CHAPTER

1

Introduction to Computers, Programs, and Java™

Objectives

- To understand computer basics, programs, and operating systems (§§1.2–1.4).
- To describe the relationship between Java and the World Wide Web (§1.5).
- To understand the meaning of Java language specification, API, JDK™, JRE™, and IDE (§1.6).
- To write a simple Java program (§1.7).
- To display output on the console (§1.7).
- To explain the basic syntax of a Java program (§1.7).
- To create, compile, and run Java programs (§1.8).
- To use sound Java programming style and document programs properly (§1.9).
- To explain the differences between syntax errors, runtime errors, and logic errors (§1.10).
- To develop Java programs using NetBeans™ (§1.11).
- To develop Java programs using Eclipse™ (§1.12).

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 instructions that tell a computer—or a computerized device—what to do.

Software is all around you, even in devices 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 there are so many programming languages available, it would be natural for you to wonder which one is best. However, in truth, there is no "best" language. Each one has its own strengths and weaknesses. Experienced programmers know 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 (OSs). If you are already familiar with such terms as central processing unit (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 (see 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 the keyboard)
- Output devices (such as monitors and printers)
- Communication devices (such as modems and network interface cards (NIC))

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,

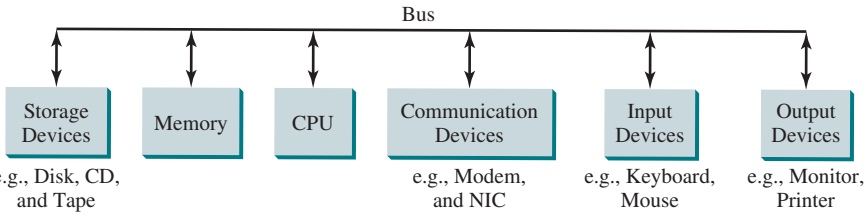


FIGURE 1.1 A computer consists of a CPU, memory, storage devices, input devices, output devices, and communication devices.

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 the 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, and 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 that 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 Hz equaling 1 pulse per second. In the 1990s, computers measured clock speed in *megahertz* (*MHz*, i.e., 1 million pulses per second), but CPU speed has been improving continuously; the clock speed of a computer is now usually stated in *gigahertz* (*GHz,* i.e., 1 billion pulses per second). 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 the 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, four, and even eight 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 and numbers into data with which the computer can actually work. Most schemes translate each character into

encoding scheme

a predetermined string of bits. In the popular ASCII encoding scheme, for example, the character **C** is represented as **01000011** in 1 byte.

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

kilobyte (KB)
- A *kilobyte (KB)* is about 1,000 bytes.

megabyte (MB)
- A *megabyte (MB)* is about 1 million bytes.

gigabyte (GB)
- A *gigabyte (GB)* is about 1 billion bytes.

terabyte (TB)
- 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

memory

A computer's *memory* consists of an ordered sequence of bytes for storing programs as well as data with which the program is working. 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.

unique address

RAM

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)*.

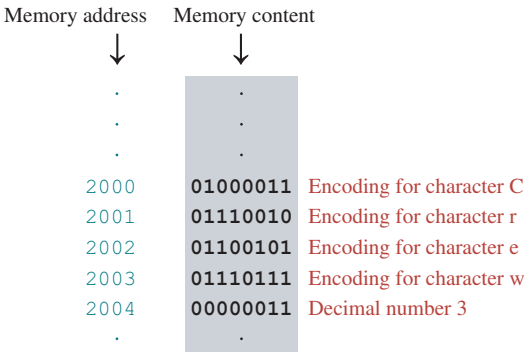


FIGURE 1.2 Memory stores data and program instructions in uniquely addressed memory locations.

Today's personal computers usually have at least 4 GB of RAM, but they more commonly have 8 to 32 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 saved in memory 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 actually uses them, to memory, which operates at much faster speeds than permanent storage devices can.

storage devices

There are four main types of storage devices:

- Magnetic disk drives
- Optical disc drives (CD and DVD)
- Universal serial bus (USB) flash drives
- Cloud storage

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

drive

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 1 terabyte of data to 4 terabytes of data. Hard disk drives are usually encased inside the computer, but removable hard disks are also available.

hard disk

CDs and DVDs

CD stands for compact disc. There are three types of CDs: CD-ROM, CD-R, and CD-RW. A CD-ROM is a prepressed disc. It was popular for distributing software, music, and video. Software, music, and video are now increasingly distributed on the Internet without using CDs. A *CD-R* (CD-Recordable) is a write-once medium. It can be used to record data once and read any number of times. A *CD-RW* (CD-ReWritable) can be used like a hard disk; that is, you can write data onto the disc, then overwrite that data with new data. A single CD can hold up to 700 MB.

CD-ROM
CD-R
CD-RW

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. There are two types of DVDs: DVD-R (Recordable) and DVD-RW (ReWritable).

DVD

USB Flash Drives

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

An 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.

Cloud Storage

Storing data on the cloud is becoming popular. Many companies provide cloud service on the Internet. For example, you can store Microsoft Office documents in Google Docs. Google Docs can be accessed from docs.google.com on the Chrome browser. The documents can be easily shared with others. Microsoft OneDrive is provided free to Windows user for storing files. The data stored in the cloud can be accessed from any device on the Internet.

1.2.5 Input and Output Devices

Input and output devices let the user communicate with the computer. The most common input devices are the *keyboard* and *mouse*. 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 key

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.

modifier key

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.

numeric keypad

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

arrow keys

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.

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 is the display.

Touchscreens

The cellphones, tablets, appliances, electronic voting machines, as well as some computers use touchscreens. A touchscreen is integrated with a monitor to enable users to enter input and control the display using a finger.

1.2.6 Communication Devices

Computers can be networked through communication devices, such as a dial-up modem (*mo*dulator/*dem*odulator), a digital subscriber line (DSL) or cable modem, a wired network interface card, or a wireless adapter.

dial-up modem

- A *dial-up modem* uses a phone line to dial a phone number to connect to the Internet 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. Dial-up modem was used in the 90s and is now replaced by DSL and cable modem.

cable modem

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

network interface card (NIC)
local area network (LAN)

- A *network interface card (NIC)* is a device that connects a computer to a *local area network (LAN)*. LANs are commonly used to connect computers within a limited

area such as a school, a home, and an office. A high-speed NIC called *1000BaseT* can transfer data at 1,000 million bits per second (mbps). million bits per second (mbps)

- Wi-Fi, a special type of wireless networking, is common in homes, businesses, and schools to connect computers, phones, tablets, and printers to the Internet without the need for a physical wired connection. Wi-Fi

Note

Answers to the CheckPoint questions are available at **www.pearsonglobaleditions.com/liang**. Choose this book and click Companion Website to select CheckPoint.

1.2.1 What are hardware and software?

1.2.2 List the five major hardware components of a computer.

1.2.3 What does the acronym CPU stand for? What unit is used to measure CPU speed?

1.2.4 What is a bit? What is a byte?

1.2.5 What is memory for? What does RAM stand for? Why is memory called RAM?

1.2.6 What unit is used to measure memory size? What unit is used to measure disk size?

1.2.7 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 as follows: 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 as follows: 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

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

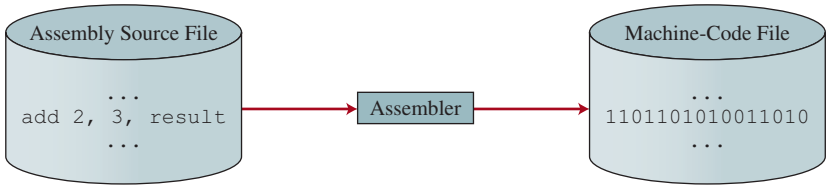


FIGURE 1.3 An assembler translates assembly-language instructions into machine code.

low-level language

corresponds to an instruction in machine code. Writing in assembly language 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.

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 similar to English 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. Developed for the Department of Defense and used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. Designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. Combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	An object-oriented language, based on C
C#	Pronounced "C Sharp." An object-oriented programming language 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. An object-oriented programming language, widely used for developing platform-independent Internet applications.
JavaScript	A Web programming language developed by Netscape
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. 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. Enables the programmers to rapidly develop Windows-based applications.

source program
source code
interpreter
compiler

A program written in a high-level language is called a *source program* or *source code*. Because a computer cannot execute a source program, a source program must be translated into machine code for execution. The translation can be done using another programming tool 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, then executes it right away, as shown in Figure 1.4a. Note 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.

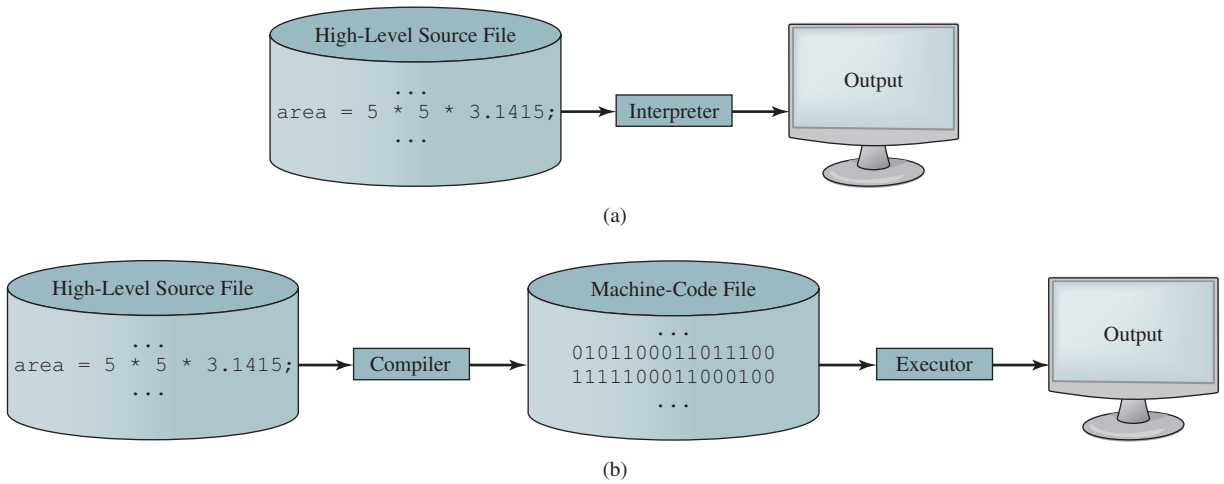


FIGURE 1.4 (a) An interpreter translates and executes a program one statement at a time. (b) A compiler translates the entire source program into a machine-language file for execution.

1.3.1 What language does the CPU understand?

1.3.2 What is an assembly language? What is an assembler?

1.3.3 What is a high-level programming language? What is a source program?

1.3.4 What is an interpreter? What is a compiler?

1.3.5 What is the difference between an interpreted language and a compiled language?

Check Point

1.4 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.*

Key Point

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.

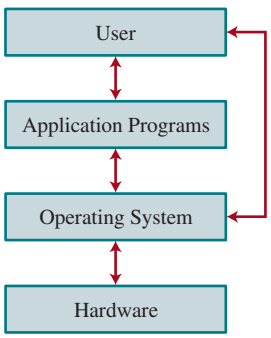


FIGURE 1.5 Users and applications access the computer's hardware via the operating system.

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 different programs and users working at the same time do not interfere with each other. In addition, the OS is responsible for security, ensuring 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, and input and output devices) and for allocating and assigning them to run the program.

1.4.3 Scheduling Operations

multiprogramming
multithreading
multiprocessing

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 allows multiple programs such as Microsoft Word, E-mail, and web browser 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 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 program. These two tasks may run concurrently.

Multiprocessing is similar to multithreading. The difference is that multithreading is for running multithreads concurrently within one program, but multiprocessing is for running multiple programs concurrently using multiple processors.

1.4.1 What is an operating system? List some popular operating systems.

1.4.2 What are the major responsibilities of an operating system?

1.4.3 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.

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*, *multithreaded*, and *dynamic*. For the anatomy of Java characteristics, see liveexample.pearsoncmg.com/etc/JavaCharacteristics.pdf.

Java is a full-featured, general-purpose programming language that can be used to develop robust mission-critical applications. It is employed not only on desktop computers, but also on servers and mobile devices. Today, more than 3 billion devices run Java. Most major companies use Java in some applications. Most server-side applications were developed using Java. Java was used to develop the code to communicate with and control the robotic rover on Mars. The software for Android cell phones is developed using Java.

Java initially became attractive because Java programs can run from a web browser. Such programs are called *applets*. Today applets are no longer allowed to run from a Web browser due to security issues. Java, however, is now very popular for developing applications on web servers. These applications process data, perform computations, and generate dynamic webpages. Many commercial Websites are developed using Java on the backend.

1.5.1 Who invented Java? Which company owns Java now?

1.5.2 What is a Java applet?

1.5.3 What programming language does Android use?

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

Java syntax is defined in the Java language specification, and the Java library is defined in the Java application program interface (API). The JDK is the software for compiling 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 docs.oracle.com/javase/specs/.

Java language specification

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 the latest Java API documentation at https://docs.oracle.com/en/java/javase/11/.

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 on desktop.
- *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.

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 Development Toolkit (JDK)

Java SE 11 (or simply Java 11), is used in this book. Oracle releases each version with a *Java Development Toolkit (JDK)*. For Java 11, the Java Development Toolkit is called *JDK 11*.

Java Runtime Environment (JRE)

Integrated development environment

The JDK consists of a set of separate programs, each invoked from a command line, for compiling, running, and testing Java programs. The program for running Java programs is known as *Java Runtime Environment (JRE)*. 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.

1.6.1 What is the Java language specification?

1.6.2 What does JDK stand for? What does JRE stand for?

1.6.3 What does IDE stand for?

1.6.4 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 given in Listing 1.1.

LISTING 1.1 Welcome.java

class
main method
display message

```
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  }
```

VideoNote

Your first Java program

```
Welcome to Java!
```

line numbers

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

class name

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`.

main method

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

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*.

statement terminator
keyword

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 keywords 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:

comment

line comment

block comment

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

A pair of 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:

block

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

Method block

Class 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.

match braces

Caution

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

case sensitive

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

special characters

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 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

common errors

syntax rules

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.

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.print("(10.5 + 2 * 3) / (45 - 3.5) = ");
4      System.out.println((10.5 + 2 * 3) / (45 - 3.5));
5    }
6  }
```

```
(10.5 + 2 * 3) / (45 - 3.5) = 0.39759036144578314
```

print vs. println

The `print` method in line 3

```
System.out.print("(10.5 + 2 * 3) / (45 - 3.5) = ");
```

is identical to the `println` method except that `println` moves to the beginning of the next line after displaying the string, but `print` does not advance to the next line when completed.

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.

1.7.1 What is a keyword? List some Java keywords.

1.7.2 Is Java case sensitive? What is the case for Java keywords?

1.7.3 What is a comment? Is the comment ignored by the compiler? How do you denote a comment line and a comment paragraph?

1.7.4 What is the statement to display a string on the console?

1.7.5 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 (JVM).

Key Point

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, 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 1.11 and 1.12 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

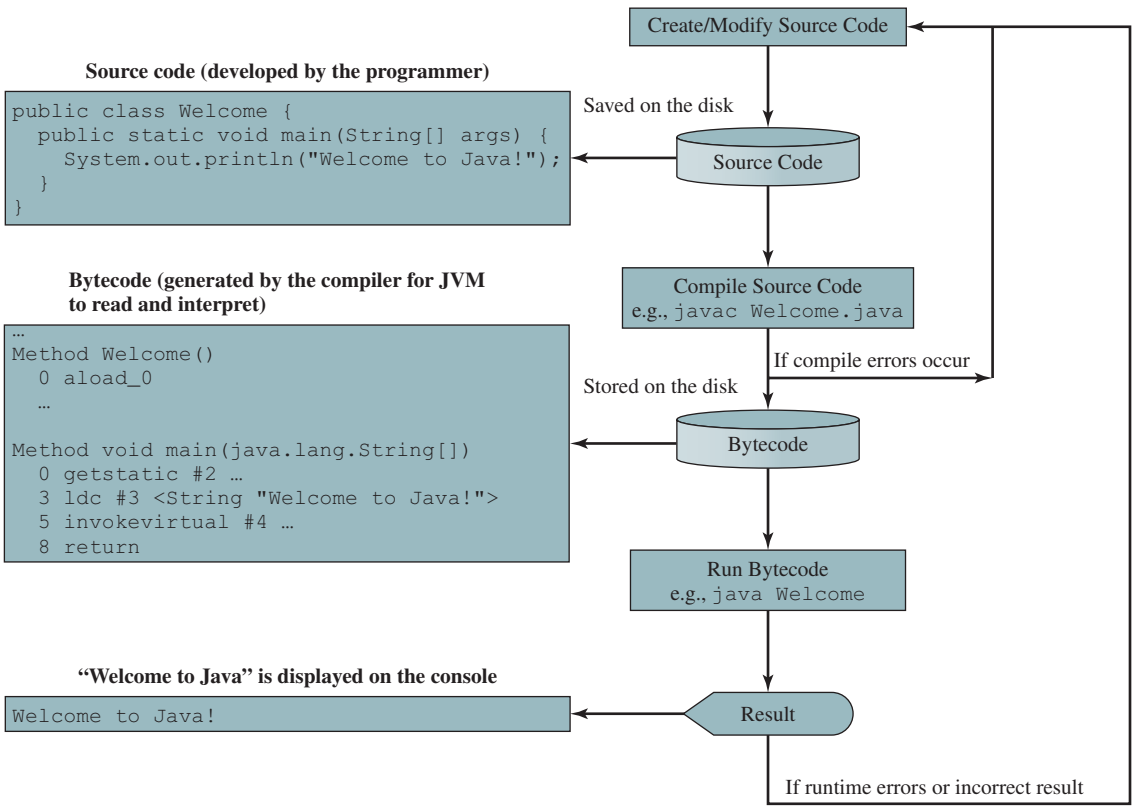


FIGURE 1.6 The Java program-development process consists of repeatedly creating/modifying source code, compiling, and executing programs.

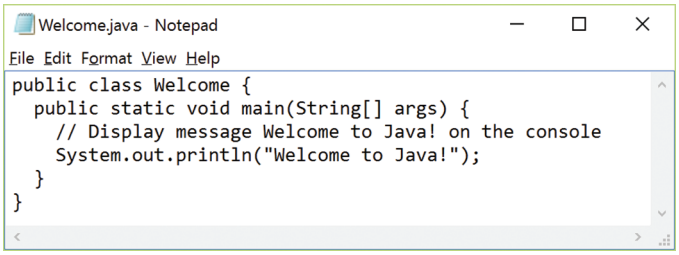


FIGURE 1.7 You can create a Java source file using Windows Notepad.

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,

compile

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

```
javac Welcome.java
```

Note

You must first install and configure the JDK before you can compile and run programs. See Supplement I.A, Installing and Configuring JDK 11, 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.B, 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.

Supplement I.B

Supplement I.C

.class bytecode file

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 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.

bytecode

Java Virtual Machine (JVM)

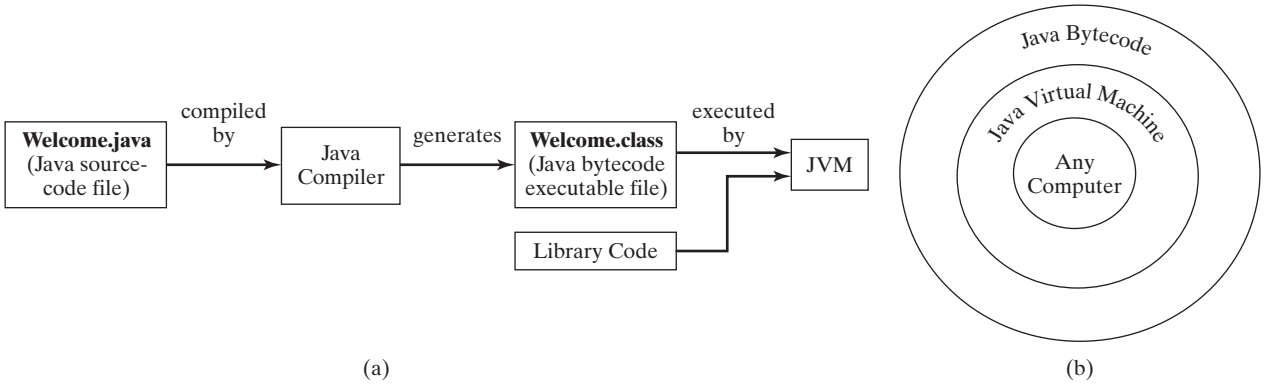


FIGURE 1.8 (a) Java source code is translated into bytecode. (b) Java bytecode can be executed on any computer with a Java Virtual Machine.

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.

interpret bytecode

The following command runs the bytecode for Listing 1.1:

run

```
java Welcome
```

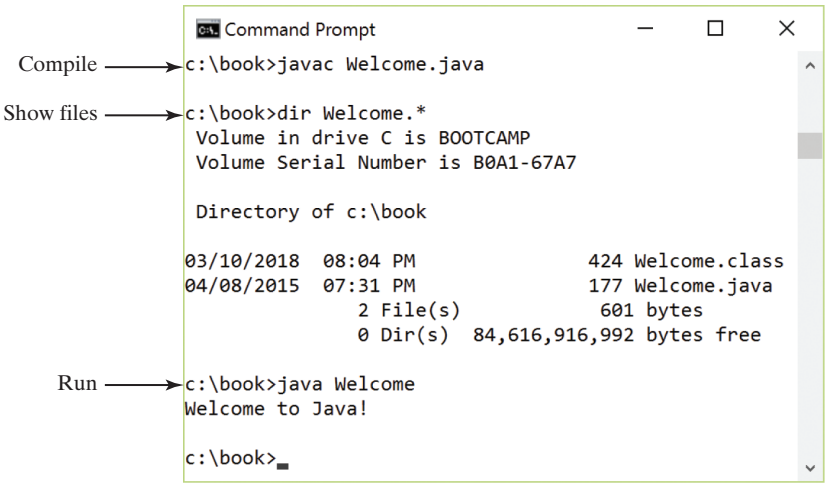
Figure 1.9 shows the `javac` command for compiling **Welcome.java**. The compiler generates the **Welcome.class** file, and this file is executed using the `java` command.

`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



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

Note

In JDK 11, you can use `java ClassName.java` to compile and run a single-file source code program. This command combines compiling and running in one command. A single-file source code program contains only one class in the file. This is the case for all of our programs in the first eight chapters.

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

class loader

bytecode verifier

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

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 *chapter 1*. For instructions on how to use packages, see Supplement I.F, Using Packages to Organize the Classes in the Text.

use package

1.8.1 What is the Java source filename extension, and what is the Java bytecode filename extension?

1.8.2 What are the input and output of a Java compiler?

1.8.3 What is the command to compile a Java program?

1.8.4 What is the command to run a Java program?

1.8.5 What is the JVM?

1.8.6 Can Java run on any machine? What is needed to run Java on a computer?

1.8.7 If a `NoClassDefFoundError` occurs when you run a program, what is the cause of the error?

1.8.8 If a `NoSuchMethodError` occurs when you run a program, what is the cause of the error?

1.9 Programming Style and Documentation

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 a 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.C, 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.X, 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 liveexample.pearsoncmg.com/javadoc/Exercise1.html. Its corresponding Java code is shown in liveexample.pearsoncmg.com/javadoc/Exercise1.txt.

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 (a), rather in (b).

```
System.out.println(3 + 4 * 4);
```

(a) Good style

```
System.out.println(3+4*4);
```

(b) Bad 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.

1.9.1 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.

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.

syntax errors
compile errors

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.

Compile →

```
Command Prompt
c:\book>javac ShowSyntaxErrors.java
ShowSyntaxErrors.java:3: error: invalid method declaration; return type required
  public static main(String[] args) {
                ^
ShowSyntaxErrors.java:4: error: unclosed string literal
    System.out.println("Welcome to Java);
                       ^
ShowSyntaxErrors.java:4: error: ';' expected
    System.out.println("Welcome to Java);
                                        ^
ShowSyntaxErrors.java:6: error: reached end of file while parsing
}
 ^
4 errors

c:\book>
```

FIGURE 1.10 The compiler reports syntax errors.

Tip

If you don't know how to correct an error, 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

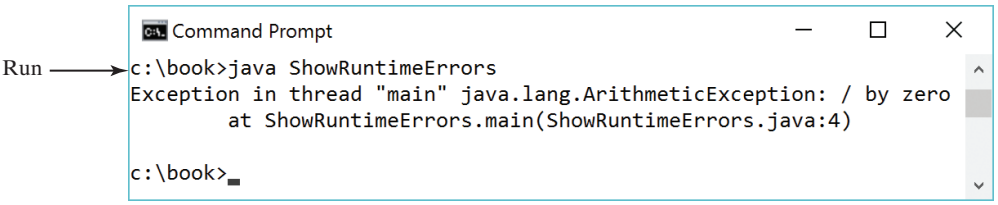


FIGURE 1.11 The runtime error causes the program to terminate abnormally.

1.10.3 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.print("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.

1.10.4 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:

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

1.10.1 What are syntax errors (compile errors), runtime errors, and logic errors?

1.10.2 Give examples of syntax errors, runtime errors, and logic errors.

1.10.3 If you forget to put a closing quotation mark on a string, what kind of error will be raised?

1.10.4 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?

1.10.5 Suppose you write a program for computing the perimeter of a rectangle and you mistakenly write your program so it computes the area of a rectangle. What kind of error is this?

1.10.6 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  )
```

1.11 Developing Java Programs Using NetBeans

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

Note

Section 1.8 introduced developing programs from the command line. Many of our readers also use an IDE. This section and next section introduce two most popular Java IDEs: NetBeans and Eclipse. These two sections may be skipped.

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. To use JDK 11, you need NetBeans 9 or higher. For instructions on downloading and installing latest version of NetBeans, see Supplement II.B.

VideoNote

NetBeans brief tutorial

1.11.1 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 then 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.

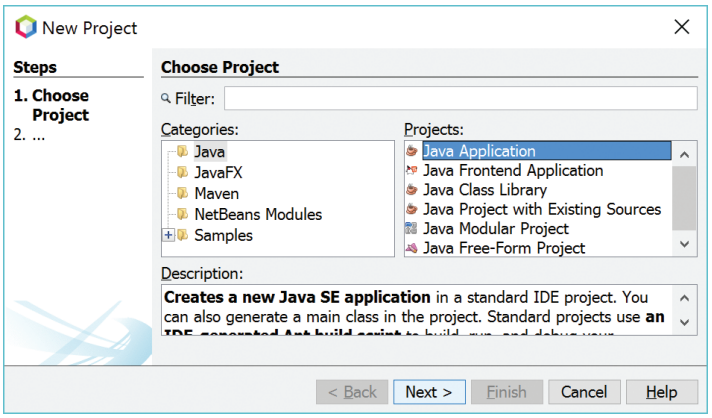


FIGURE 1.12 The New Project dialog is used to create a new project and specify a project type. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

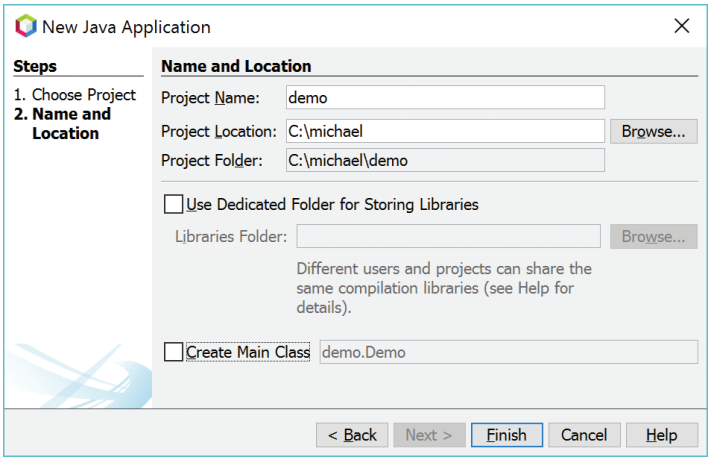


FIGURE 1.13 The New Java Application dialog is for specifying a project name and location. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

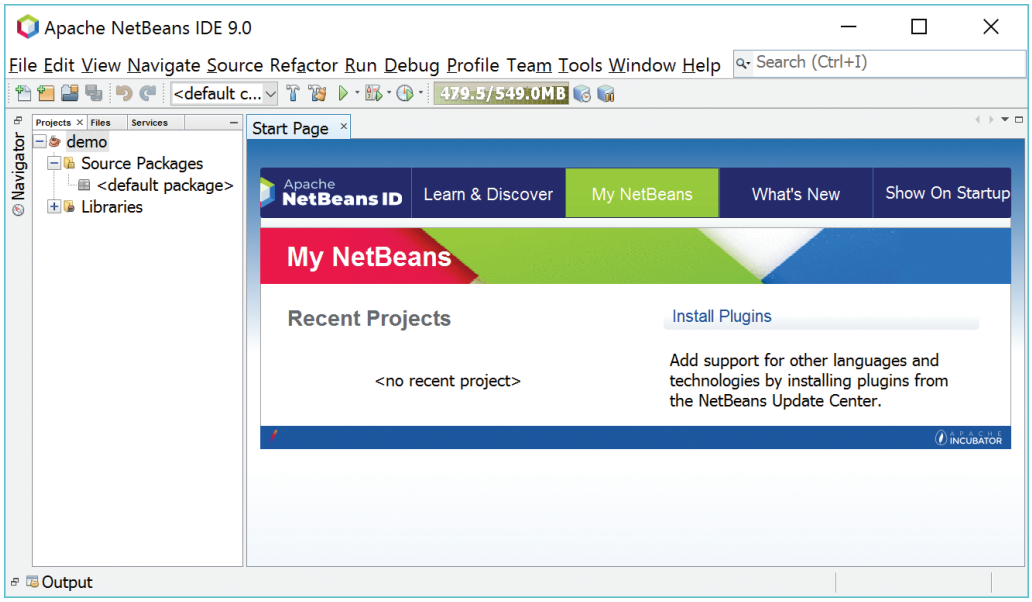


FIGURE 1.14 A New Java project named demo is created. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

1.11.2 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.
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.

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.

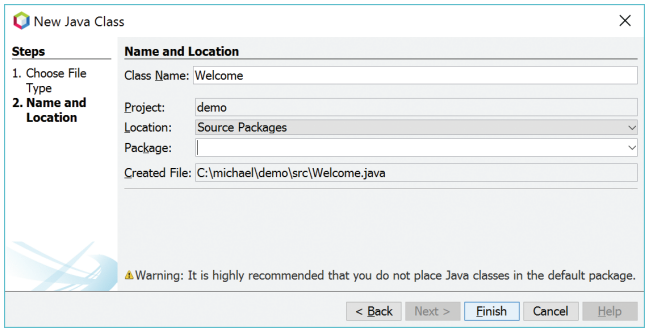


FIGURE 1.15 The New Java Class dialog box is used to create a new Java class. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

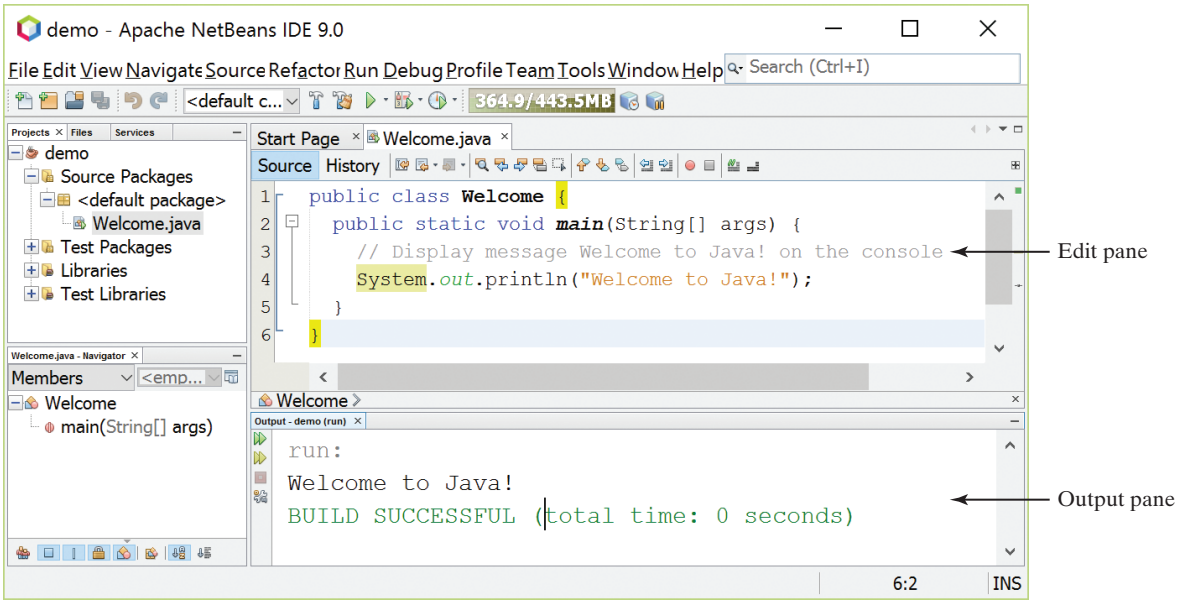


FIGURE 1.16 You can edit a program and run it in NetBeans. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

1.12 Developing Java Programs Using Eclipse

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

Key Point

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. To use JDK 11, you need Eclipse 4.9 or higher. For instructions on downloading and installing latest version of Eclipse, see Supplement II.D.

VideoNote
Eclipse brief tutorial

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:

1. Choose *File*, *New*, *Java Project* to display the New Project wizard, as shown in Figure 1.17.

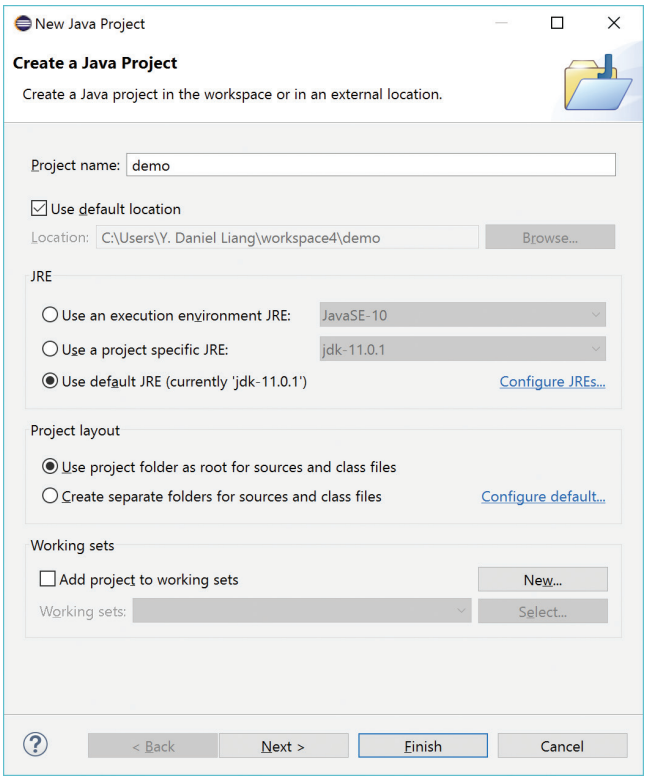


FIGURE 1.17 The New Java Project dialog is for specifying a project name and the properties. *Source*: Eclipse Foundation, Inc.

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 you selected the options *Use project folder as root for sources and class files* so 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.

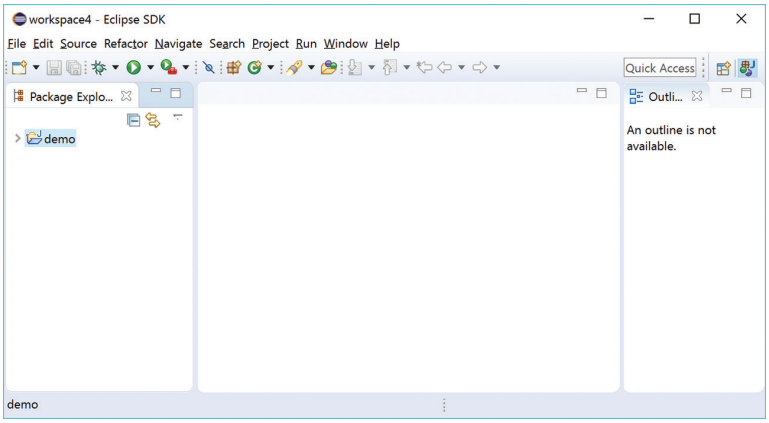


FIGURE 1.18 A New Java project named demo is created. *Source*: Eclipse Foundation, Inc.

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.
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.

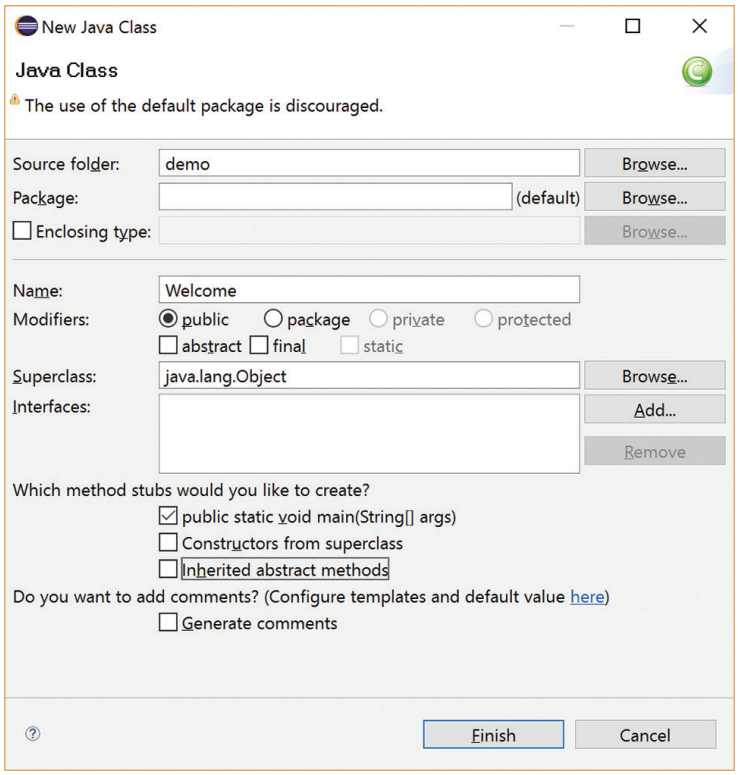


FIGURE 1.19 The New Java Class dialog box is used to create a new Java class. *Source*: Eclipse Foundation, Inc.

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. The *Run* command automatically compiles the program if the program has been changed.

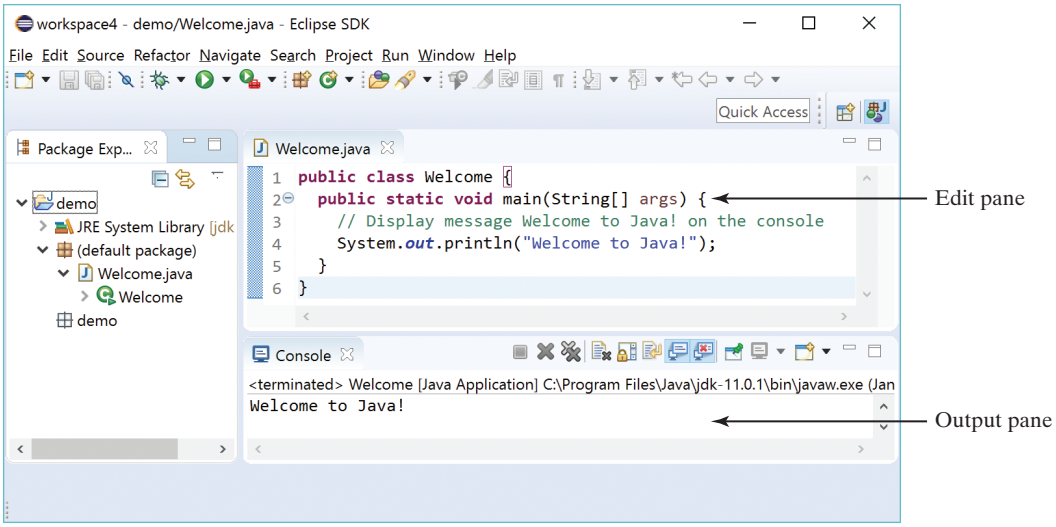


FIGURE 1.20 You can edit a program and run it in Eclipse. *Source*: Eclipse Foundation, Inc.

KEY TERMS

Application Program Interface (API) 33
assembler 29
assembly language 29
bit 25
block 35
block comment 35
bus 24
byte 25
bytecode 38
bytecode verifier 39
cable modem 28
central processing unit (CPU) 25
class loader 39
comment 35
compiler 30
console 34
dial-up modem 28
dot pitch 28
DSL (digital subscriber line) 28
encoding scheme 25
hardware 24
high-level language 30
integrated development environment (IDE) 34
interpreter 30
java *command* 39
Java Development Toolkit (JDK) 34
Java language specification 33
Java Runtime Environment (JRE) 34
Java Virtual Machine (JVM) 38
javac *command* 39
keyword 34
library 33
line comment 35
logic error 43
low-level language 30
machine language 29
main *method* 34
memory 26
motherboard 25
network interface card (NIC) 28
operating system (OS) 31
pixel 28
program 24
programming 24
runtime error 42
screen resolution 28
software 24

source code 30
source program 30
statement 30
statement terminator 34
storage devices 26
syntax error 41

Note
The above terms are defined in this chapter. Glossary (at the end of TOC) lists all the key terms and descriptions in the book, organized by chapters.

Supplement I.A

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 you can write a program once and run it on any computer.

22. The Java source file name must match the public class name in the program. Java source-code files must end with the `.java` extension.

23. Every class is compiled into a separate bytecode file that has the same name as the class and ends with the `.class` extension.

24. To compile a Java source-code file from the command line, use the `javac` command.

25. To run a Java class from the command line, use the `java` command.

26. 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*.

27. A block begins with an opening brace (`{`) and ends with a closing brace (`}`).

28. 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.

29. Every *statement* in Java ends with a semicolon (`;`), known as the *statement terminator*.

30. *Keywords* have a specific meaning to the compiler and cannot be used for other purposes in the program.

31. 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*. Comments are ignored by the compiler.

32. Java source programs are case sensitive.

33. Programming errors can be categorized into three types: *syntax errors*, *runtime errors*, and *logic errors*. Errors reported by a compiler are called syntax errors or *compile errors*. Runtime errors are errors that cause a program to terminate abnormally. Logic errors occur when a program does not perform the way it was intended to.

Quiz

Answer the quiz for this chapter at www.pearsonglobaleditions.com/liang. Choose this book and click Companion Website to select Quiz.

MyProgrammingLab™

Programming Exercises

Pedagogical Note

We cannot stress enough the importance of learning programming through exercises. For this reason, the book provides a large number of programming exercises at various levels of difficulty. The problems cover many application areas, including math, science, business, financial, gaming, animation, and multimedia. Solutions to most even-numbered programming exercises are on the Companion Website. Solutions to most odd-numbered programming exercises are on the Instructor Resource Website. The level of difficulty is rated easy (no star), moderate (*), hard (**), or challenging (***).

level of difficulty

1.1 (*Display three messages*) Write a program that displays `Welcome to Java`, `Learning Java Now`, and `Programming is fun`.

1.2 (*Display five messages*) Write a program that displays `I love Java` five times.

***1.3** (*Display a pattern*) Write a program that displays the following pattern:

```
   J
J aaa  v   vaaa
J  J aa vv  a  a
J aaaa   v   aaaa
```

1.4 (*Print a table*) Write a program that displays the following table:

```
aa^2a^3a^4
1  1   1   1
2  4   8   16
3  9   27  81
4 16  64 256
```

1.5 (*Compute expressions*) Write a program that displays the result of

$$\frac{7.5 \times 6.5 - 4.5 \times 3}{47.5 - 5.5}.$$

1.6 (*Summation of a series*) Write a program that displays the result of

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.$$

1.7 (*Approximate π*) π can be computed using the following formula:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots\right)$$

Write a program that displays the result of $4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11}\right)$ and $4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13}\right)$. Use `1.0` instead of `1` in your program.

1.8 (*Area and perimeter of a circle*) Write a program that displays the area and perimeter of a circle that has a radius of `6.5` using the following formula:

$$\pi = 3.14159$$

$$perimeter = 2 \times radius \times \pi$$

$$area = radius \times radius \times \pi$$

1.9 (*Area and perimeter of a rectangle*) Write a program that displays the area and perimeter of a rectangle with a width of `5.3` and height of `8.6` using the following formula:

$$area = width \times height$$

$$perimeter = 2 \times (width + height)$$

1.10 (*Average speed in miles*) Assume that a runner runs `15` kilometers in `50` minutes and `30` seconds. Write a program that displays the average speed in miles per hour. (Note that `1` mile is `1.6` kilometers.)

***1.11** (*Population projection*) The U.S. Census Bureau projects population based on the following assumptions:

- One birth every 7 seconds
- One death every 13 seconds
- One new immigrant every 45 seconds

Write a program to display the population for each of the next five years. Assume that the current population is 312,032,486, and one year has 365 days. *Hint*: In Java, if two integers perform division, the result is an integer. The fractional part is truncated. For example, `5 / 4` is `1` (not `1.25`) and `10 / 4` is `2` (not `2.5`). To get an accurate result with the fractional part, one of the values involved in the division must be a number with a decimal point. For example, `5.0 / 4` is `1.25` and `10 / 4.0` is `2.5`.

1.12 (*Average speed in kilometers*) Assume that a runner runs `24` miles in `1` hour, `40` minutes, and `35` seconds. Write a program that displays the average speed in kilometers per hour. (Note `1` mile is equal to `1.6` kilometers.)

***1.13** (*Algebra: solve* 2×2 *linear equations*) You can use Cramer's rule to solve the following 2×2 system of linear equation provided that $ad - bc$ is not 0:

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \quad x = \frac{ed - bf}{ad - bc} \quad y = \frac{af - ec}{ad - bc}$$

Write a program that solves the following equation and displays the value for x and y: (Hint: replace the symbols in the formula with numbers to compute x and y. This exercise can be done in Chapter 1 without using materials in later chapters.)

$$\begin{aligned} 3.4x + 50.2y &= 44.5 \\ 2.1x + .55y &= 5.9 \end{aligned}$$

Note

More than 200 additional programming exercises with solutions are provided to the instructors on the Instructor Resource Website.