same line. When it sees `/*`, it scans for the next `*/` and ignores any text between `/*` and `*/`. Here are examples of comments:

```
// This application program displays Welcome to Java!
/* This application program displays Welcome to Java! */
/* This application program
   displays Welcome to Java! */
```

A pair of curly braces in a program forms a *block* that groups the program's components. In Java, each block begins with an opening brace (`{`) and ends with a closing brace (`}`). Every class has a *class block* that groups the data and methods of the class. Similarly, every method has a *method block* that groups the statements in the method. Blocks can be *nested*, meaning that one block can be placed within another, as shown in the following code.

```
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Diagram labels:   
 - `public class Welcome {` to `}`: Class block   
 - `public static void main(String[] args) {` to `}`: Method block



**Tip**

An opening brace must be matched by a closing brace. Whenever you type an opening brace, immediately type a closing brace to prevent the missing-brace error. Most Java IDEs automatically insert the closing brace for each opening brace.



**Caution**

Java source programs are case sensitive. It would be wrong, for example, to replace `main` in the program with `Main`.

You have seen several special characters (e.g., `{`, `}`, `//`, `;`) in the program. They are used in almost every program. Table 1.2 summarizes their uses.

The most common errors you will make as you learn to program will be syntax errors. Like any programming language, Java has its own syntax, and you need to write code that

TABLE 1.2 Special Characters

Character	Name	Description
{ }	Opening and closing braces	Denote a block to enclose statements.
( )	Opening and closing parentheses	Used with methods.
[ ]	Opening and closing brackets	Denote an array.
//	Double slashes	Precede a comment line.
" "	Opening and closing quotation marks	Enclose a string (i.e., sequence of characters).
;	Semicolon	Mark the end of a statement.

syntax rules

conforms to the *syntax rules*. If your program violates a rule—for example, if the semicolon is missing, a brace is missing, a quotation mark is missing, or a word is misspelled—the Java compiler will report syntax errors. Try to compile the program with these errors and see what the compiler reports.



Note

You are probably wondering why the `main` method is defined this way and why `System.out.println(...)` is used to display a message on the console. For the time being, simply accept that this is how things are done. Your questions will be fully answered in subsequent chapters.

The program in Listing 1.1 displays one message. Once you understand the program, it is easy to extend it to display more messages. For example, you can rewrite the program to display three messages, as shown in Listing 1.2.

LISTING 1.2 `WelcomeWithThreeMessages.java`

class  
main method  
display message

```
1 public class WelcomeWithThreeMessages {
2     public static void main(String[] args) {
3         System.out.println("Programming is fun!");
4         System.out.println("Fundamentals First");
5         System.out.println("Problem Driven");
6     }
7 }
```



Programming is fun!  
Fundamentals First  
Problem Driven

Further, you can perform mathematical computations and display the result on the console.

Listing 1.3 gives an example of evaluating  $\frac{10.5 + 2 \times 3}{45 - 3.5}$ .

LISTING 1.3 `ComputeExpression.java`

class  
main method  
compute expression

```
1 public class ComputeExpression {
2     public static void main(String[] args) {
3         System.out.println((10.5 + 2 * 3) / (45 - 3.5));
4     }
5 }
```



0.39759036144578314

The multiplication operator in Java is `*`. As you can see, it is a straightforward process to translate an arithmetic expression to a Java expression. We will discuss Java expressions further in Chapter 2.

- 1.28** What is a keyword? List some Java keywords.
- 1.29** Is Java case sensitive? What is the case for Java keywords?
- 1.30** What is a comment? Is the comment ignored by the compiler? How do you denote a comment line and a comment paragraph?
- 1.31** What is the statement to display a string on the console?
- 1.32** Show the output of the following code:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("3.5 * 4 / 2 - 2.5 is ");
        System.out.println(3.5 * 4 / 2 - 2.5);
    }
}
```



## 1.8 Creating, Compiling, and Executing a Java Program

*You save a Java program in a .java file and compile it into a .class file. The .class file is executed by the Java Virtual Machine.*



You have to create your program and compile it before it can be executed. This process is repetitive, as shown in Figure 1.6. If your program has compile errors, you have to modify the program to fix them, and then recompile it. If your program has runtime errors or does not produce the correct result, you have to modify the program, recompile it, and execute it again.

You can use any text editor or IDE to create and edit a Java source-code file. This section demonstrates how to create, compile, and run Java programs from a command window. Sections 1.10 and 1.11 will introduce developing Java programs using NetBeans and Eclipse. From the command window, you can use a text editor such as Notepad to create the Java source-code file, as shown in Figure 1.7.

command window



### Note

The source file must end with the extension **.java** and must have the same exact name as the public class name. For example, the file for the source code in Listing 1.1 should be named **Welcome.java**, since the public class name is **Welcome**.

file name Welcome.java,

A Java compiler translates a Java source file into a Java bytecode file. The following command compiles **Welcome.java**:

compile

```
javac Welcome.java
```



### Note

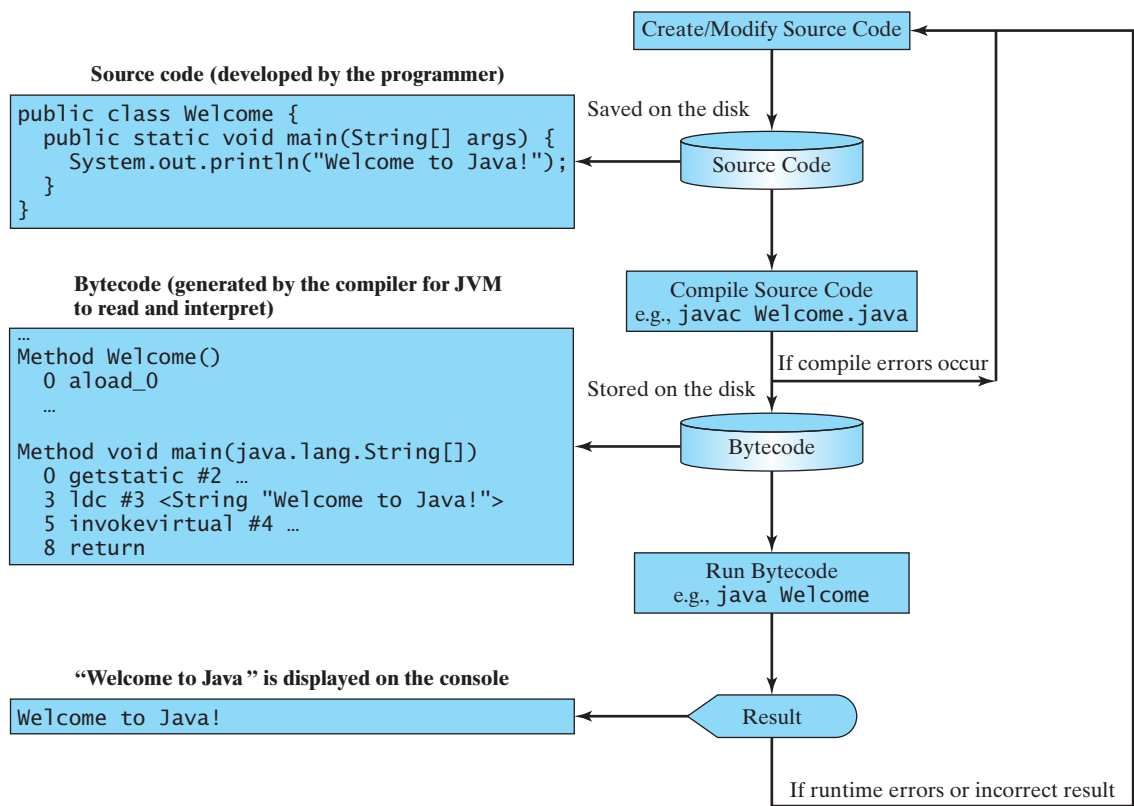
You must first install and configure the JDK before you can compile and run programs. See Supplement I.B, Installing and Configuring JDK 8, for how to install the JDK and set up the environment to compile and run Java programs. If you have trouble compiling and running programs, see Supplement I.C, Compiling and Running Java from the Command Window. This supplement also explains how to use basic DOS commands and how to use Windows Notepad to create and edit files. All the supplements are accessible from the Companion Website at [www.cs.armstrong.edu/liang/intro10e/supplement.html](http://www.cs.armstrong.edu/liang/intro10e/supplement.html).

Supplement I.B

Supplement I.C

If there aren't any syntax errors, the *compiler* generates a bytecode file with a **.class** extension. Thus, the preceding command generates a file named **Welcome.class**, as shown

.class bytecode file



**FIGURE 1.6** The Java program-development process consists of repeatedly creating/modifying source code, compiling, and executing programs.



**FIGURE 1.7** You can create a Java source file using Windows Notepad.

bytecode  
Java Virtual Machine (JVM)

in Figure 1.8a. The Java language is a high-level language, but Java bytecode is a low-level language. The *bytecode* is similar to machine instructions but is architecture neutral and can run on any platform that has a *Java Virtual Machine (JVM)*, as shown in Figure 1.8b. Rather than a physical machine, the virtual machine is a program that interprets Java bytecode. This is one of Java’s primary advantages: *Java bytecode can run on a variety of hardware platforms and operating systems*. Java source code is compiled into Java bytecode and Java bytecode is interpreted by the JVM. Your Java code may use the code in the Java library. The JVM executes your code along with the code in the library.

interpret bytecode

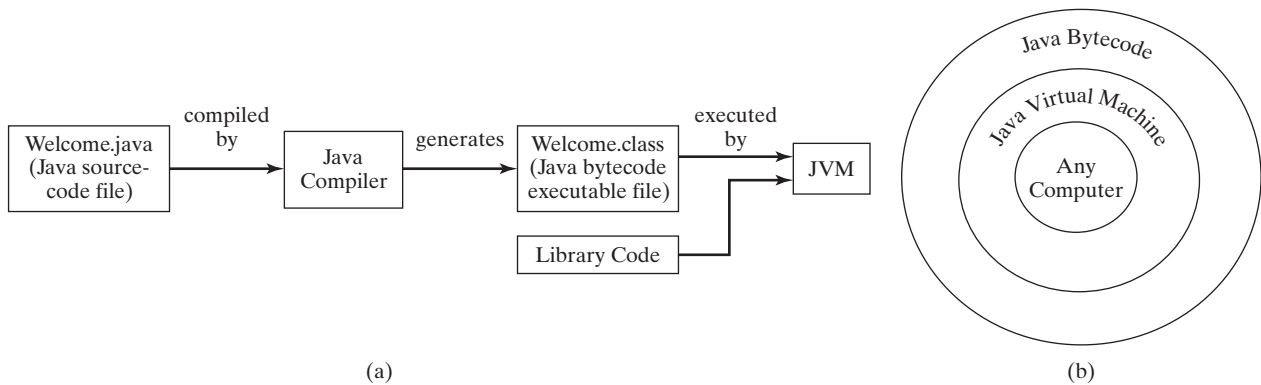
To execute a Java program is to run the program’s bytecode. You can execute the bytecode on any platform with a JVM, which is an interpreter. It translates the individual instructions in the bytecode into the target machine language code one at a time rather than the whole program as a single unit. Each step is executed immediately after it is translated.

run

The following command runs the bytecode for Listing 1.1:

`java Welcome`





**FIGURE 1.8** (a) Java source code is translated into bytecode. (b) Java bytecode can be executed on any computer with a Java Virtual Machine.

Figure 1.9 shows the **javac** command for compiling **Welcome.java**. The compiler generates the **Welcome.class** file, and this file is executed using the **java** command.

javac command  
java command



### Note

For simplicity and consistency, all source-code and class files used in this book are placed under **c:\book** unless specified otherwise.

c:\book

```

Compile —————> c:\book>javac Welcome.java
Show files —————> c:\book>dir Welcome.*
                        Volume in drive C has no label.
                        Volume Serial Number is 2EF7-CA93

                        Directory of c:\book

10/29/2011  03:43 PM                424 Welcome.class
10/29/2011  03:42 PM                176 Welcome.java
                2 File(s)              600 bytes
                0 Dir(s)  70,200,397,824 bytes free

Run —————> c:\book>java Welcome
                Welcome to Java!
c:\book>_
    
```



### VideoNote

Compile and run a Java program

**FIGURE 1.9** The output of Listing 1.1 displays the message “Welcome to Java!”



### Caution

Do not use the extension **.class** in the command line when executing the program. Use **java ClassName** to run the program. If you use **java ClassName.class** in the command line, the system will attempt to fetch **ClassName.class.class**.

java ClassName



### Tip

If you execute a class file that does not exist, a **NoClassDefFoundError** will occur. If you execute a class file that does not have a **main** method or you mistype the **main** method (e.g., by typing **Main** instead of **main**), a **NoSuchMethodError** will occur.

NoClassDefFoundError

NoSuchMethodError



### Note

When executing a Java program, the JVM first loads the bytecode of the class to memory using a program called the **class loader**. If your program uses other classes, the class loader dynamically loads them just before they are needed. After a class is loaded, the JVM uses a program called the **bytecode verifier** to check the validity of the bytecode and

class loader

bytecode verifier

to ensure that the bytecode does not violate Java's security restrictions. Java enforces strict security to make sure that Java class files are not tampered with and do not harm your computer.

use package



### Pedagogical Note

Your instructor may require you to use packages for organizing programs. For example, you may place all programs in this chapter in a package named *chapter1*. For instructions on how to use packages, see Supplement I.F, Using Packages to Organize the Classes in the Text.



Check  
Point

- I.33** What is the Java source filename extension, and what is the Java bytecode filename extension?
- I.34** What are the input and output of a Java compiler?
- I.35** What is the command to compile a Java program?
- I.36** What is the command to run a Java program?
- I.37** What is the JVM?
- I.38** Can Java run on any machine? What is needed to run Java on a computer?
- I.39** If a **NoClassDefFoundError** occurs when you run a program, what is the cause of the error?
- I.40** If a **NoSuchMethodError** occurs when you run a program, what is the cause of the error?

## 1.9 Programming Style and Documentation



Key  
Point

*Good programming style and proper documentation make a program easy to read and help programmers prevent errors.*

programming style

documentation

*Programming style* deals with what programs look like. A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read. *Documentation* is the body of explanatory remarks and comments pertaining to a program. Programming style and documentation are as important as coding. Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read. This section gives several guidelines. For more detailed guidelines, see Supplement I.D, Java Coding Style Guidelines, on the Companion Website.

### 1.9.1 Appropriate Comments and Comment Styles

Include a summary at the beginning of the program that explains what the program does, its key features, and any unique techniques it uses. In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read. It is important to make comments concise so that they do not crowd the program or make it difficult to read.

javadoc comment

In addition to line comments (beginning with `//`) and block comments (beginning with `/*`), Java supports comments of a special type, referred to as *javadoc comments*. javadoc comments begin with `/**` and end with `*/`. They can be extracted into an HTML file using the JDK's **javadoc** command. For more information, see Supplement III.Y, javadoc Comments, on the companion Website.

Use javadoc comments (`/** ... */`) for commenting on an entire class or an entire method. These comments must precede the class or the method header in order to be extracted into a javadoc HTML file. For commenting on steps inside a method, use line comments (`//`).

To see an example of a javadoc HTML file, check out [www.cs.armstrong.edu/liang/javadoc/Exercise1.html](http://www.cs.armstrong.edu/liang/javadoc/Exercise1.html). Its corresponding Java code is shown in [www.cs.armstrong.edu/liang/javadoc/Exercise1.java](http://www.cs.armstrong.edu/liang/javadoc/Exercise1.java).

### 1.9.2 Proper Indentation and Spacing

A consistent indentation style makes programs clear and easy to read, debug, and maintain. *Indentation* is used to illustrate the structural relationships between a program's components or statements. Java can read the program even if all of the statements are on the same long line, but humans find it easier to read and maintain code that is aligned properly. Indent each subcomponent or statement at least *two* spaces more than the construct within which it is nested.

indent code

A single space should be added on both sides of a binary operator, as shown in the following statement:

<code>System.out.println(3+4*4);</code>	Bad style
<code>System.out.println(3 + 4 * 4);</code>	Good style

### 1.9.3 Block Styles

A *block* is a group of statements surrounded by braces. There are two popular styles, *next-line* style and *end-of-line* style, as shown below.

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

Next-line style

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-line style

The next-line style aligns braces vertically and makes programs easy to read, whereas the end-of-line style saves space and may help avoid some subtle programming errors. Both are acceptable block styles. The choice depends on personal or organizational preference. You should use a block style consistently—mixing styles is not recommended. This book uses the *end-of-line* style to be consistent with the Java API source code.

- 1.41** Reformat the following program according to the programming style and documentation guidelines. Use the end-of-line brace style.



```
public class Test
{
    // Main method
    public static void main(String[] args) {
        /** Display output */
        System.out.println("Welcome to Java");
    }
}
```

## 1.10 Programming Errors



*Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.*

syntax errors  
compile errors

### 1.10.1 Syntax Errors

Errors that are detected by the compiler are called *syntax errors* or *compile errors*. Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace. These errors are usually easy to detect because the compiler tells you where they are and what caused them. For example, the program in Listing 1.4 has a syntax error, as shown in Figure 1.10.

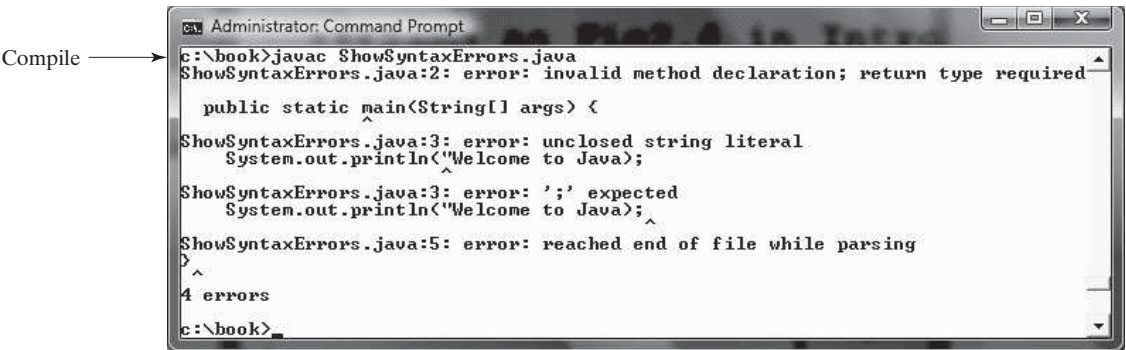
#### LISTING 1.4 ShowSyntaxErrors.java

```
1 public class ShowSyntaxErrors {  
2     public static main(String[] args) {  
3         System.out.println("Welcome to Java");  
4     }  
5 }
```

Four errors are reported, but the program actually has two errors:

- The keyword **void** is missing before **main** in line 2.
- The string **Welcome to Java** should be closed with a closing quotation mark in line 3.

Since a single error will often display many lines of compile errors, it is a good practice to fix errors from the top line and work downward. Fixing errors that occur earlier in the program may also fix additional errors that occur later.



**FIGURE 1.10** The compiler reports syntax errors.



#### Tip

If you don't know how to correct it, compare your program closely, character by character, with similar examples in the text. In the first few weeks of this course, you will probably spend a lot of time fixing syntax errors. Soon you will be familiar with Java syntax and can quickly fix syntax errors.

fix syntax errors

### 1.10.2 Runtime Errors

runtime errors

*Runtime errors* are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out. Input mistakes typically cause runtime errors. An *input error* occurs when the program is

waiting for the user to enter a value, but the user enters a value that the program cannot handle. For instance, if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program.

Another example of runtime errors is division by zero. This happens when the divisor is zero for integer divisions. For instance, the program in Listing 1.5 would cause a runtime error, as shown in Figure 1.11.

### LISTING 1.5 ShowRuntimeErrors.java

```
1 public class ShowRuntimeErrors {
2     public static void main(String[] args) {
3         System.out.println(1 / 0);
4     }
5 }
```

runtime error

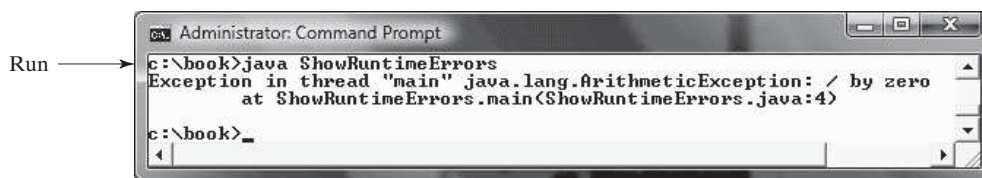


FIGURE 1.11 The runtime error causes the program to terminate abnormally.

### I.10.3 Logic Errors

*Logic errors* occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the program in Listing 1.6 to convert Celsius 35 degrees to a Fahrenheit degree:

logic errors

### LISTING 1.6 ShowLogicErrors.java

```
1 public class ShowLogicErrors {
2     public static void main(String[] args) {
3         System.out.println("Celsius 35 is Fahrenheit degree ");
4         System.out.println((9 / 5) * 35 + 32);
5     }
6 }
```

Celsius 35 is Fahrenheit degree  
67



You will get Fahrenheit 67 degrees, which is wrong. It should be 95.0. In Java, the division for integers is the quotient—the fractional part is truncated—so in Java  $9 / 5$  is 1. To get the correct result, you need to use  $9.0 / 5$ , which results in 1.8.

In general, syntax errors are easy to find and easy to correct because the compiler gives indications as to where the errors came from and why they are wrong. Runtime errors are not difficult to find, either, since the reasons and locations for the errors are displayed on the console when the program aborts. Finding logic errors, on the other hand, can be very challenging. In the upcoming chapters, you will learn the techniques of tracing programs and finding logic errors.

### I.10.4 Common Errors

Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

**Common Error 1: Missing Braces**

The braces are used to denote a block in the program. Each opening brace must be matched by a closing brace. A common error is missing the closing brace. To avoid this error, type a closing brace whenever an opening brace is typed, as shown in the following example.

```
public class Welcome {
```

```
} ←—Type this closing brace right away to match the opening brace
```

If you use an IDE such as NetBeans and Eclipse, the IDE automatically inserts a closing brace for each opening brace typed.

**Common Error 2: Missing Semicolons**

Each statement ends with a statement terminator (;). Often, a new programmer forgets to place a statement terminator for the last statement in a block, as shown in the following example.

```
public static void main(String[] args) {
    System.out.println("Programming is fun!");
    System.out.println("Fundamentals First");
    System.out.println("Problem Driven")
}
```

↑  
Missing a semicolon

**Common Error 3: Missing Quotation Marks**

A string must be placed inside the quotation marks. Often, a new programmer forgets to place a quotation mark at the end of a string, as shown in the following example.

```
System.out.println("Problem Driven");
```

↑  
Missing a quotation mark

If you use an IDE such as NetBeans and Eclipse, the IDE automatically inserts a closing quotation mark for each opening quotation mark typed.

**Common Error 4: Misspelling Names**

Java is case sensitive. Misspelling names is a common error for new programmers. For example, the word `main` is misspelled as `Main` and `String` is misspelled as `string` in the following code.

```
1 public class Test {
2     public static void Main(string[] args) {
3         System.out.println((10.5 + 2 * 3) / (45 - 3.5));
4     }
5 }
```



- I.42** What are syntax errors (compile errors), runtime errors, and logic errors?
- I.43** Give examples of syntax errors, runtime errors, and logic errors.
- I.44** If you forget to put a closing quotation mark on a string, what kind error will be raised?
- I.45** If your program needs to read integers, but the user entered strings, an error would occur when running this program. What kind of error is this?
- I.46** Suppose you write a program for computing the perimeter of a rectangle and you mistakenly write your program so that it computes the area of a rectangle. What kind of error is this?

**1.47** Identify and fix the errors in the following code:

```
1 public class Welcome {
2     public void Main(String[] args) {
3         System.out.println('Welcome to Java!');
4     }
5 }
```

## 1.1.1 Developing Java Programs Using NetBeans

*You can edit, compile, run, and debug Java Programs using NetBeans.*



**VideoNote**

NetBeans brief tutorial

NetBeans and Eclipse are two free popular integrated development environments for developing Java programs. They are easy to learn if you follow simple instructions. We recommend that you use either one for developing Java programs. This section gives the essential instructions to guide new users to create a project, create a class, compile, and run a class in NetBeans. The use of Eclipse will be introduced in the next section. For instructions on downloading and installing latest version of NetBeans, see Supplement II.B.

### 1.1.1.1 Creating a Java Project

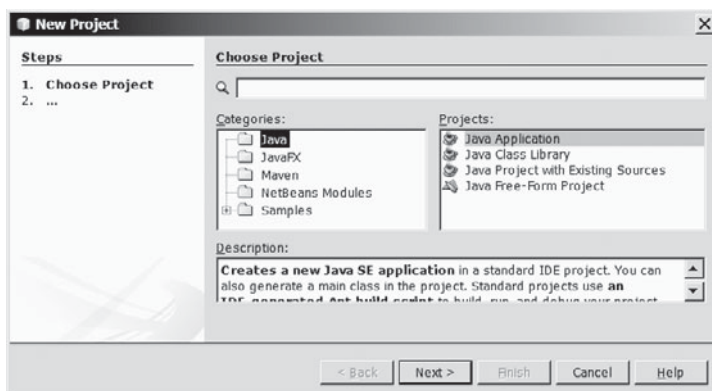
Before you can create Java programs, you need to first create a project. A project is like a folder to hold Java programs and all supporting files. You need to create a project only once. Here are the steps to create a Java project:

1. Choose *File, New Project* to display the New Project dialog box, as shown in Figure 1.12.
2. Select Java in the Categories section and Java Application in the Projects section and click *Next* to display the New Java Application dialog box, as shown in Figure 1.13.
3. Type **demo** in the Project Name field and **c:\michael** in Project Location field. Uncheck *Use Dedicated Folder for Storing Libraries* and uncheck *Create Main Class*.
4. Click *Finish* to create the project, as shown in Figure 1.14.

### 1.1.1.2 Creating a Java Class

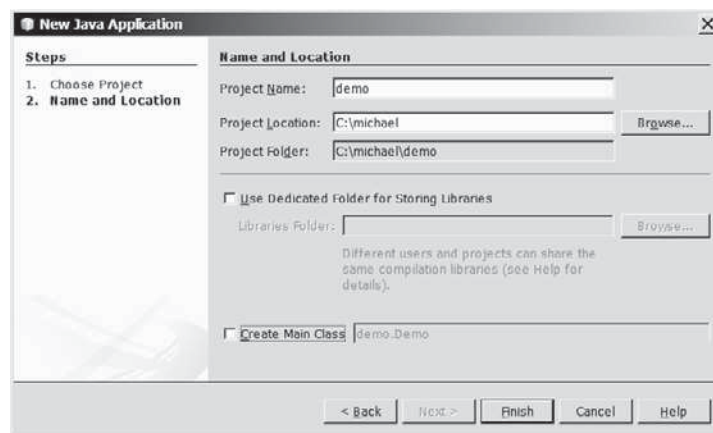
After a project is created, you can create Java programs in the project using the following steps:

1. Right-click the demo node in the project pane to display a context menu. Choose *New, Java Class* to display the New Java Class dialog box, as shown in Figure 1.15.

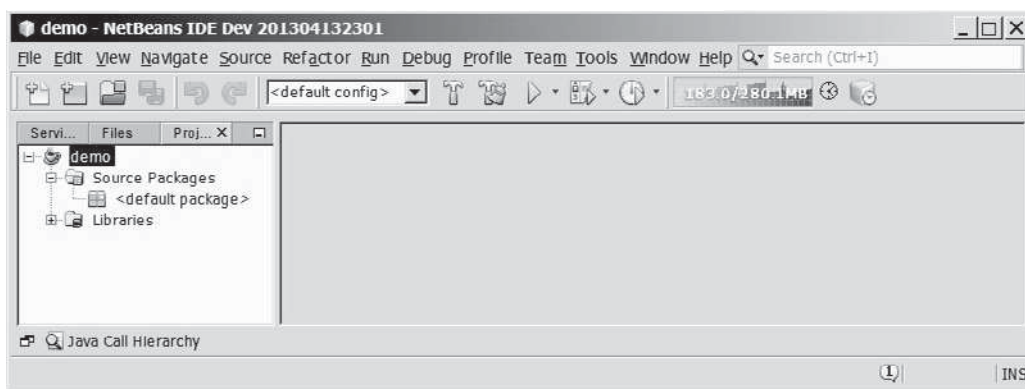


**FIGURE 1.12** The New Project dialog is used to create a new project and specify a project type.

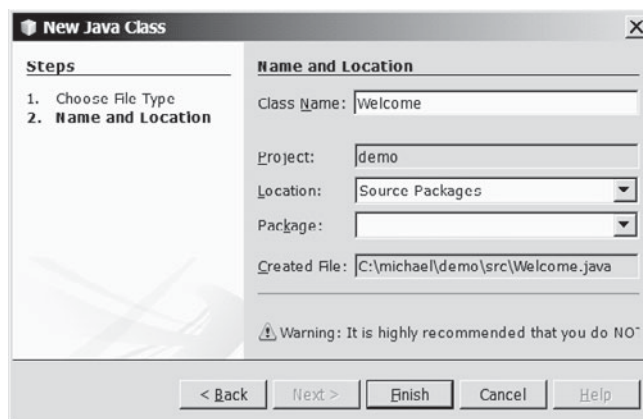




**FIGURE 1.13** The New Java Application dialog is for specifying a project name and location.

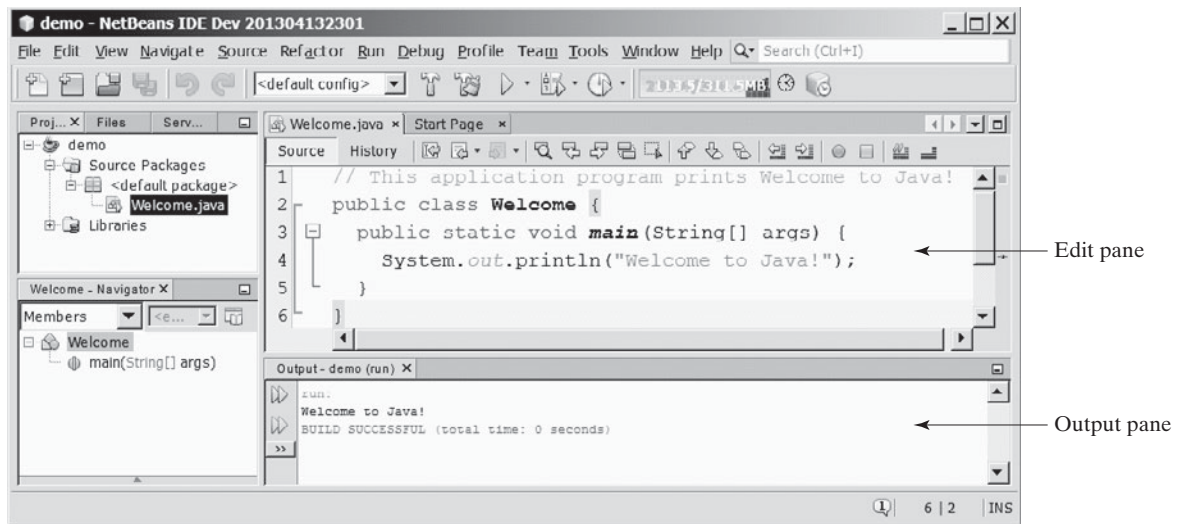


**FIGURE 1.14** A New Java project named demo is created.



**FIGURE 1.15** The New Java Class dialog box is used to create a new Java class.

2. Type **Welcome** in the Class Name field and select the Source Packages in the Location field. Leave the Package field blank. This will create a class in the default package.
3. Click *Finish* to create the Welcome class. The source code file Welcome.java is placed under the <default package> node.
4. Modify the code in the Welcome class to match Listing 1.1 in the text, as shown in Figure 1.16.



**FIGURE 1.16** You can edit a program and run it in NetBeans.

### 1.11.3 Compiling and Running a Class

To run **Welcome.java**, right-click **Welcome.java** to display a context menu and choose *Run File*, or simply press Shift + F6. The output is displayed in the Output pane, as shown in Figure 1.16. The *Run File* command automatically compiles the program if the program has been changed.

## 1.12 Developing Java Programs Using Eclipse

*You can edit, compile, run, and debug Java Programs using Eclipse.*



The preceding section introduced developing Java programs using NetBeans. You can also use Eclipse to develop Java programs. This section gives the essential instructions to guide new users to create a project, create a class, and compile/run a class in Eclipse. For instructions on downloading and installing latest version of Eclipse, see Supplement II.D.

### 1.12.1 Creating a Java Project

Before creating Java programs in Eclipse, you need to first create a project to hold all files. Here are the steps to create a Java project in Eclipse:



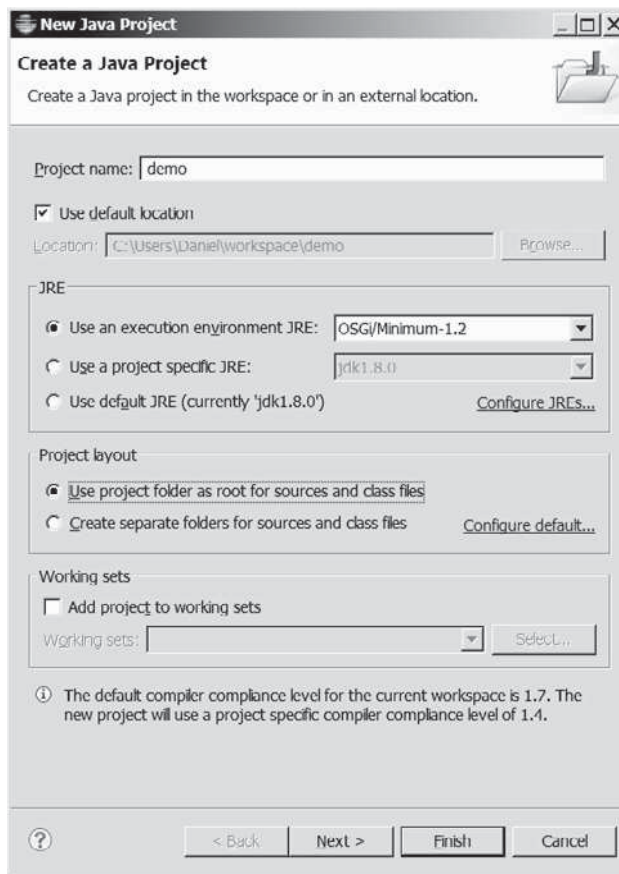
**VideoNote**  
Eclipse brief tutorial

1. Choose *File, New, Java Project* to display the New Project wizard, as shown in Figure 1.17.
2. Type **demo** in the Project name field. As you type, the Location field is automatically set by default. You may customize the location for your project.
3. Make sure that you selected the options *Use project folder as root for sources and class files* so that the .java and .class files are in the same folder for easy access.
4. Click *Finish* to create the project, as shown in Figure 1.18.

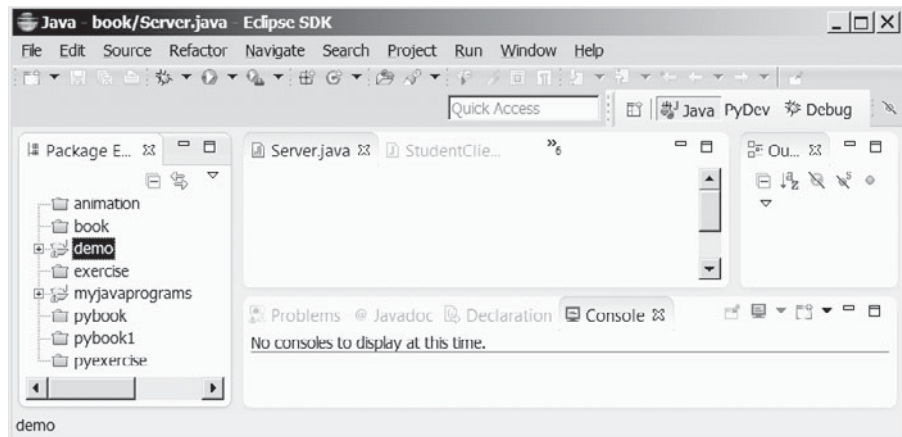
### 1.12.2 Creating a Java Class

After a project is created, you can create Java programs in the project using the following steps:

1. Choose *File, New, Class* to display the New Java Class wizard.
2. Type **Welcome** in the Name field.

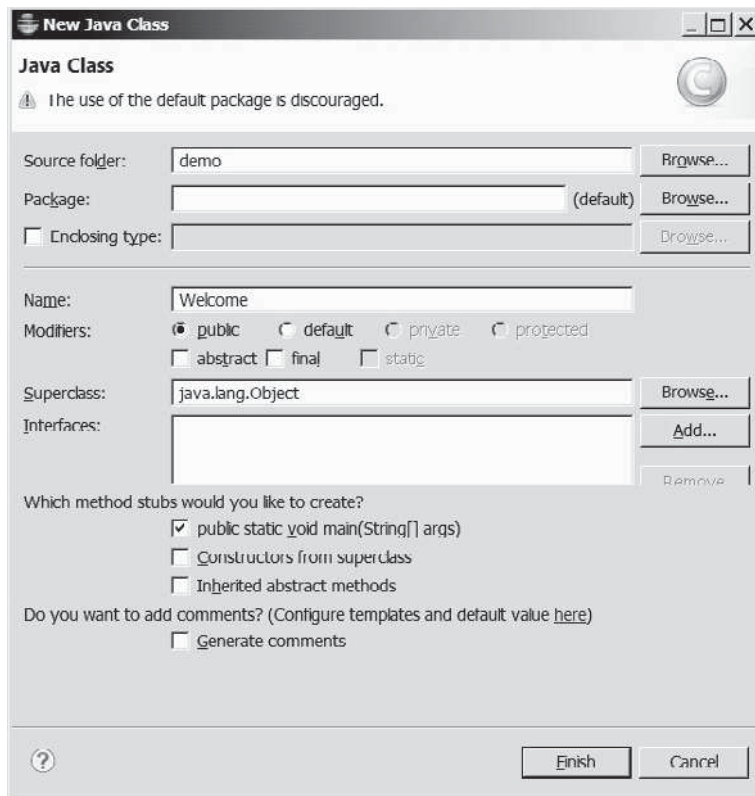


**FIGURE 1.17** The New Java Project dialog is for specifying a project name and properties.



**FIGURE 1.18** A New Java project named demo is created.

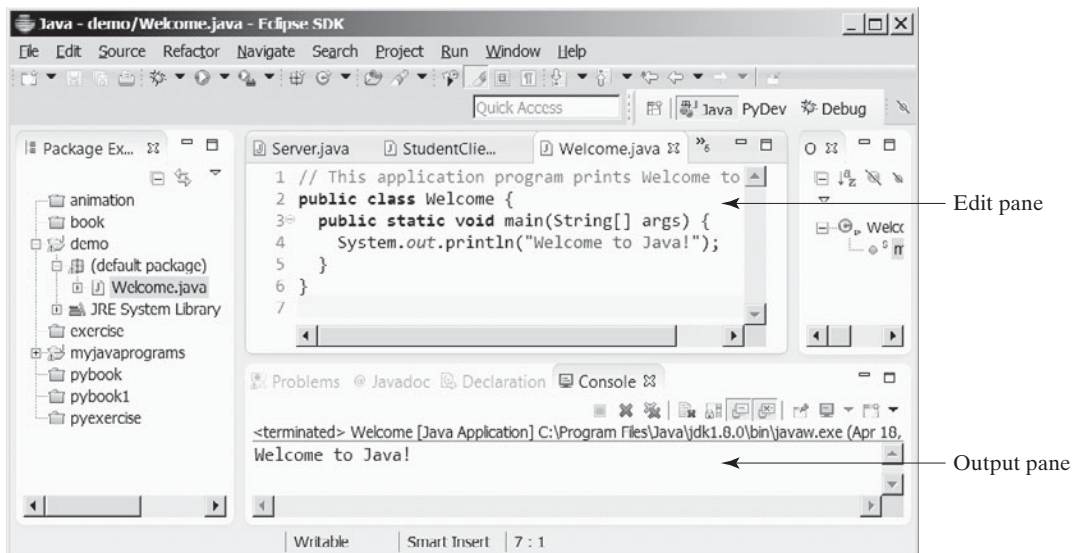
3. Check the option `public static void main(String[] args)`.
4. Click *Finish* to generate the template for the source code `Welcome.java`, as shown in Figure 1.19.



**FIGURE 1.19** The New Java Class dialog box is used to create a new Java class.

### 1.12.3 Compiling and Running a Class

To run the program, right-click the class in the project to display a context menu. Choose *Run, Java Application* in the context menu to run the class. The output is displayed in the Console pane, as shown in Figure 1.20.



**FIGURE 1.20** You can edit a program and run it in Eclipse.

## KEY TERMS

---

Application Program Interface (API)	11	Java Virtual Machine (JVM)	16
assembler	7	javac command	17
assembly language	7	keyword (or reserved word)	13
bit	3	library	11
block	13	line comment	13
block comment	13	logic error	21
bus	2	low-level language	7
byte	3	machine language	7
bytecode	16	main method	13
bytecode verifier	17	memory	4
cable modem	6	modem	00
central processing unit (CPU)	3	motherboard	3
class loader	17	network interface card (NIC)	6
comment	13	operating system (OS)	9
compiler	8	pixel	6
console	12	program	2
dot pitch	6	programming	2
DSL (digital subscriber line)	6	runtime error	20
encoding scheme	3	screen resolution	6
hardware	2	software	2
high-level language	8	source code	8
integrated development environment (IDE)	12	source program	8
interpreter	8	statement	8
java command	17	statement terminator	13
Java Development Toolkit (JDK)	12	storage devices	4
Java language specification	11	syntax error	20

Supplement I.A



### Note

The above terms are defined in this chapter. Supplement I.A, Glossary, lists all the key terms and descriptions in the book, organized by chapters.

## CHAPTER SUMMARY

---

1. A computer is an electronic device that stores and processes data.
2. A computer includes both *hardware* and *software*.
3. Hardware is the physical aspect of the computer that can be touched.
4. Computer *programs*, known as *software*, are the invisible instructions that control the hardware and make it perform tasks.
5. Computer *programming* is the writing of instructions (i.e., code) for computers to perform.
6. The *central processing unit (CPU)* is a computer's brain. It retrieves instructions from *memory* and executes them.
7. Computers use zeros and ones because digital devices have two stable states, referred to by convention as zero and one.

8. A *bit* is a binary digit 0 or 1.
9. A *byte* is a sequence of 8 bits.
10. A kilobyte is about 1,000 bytes, a megabyte about 1 million bytes, a gigabyte about 1 billion bytes, and a terabyte about 1,000 gigabytes.
11. Memory stores data and program instructions for the CPU to execute.
12. A memory unit is an ordered sequence of bytes.
13. Memory is volatile, because information is lost when the power is turned off.
14. Programs and data are permanently stored on *storage devices* and are moved to memory when the computer actually uses them.
15. The *machine language* is a set of primitive instructions built into every computer.
16. *Assembly language* is a *low-level programming language* in which a mnemonic is used to represent each machine-language instruction.
17. *High-level languages* are English-like and easy to learn and program.
18. A program written in a high-level language is called a *source program*.
19. A *compiler* is a software program that translates the source program into a *machine-language program*.
20. The *operating system (OS)* is a program that manages and controls a computer's activities.
21. Java is platform independent, meaning that you can write a program once and run it on any computer.
22. Java programs can be embedded in HTML pages and downloaded by Web browsers to bring live animation and interaction to Web clients.
23. The Java source file name must match the public class name in the program. Java source code files must end with the **.java** extension.
24. Every class is compiled into a separate bytecode file that has the same name as the class and ends with the **.class** extension.
25. To compile a Java source-code file from the command line, use the **javac** command.
26. To run a Java class from the command line, use the **java** command.
27. Every Java program is a set of class definitions. The keyword **class** introduces a class definition. The contents of the class are included in a *block*.
28. A block begins with an opening brace (**{**) and ends with a closing brace (**}**).
29. Methods are contained in a class. To run a Java program, the program must have a **main** method. The **main** method is the entry point where the program starts when it is executed.