

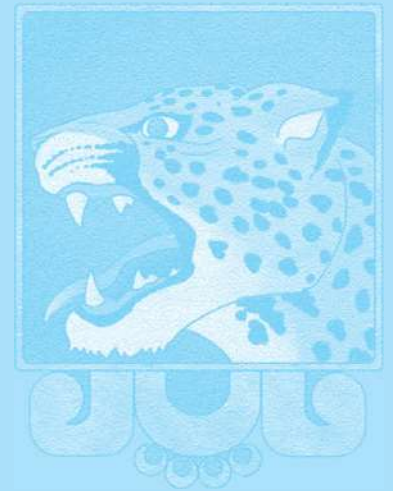
CHAPTER

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, 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 display output using the **JOptionPane** message dialog boxes (§1.9).
- To become familiar with Java programming style and documentation (§1.10).
- To explain the differences between syntax errors, runtime errors, and logic errors (§1.11).



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 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 the review in 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

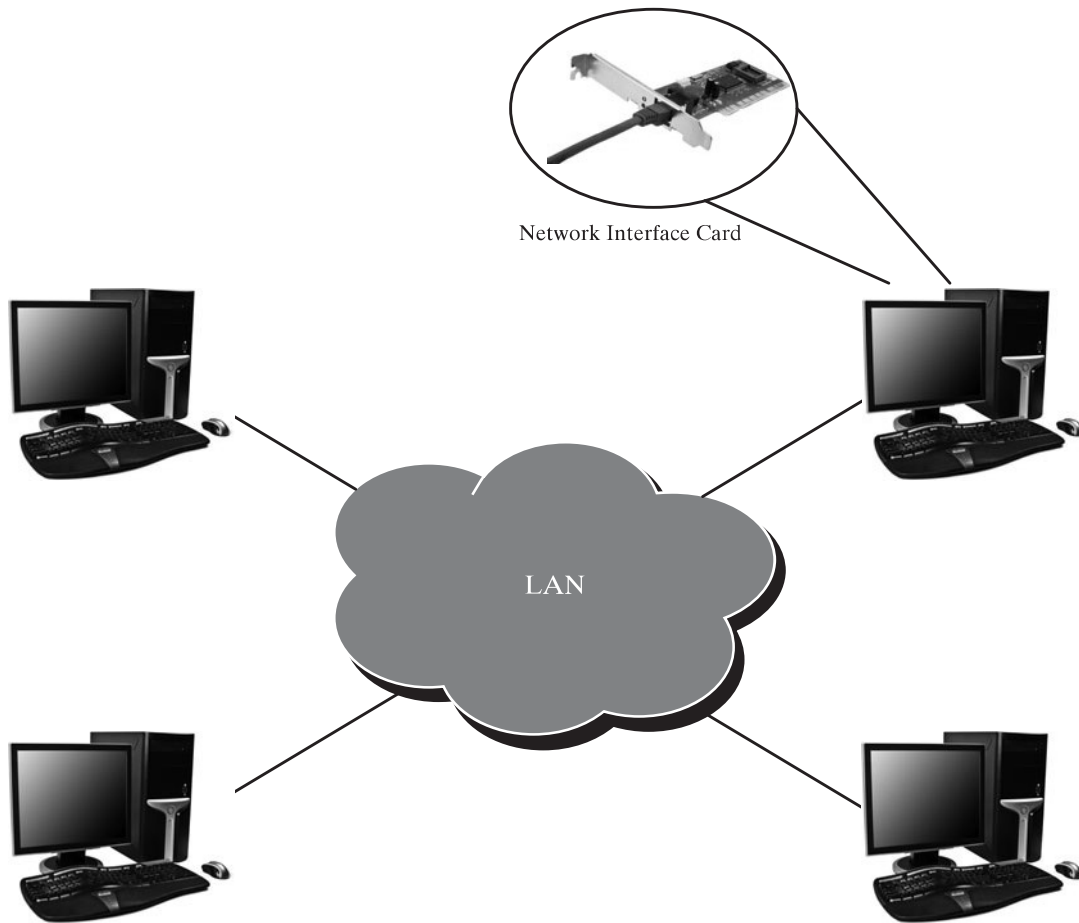


FIGURE 1.7 A local area network connects computers in close proximity to each other.

- 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 a language the computer can understand.

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

machine language

MyProgrammingLab™

- 1.12** What is an assembler?
- 1.13** What is a high-level programming language?
- 1.14** What is a source program?
- 1.15** What is an interpreter?
- 1.16** What is a compiler?
- 1.17** What is the difference between an interpreted language and a compiled language?

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.

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 1.10 shows the interrelationship of hardware, operating system, application software, and the user.

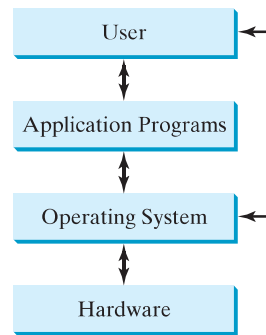


FIGURE 1.10 Users and applications access the computer's hardware via the operating system.

The major tasks of an operating system are:

- 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 do not 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 such techniques as *multiprogramming*, *multithreading*, and *multiprocessing* to increase system performance.

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.

multiprogramming

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.

multithreading

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.

multiprocessing

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?



MyProgrammingLab™

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 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.

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.

Java language specification

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 java.sun.com/docs/books/jls.

API

library

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 www.oracle.com/technetwork/java/index.html.

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 standalone applications or applets.
- *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 SE 7, is used in this book. Oracle releases each version with a *Java Development Toolkit (JDK)*. For Java SE 7, the Java Development Toolkit is called *JDK 1.7* (also known as *Java 7* or *JDK 7*).

Java Development Toolkit (JDK)

JDK 1.7 = JDK 7

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.

Integrated development environment



MyProgrammingLab™

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.

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

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

class
main method
compute expression

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.

- I.28** What is a keyword? List some Java keywords.
- I.29** Is Java case sensitive? What is the case for Java keywords?
- I.30** What is a comment? Is the comment ignored by the compiler? How do you denote a comment line and a comment paragraph?
- I.31** What is the statement to display a string on the console?
- I.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);
    }
}
```



MyProgrammingLab™

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 1.14. 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. If you wish to use an *IDE* such as Eclipse, NetBeans, or TextPad, refer to Supplement II for tutorials. From the command window, you can use a text editor such as Notepad to create the Java source-code file, as shown in Figure 1.15.



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`.



VideoNote
Eclipse brief tutorial

command window
IDE Supplements



VideoNote
NetBeans brief tutorial

file name

22 Chapter I Introduction to Computers, Programs, and Java

java ClassName



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**.

NoClassDefFoundError

NoSuchMethodError



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.

class loader

bytecode verifier



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 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 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

MyProgrammingLab™

- 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?



Key
Point

JOptionPane

showMessageDialog

I.9 Displaying Text in a Message Dialog Box

You can display text in a graphical dialog box.


The program in Listing 1.1 displays the text on the console, as shown in Figure 1.17. You can rewrite the program to display the text in a message dialog box. To do so, you need to use the **showMessageDialog** method in the **JOptionPane** class. **JOptionPane** is one of the many predefined classes in the Java library that you can reuse rather than “reinvent the wheel.” You can use the **showMessageDialog** method to display any text in a message dialog box, as shown in Figure 1.18. The new program is given in Listing 1.4.

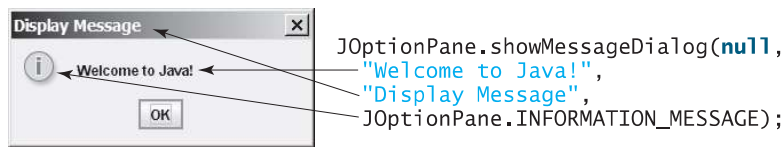
LISTING 1.4 WelcomeInMessageDialogBox.java

block comment

```
1  /* This application program displays Welcome to Java!
2   *   in a message dialog box.
3   */
```


24 Chapter 1 Introduction to Computers, Programs, and Java

where **x** is a string for the text to be displayed, and **y** is a string for the title of the message box. The fourth argument can be `JOptionPane.INFORMATION_MESSAGE`, which causes the information icon () to be displayed in the message box, as shown in the following example.



Note

There are two types of **import** statements: *specific import* and *wildcard import*. The *specific import* specifies a single class in the import statement. For example, the following statement imports `JOptionPane` from the package `javax.swing`.

```
import javax.swing.JOptionPane;
```

The *wildcard import* imports all the classes in a package by using the asterisk as the wildcard. For example, the following statement imports all the classes from the package `javax.swing`.

```
import javax.swing.*;
```

The information for the classes in an imported package is not read in at compile time or runtime unless the class is used in the program. The import statement simply tells the compiler where to locate the classes. There is no performance difference between a specific import and a wildcard import declaration.



Note

Recall that you have used the `System` class in the statement `System.out.println("Welcome to Java");` in Listing 1.1. The `System` class is not imported because it is in the `java.lang` package. All the classes in the `java.lang` package are *implicitly* imported in every Java program.



Check Point

MyProgrammingLab™

- 1.41 What is the statement to display the message “Hello world” in a message dialog box?
- 1.42 Why does the `System` class not need to be imported?
- 1.43 Are there any performance differences between the following two **import** statements?

```
import javax.swing.JOptionPane;
import javax.swing.*;
```



Key Point

1.10 Programming Style and Documentation

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

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



MyProgrammingLab™

I.44 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");
    }
}
```

1.11 Programming Errors



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

1.11.1 Syntax Errors

syntax errors
compile 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 1.5 has a syntax error, as shown in Figure 1.19.

LISTING 1.5 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.

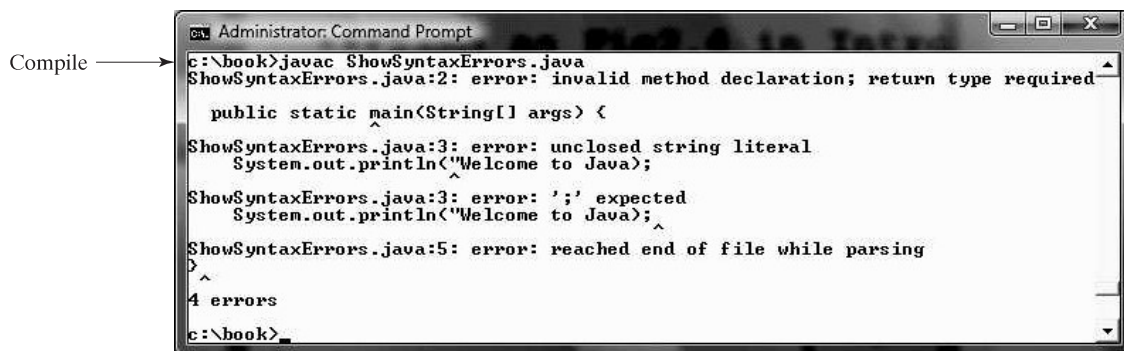


FIGURE 1.19 The compiler reports syntax errors.