Chapter 1 Introduction

<div style="border:1px solid">

CHAPTER GOALS

- To understand the activity of programming

- To learn about the architecture of computers

- To learn about machine code and high-level programming languages

- To become familiar with your computing environment and your compiler

- To compile and run your first Java program

- To recognize syntax and logic errors

</div>

The purpose of this chapter is to familiarize you with the concept of programming. It reviews the architecture of a computer and discusses the difference between machine code and high-level programming languages. Finally, you will see how to compile and run your first Java program, and how to diagnose errors that may occur when a program is compiled or executed.

2

1.1 What Is Programming?

You have probably used a computer for work or fun. Many people use computers for everyday tasks such as balancing a checkbook or writing a term paper. Computers are good for such tasks. They can handle repetitive chores, such as totaling up numbers or placing words on a page, without getting bored or exhausted. Computers also make good game machines because they can play sequences of sounds and pictures, involving the human user in the process.

The flexibility of a computer is quite an amazing phenomenon. The same machine can balance your checkbook, print your term paper, and play a game. In contrast, other machines carry out a much narrower range of tasks—a car drives and a toaster toasts.

To achieve this flexibility, the computer must be *programmed* to perform each task. A computer itself is a machine that stores data (numbers, words, pictures), interacts with devices (the monitor screen, the sound system, the printer), and executes programs.

Programs are sequences of instructions and decisions that the computer carries out to achieve a task. One program balances checkbooks; a different program, perhaps designed and constructed by a different company, processes words; and a third program, probably from yet another company, plays a game.

> A computer must be programmed to perform tasks. Different tasks require different programs.

Today's computer programs are so sophisticated that it is hard to believe that they are all composed of extremely primitive operations.

> A computer program executes a sequence of very basic operations in rapid succession.

A typical operation may be one of the following:

- Put a red dot onto this screen position.

- Send the letter A to the printer.

- Get a number from this location in memory.

- Add up two numbers.

- If this value is negative, continue the program at that instruction.

A computer program tells a computer, in minute detail, the sequence of steps that are needed to complete a task. A program contains a huge number of simple operations, and the computer executes them at great speed. The computer has no intelligence—it simply executes instruction sequences that have been prepared in advance.

> A computer program contains the instruction sequences for all tasks that it can execute.

To use a computer, no knowledge of programming is required. When you write a term paper with a word processor, that software package has been programmed by the manufacturer and is ready for you to use. That is only to be expected—you can drive a car without being a mechanic and toast bread without being an electrician.

A primary purpose of this book is to teach you how to design and implement computer programs. You will learn how to formulate instructions for all tasks that your programs need to execute.

Keep in mind that programming a sophisticated computer game or word processor requires a team of many highly skilled programmers, graphic artists, and other professionals. Your first programming efforts will be more mundane. The concepts and skills you learn in this book form an important foundation, but you should not expect to immediately produce professional software. A typical college program in computer science or software engineering takes four years to complete; this book is intended as an introductory course in such a program.

Many students find that there is an immense thrill even in simple programming tasks. It is an amazing experience to see the computer carry out a task precisely and quickly that would take you hours of drudgery.

SELF CHECK

1. What is required to play a music CD on a computer?

2. Why is a CD player less flexible than a computer?

3. Can a computer program develop the initiative to execute tasks in a better way than its programmers envisioned?

1.2 The Anatomy of a Computer

To understand the programming process, you need to have a rudimentary understanding of the building blocks that make up a computer. This section will describe a personal computer. Larger computers have faster, larger, or more powerful components, but they have fundamentally the same design.

Figure 1

Central Processing Unit

At the heart of the computer lies the *central processing unit* (CPU) (see Figure 1). It consists of a single *chip* (integrated circuit) or a small number of chips. A computer chip is a component with a plastic or metal housing, metal connectors, and inside wiring made principally from silicon. For a CPU chip, the inside wiring is enormously complicated. For example, the Pentium 4 chip (a popular CPU for personal computers at the time of this writing) contains over 50 million structural elements called *transistors*—the elements that enable electrical signals to control other electrical signals, making automatic computing possible. The CPU locates and executes the program instructions; it carries out arithmetic operations such as addition, subtraction, multiplication, and division; and it fetches data from storage and input/output devices and sends data back.

At the heart of the computer lies the central processing unit (CPU).

The computer keeps data and programs in *storage*. There are two kinds of storage. *Primary storage*, also called *random-access memory* (RAM) or simply *memory*, is fast but expensive; it is made from memory chips (see Figure 2). Primary storage has two disadvantages. It is comparatively expensive, and it loses all its data when the power is turned off. *Secondary storage*, usually a *hard disk* (see Figure 3), provides less

expensive storage that persists without electricity. A hard disk consists of rotating platters, which are coated with a magnetic material, and read/write heads, which can detect and change the patterns of varying magnetic flux on the platters. This is essentially the same recording and playback process that is used in audio or video tapes.

> Data and programs are stored in primary storage (memory) and secondary storage (such as a hard disk).

Some computers are self-contained units, whereas others are interconnected through *networks*. Home computers are usually intermittently connected to the Internet via a dialup or broadband connection. The computers in your computer lab are probably permanently connected to a local area network. Through the network cabling, the computer can read programs from central storage locations or send data to other computers. For the user of a networked computer, it may not even be obvious which data reside on the computer itself and which are transmitted through the network.
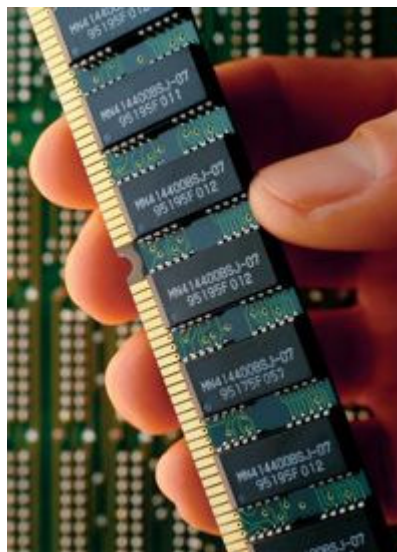
Figure 2

A Memory Module with Memory Chips

Most computers have *removable storage* devices that can access data or programs on media such as floppy disks, tapes, or compact discs (CDs).
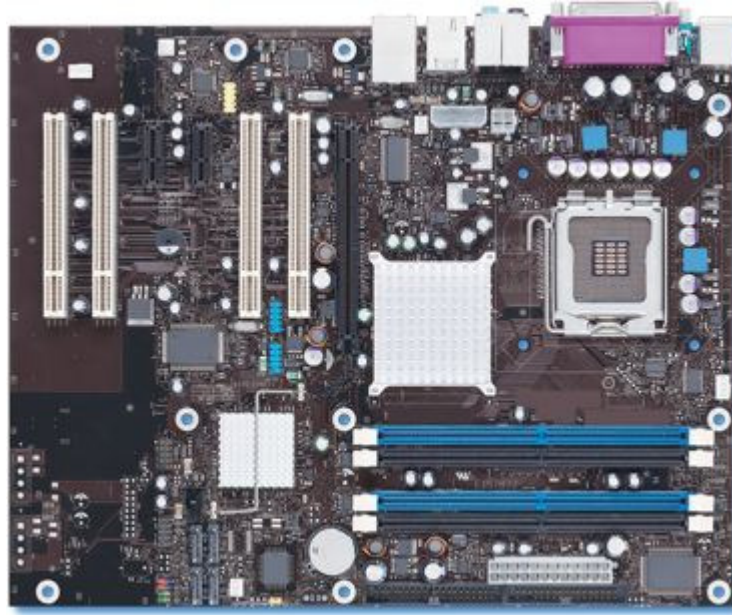
Figure 3

A Hard Disk.

5

Figure 4

A Motherboard

To interact with a human user, a computer requires other peripheral devices. The computer transmits information to the user through a display screen, loudspeakers, and printers. The user can enter information and directions to the computer by using a keyboard or a pointing device such as a mouse.

The CPU, the RAM, and the electronics controlling the hard disk and other devices are interconnected through a set of electrical lines called a *bus*. Data travel along the bus from the system memory and peripheral devices to the CPU and back. Figure 4 shows a *motherboard*, which contains the CPU, the RAM, and connectors to peripheral devices.

Figure 5 gives a schematic overview of the architecture of a computer. Program instructions and data (such as text, numbers, audio, or video) are stored on the hard disk, on a CD, or on a network. When a program is started, it is brought into memory where it can be read by the CPU. The CPU reads the program one instruction at a time. As directed by these instructions, the CPU reads data, modifies it, and writes it back to RAM or to secondary storage. Some program instructions will cause the CPU to

interact with the devices that control the display screen or the speaker. Because these actions happen many times over and at great speed, the human user will perceive images and sound. Similarly, the CPU can send instructions to a printer to mark the paper with patterns of closely spaced dots, which a human recognizes as text characters and pictures. Some program instructions read user input from the keyboard or mouse. The program analyzes the nature of these inputs and then executes the next appropriate instructions.

> The CPU reads machine instructions from memory. The instructions direct it to communicate with memory, secondary storage, and peripheral devices.

6
7

Figure 5
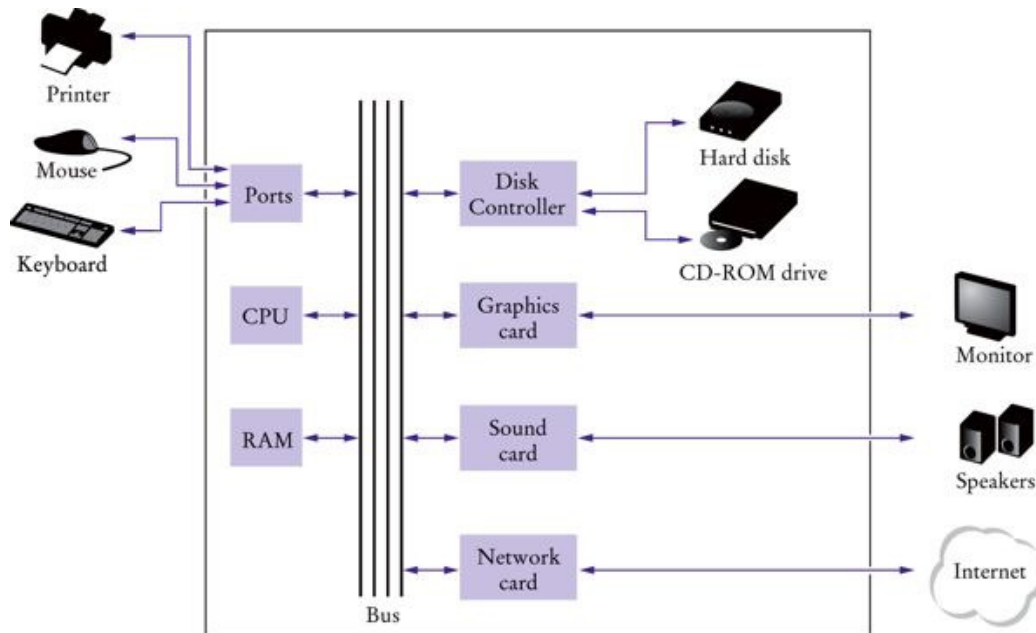
Schematic Diagram of a Computer

SELF CHECK

4. Where is a program stored when it is not currently running?

5. Which part of the computer carries out arithmetic operations, such as addition and multiplication?

RANDOM FACT 1.1: The ENIAC and the Dawn of Computing

The ENIAC (*e*lectronic *n*umerical *i*ntegrator *a*nd *c*omputer) was the first usable electronic computer. It was designed by J. Presper Eckert and John Mauchly at the University of Pennsylvania and was completed in 1946. Instead of transistors, which were not invented until two years after it was built, the ENIAC contained about 18,000 *vacuum tubes* in many cabinets housed in a large room (see The ENIAC figure). Vacuum tubes burned out at the rate of several tubes per day. An attendant with a shopping cart full of tubes constantly made the rounds and replaced defective ones. The computer was programmed by connecting wires on panels. Each wiring configuration would set up the computer for a particular problem. To have the computer work on a different problem, the wires had to be replugged.

Work on the ENIAC was supported by the U.S. Navy, which was interested in computations of ballistic tables that would give the trajectory of a projectile, depending on the wind resistance, initial velocity, and atmospheric conditions. To compute the trajectories, one must find the numerical solutions of certain differential equations; hence the name "numerical integrator". Before machines like ENIAC were developed, humans did this kind of work, and until the 1950s the word "computer" referred to these people. The ENIAC was later used for peaceful purposes, such as the tabulation of U.S. census data.
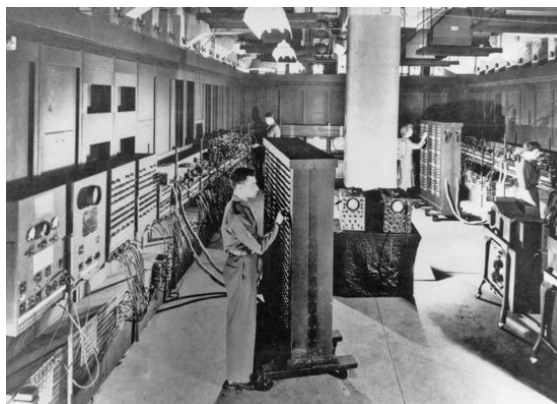
The ENIAC

1.3 Translating Human-Readable Programs to Machine Code

On the most basic level, computer instructions are extremely primitive. The processor executes *machine instructions*. CPUs from different vendors, such as the Intel Pentium or the Sun SPARC, have different sets of machine instructions. To enable Java applications to run on multiple CPUs without modification, Java programs contain machine instructions for a so-called "Java virtual machine" (JVM), an idealized CPU that is simulated by a program run on the actual CPU. The difference between actual and virtual machine instructions is not important—all you need to know is that machine instructions are very simple, are encoded as numbers and stored in memory, and can be executed very quickly.

> Generally, machine code depends on the CPU type. However, the instruction set of the Java virtual machine (JVM) can be executed on many CPUs.

8
9

A typical sequence of machine instructions is

1. Load the contents of memory location 40.

2. Load the value 100.

3. If the first value is greater than the second value, continue with the instruction that is stored in memory location 240.

Actually, machine instructions are encoded as numbers so that they can be stored in memory. On the Java virtual machine, this sequence of instruction is encoded as the sequence of numbers

```
21 40
16 100
163 240
```

When the virtual machine fetches this sequence of numbers, it decodes them and executes the associated sequence of commands.

How can you communicate the command sequence to the computer? The most direct method is to place the actual numbers into the computer memory. This is, in fact, how

the very earliest computers worked. However, a long program is composed of thousands of individual commands, and it is tedious and error-prone to look up the numeric codes for all commands and manually place the codes into memory. As we said before, computers are really good at automating tedious and error-prone activities, and it did not take long for computer programmers to realize that computers could be harnessed to help in the programming process.

> Because machine instructions are encoded as numbers, it is difficult to write programs in machine code.

In the mid-1950s, *high-level* programming languages began to appear. In these languages, the programmer expresses the idea behind the task that needs to be performed, and a special computer program, called a *compiler*, translates the high-level description into machine instructions for a particular processor.

> High-level languages allow you to describe tasks at a higher conceptual level than machine code.

For example, in Java, the high-level programming language that you will use in this book, you might give the following instruction:

```
if (intRate > 100)
    System.out.println("Interest rate error");
```

This means, "If the interest rate is over 100, display an error message". It is then the job of the compiler program to look at the sequence of characters `if (intRate > 100)` and translate that into

```
21 40 16 100 163 240 . . .
```

Compilers are quite sophisticated programs. They translate logical statements, such as the `if` statement, into sequences of computations, tests, and jumps. They assign memory locations for *variables*—items of information identified by symbolic names—like `intRate`. In this course, we will generally take the existence of a compiler for granted. If you decide to become a professional computer scientist, you may well learn more about compiler-writing techniques later in your studies.

A compiler translates programs written in a high-level language into machine code.

9

10

SELF CHECK

6. What is the code for the Java virtual machine instruction "Load the contents of memory location 100"?

7. Does a person who uses a computer for office work ever run a compiler?

1.4 The Java Programming Language

In 1991, a group led by James Gosling and Patrick Naughton at Sun Microsystems designed a programming language that they code-named "Green" for use in consumer devices, such as intelligent television "set-top" boxes. The language was designed to be simple and architecture neutral, so that it could be executed on a variety of hardware. No customer was ever found for this technology.

Java was originally designed for programming consumer devices, but it was first successfully used to write Internet applets.

Gosling recounts that in 1994 the team realized, "We could write a really cool browser. It was one of the few things in the client/server mainstream that needed some of the weird things we'd done: architecture neutral, real-time, reliable, secure". Java was introduced to an enthusiastic crowd at the SunWorld exhibition in 1995.

Since then, Java has grown at a phenomenal rate. Programmers have embraced the language because it is simpler than its closest rival, C++. In addition, Java has a rich *library* that makes it possible to write portable programs that can bypass proprietary operating systems—a feature that was eagerly sought by those who wanted to be independent of those proprietary systems and was bitterly fought by their vendors. A "micro edition" and an "enterprise edition" of the Java library make Java programmers at home on hardware ranging from smart cards and cell phones to the largest Internet servers.

> Java was designed to be safe and portable, benefiting both Internet users and students.

Because Java was designed for the Internet, it has two attributes that make it very suitable for beginners: safety and portability. If you visit a web page that contains Java code (so-called *applets*—see Figure 6 for an example), the code automatically starts running. It is important that you can trust that applets are inherently safe. If an applet could do something evil, such as damaging data or reading personal information on your computer, then you would be in real danger every time you browsed the Web— an unscrupulous designer might put up a web page containing dangerous code that would execute on your machine as soon as you visited the page. The Java language has an assortment of security features that guarantees that no evil applets can run on your computer. As an added benefit, these features also help you to learn the language faster. The Java virtual machine can catch many kinds of beginners' mistakes and report them accurately. (In contrast, many beginners' mistakes in the C++ language merely produce programs that act in random and confusing ways.) The other benefit of Java is portability. The same Java program will run, without change, on Windows, UNIX, Linux, or the Macintosh. This too is a requirement for applets. When you visit a web page, the web server that serves up the page contents has no idea what computer you are using to browse the Web. It simply returns you the portable code that was generated by the Java compiler. The virtual machine on your computer executes that portable code. Again, there is a benefit for the student. You do not have to learn how to write programs for different operating systems.

Figure 6
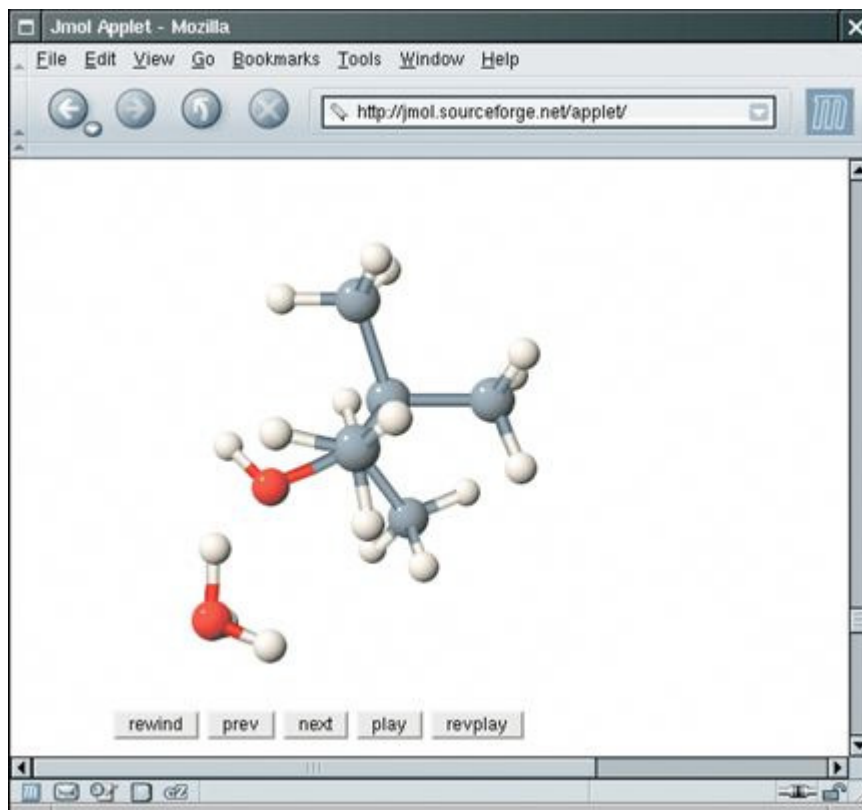
An Applet for Visualizing Molecules ([1])

At this time, Java is firmly established as one of the most important languages for general-purpose programming as well as for computer science instruction. However, although Java is a good language for beginners, it is not perfect, for three reasons.

Because Java was not specifically designed for students, no thought was given to making it really simple to write basic programs. A certain amount of technical machinery is necessary in Java to write even the simplest programs. This is not a problem for professional programmers, but it is a drawback for beginning students. As you learn how to program in Java, there will be times when you will be asked to be satisfied with a preliminary explanation and wait for complete details in a later chapter.

Java was revised and extended many times during its life—see Table 1. In this book, we assume that you have Java version 5 or later.

Finally, you cannot hope to learn all of Java in one semester. The Java language itself is relatively simple, but Java contains a vast set of *library packages* that are required to write useful programs. There are packages for graphics, user interface design, cryptography, networking, sound, database storage, and many other purposes. Even expert Java programmers cannot hope to know the contents of all of the packages— they just use those that they need for particular projects.

> Java has a very large library. Focus on learning those parts of the library that you need for your programming projects.

Using this book, you should expect to learn a good deal about the Java language and about the most important packages. Keep in mind that the central goal of this book is not to make you memorize Java minutiae, but to teach you how to think about programming.

Table 1 Java Versions

Version	Year	Important New Features
1.0	1996	
1.1	1997	Inner classes
1.2	1998	Swing, Collections
1.3	2000	Performance enhancements
1.4	2002	Assertions, XML
5	2004	Generic classes, enhanced `for` loop, auto-boxing, enumerations
6	2006	Library improvements

> ### SELF CHECK
>
> **8.** What are the two most important benefits of the Java language?
>
> **9.** How long does it take to learn the entire Java library?

## 1.5 Becoming Familiar with Your Computer

You may be taking your first programming course as you read this book, and you may well be doing your work on an unfamiliar computer system. Spend some time familiarizing yourself with the computer. Because computer systems vary widely, this book can only give an outline of the steps you need to follow. Using a new and unfamiliar computer system can be frustrating, especially if you are on your own. Look for training courses that your campus offers, or ask a friend to give you a brief tour.

> Set aside some time to become familiar with the computer system and the Java compiler that you will use for your class work.
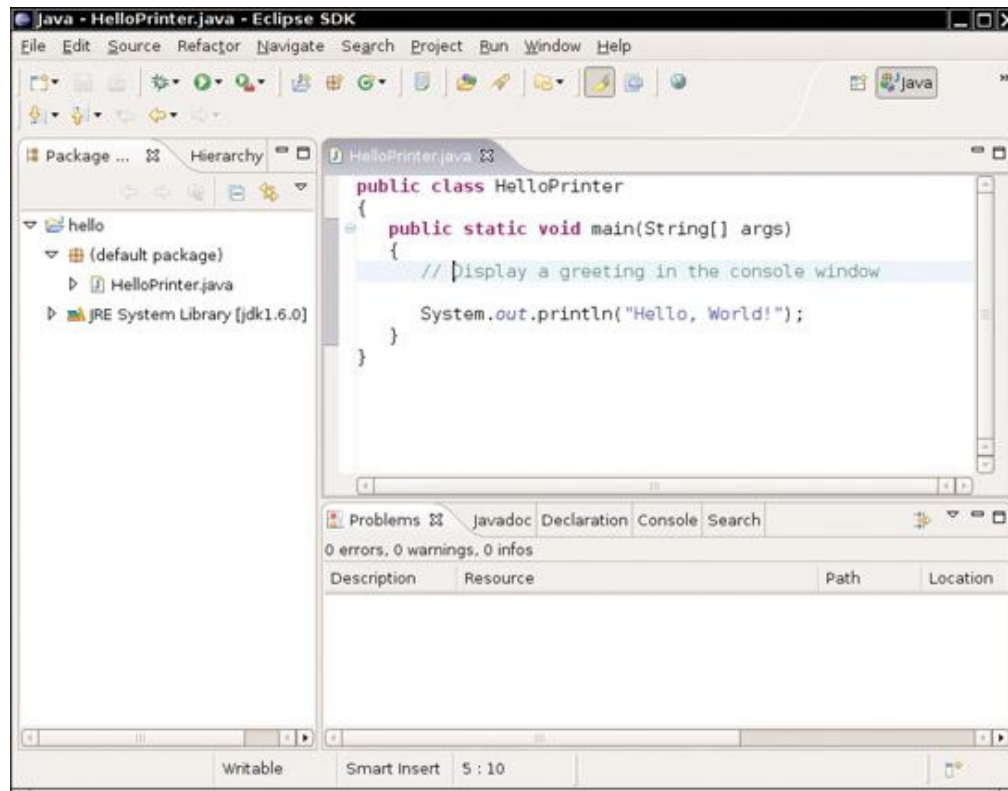
12

13

Figure 7
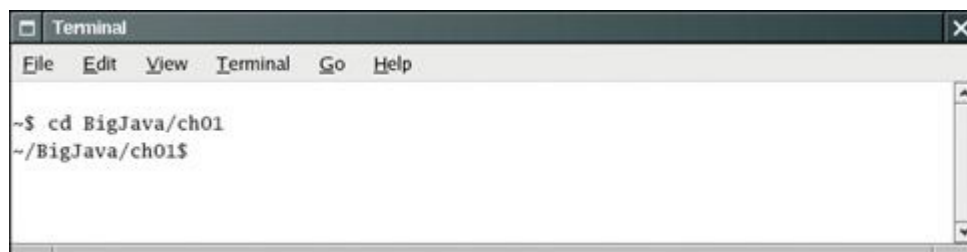
A Shell Window

Step 1. Log In

If you use your home computer, you probably don't need to worry about this step. Computers in a lab, however, are usually not open to everyone. You may need an account name or number and a password to gain access to such a system.

Step 2. Locate the Java Compiler

Figure 8

An Integrated Development Environment

Computer systems differ greatly in this regard. On some systems you must open a *shell window* (see [Figure 7](#)) and type commands to launch the compiler. Other systems have an *integrated development environment* in which you can write and test your programs (see [Figure 8](#)). Many university labs have information sheets and tutorials that walk you through the tools that are installed in the lab. Instructions for several popular compilers are available in WileyPLUS.

13

14

Step 3. Understand Files and Folders

As a programmer, you will write Java programs, try them out, and improve them. Your programs are kept in *files*. A file is a collection of items of information that are kept together, such as the text of a word-processing document or the instructions of a Java program. Files have names, and the rules for legal names differ from one system to another. Some systems allow spaces in file names; others don't. Some distinguish between upper- and lowercase letters; others don't. Most Java compilers require that Java files end in an *extension*—`.java`; for example, `Test.java`. Java file names cannot contain spaces, and the distinction between upper- and lowercase letters is important.

Figure 9
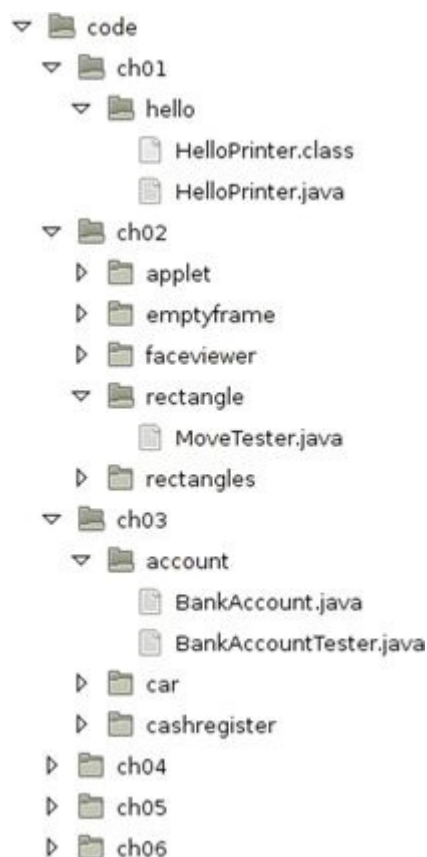
Nested Folders

Files are stored in *folders* or *directories*. These file containers can be *nested*. That is, a folder can contain not only files but also other folders, which themselves can contain more files and folders (see [Figure 9](#)). This hierarchy can be quite large, especially on networked computers, where some of the files may be on your local disk, others elsewhere on the network. While you need not be concerned with every branch of the hierarchy, you should familiarize yourself with your local environment. Different systems have different ways of showing files and directories. Some use a graphical display and let you move around by clicking the mouse on folder icons. In other systems, you must enter commands to visit or inspect different locations.

14

15

Step 4. Write a Simple Program

In the next section, we will introduce a very simple program. You will need to learn how to type it in, how to run it, and how to fix mistakes.

Step 5. Save Your Work

You will spend many hours typing Java program code and improving it. The resulting program files have some value, and you should treat them as you would other important property. A conscientious safety strategy is particularly important for computer files. They are more fragile than paper documents or other more tangible objects. It is easy to delete a file accidentally, and occasionally files are lost because of a computer malfunction. Unless you keep a copy, you must then retype the contents. Because you probably won't remember the entire file, you will likely find yourself spending almost as much time as you did to enter and improve it in the first place. This costs time, and it may cause you to miss deadlines. It is therefore crucial that you learn how to safeguard files and that you get in the habit of doing so *before* disaster strikes. You can make safety or *backup* copies of files by saving copies on a floppy or CD, into another folder, to your local area network, or on the Internet.

> Develop a strategy for keeping backup copies of your work before disaster strikes.

10. How are programming projects stored on a computer?

11. What do you do to protect yourself from data loss when you work on programming projects?

🔖 **PRODUCTIVITY HINT 1.1**: **Understand the File System**

In recent years, computers have become easier to use for home or office users. Many inessential details are now hidden from casual users. For example, many casual users simply place all their work inside a default folder (such as "Home" or "My Documents") and are blissfully ignorant about details of the file system.

But you need to know how to impose an organization on the data that you create. You also need to be able to locate and inspect files that are required for translating and running Java programs.

15

16

If you are not comfortable with files and folders, be sure to set aside some time to learn about these concepts. Enroll in a short course, or take a web tutorial. Many free tutorials are available on the Internet, but unfortunately their locations change frequently. Search the Web for "files and folders tutorial" and pick a tutorial that goes beyond the basics.

🔖 **PRODUCTIVITY HINT 1.2**: **Have a Backup Strategy**

Come up with a strategy for your backups *now*, before you lose any data. Here are a few pointers to keep in mind.

- *Select a backup medium.* Floppy disks are the traditional choice, but they can be unreliable. CD media are more reliable and hold far more information, but they are more expensive. An increasingly popular form of backup is Internet file storage. Many people use two levels of backup: a folder on the hard disk for quick and dirty backups, and a CD-ROM for higher security. (After all, a hard disk can crash—a particularly common problem with laptop computers.)

- *Back up often.* Backing up a file takes only a few seconds, and you will hate yourself if you have to spend many hours recreating work that you easily could have saved.

- *Rotate backups.* Use more than one set of disks or folders for backups, and rotate them. That is, first back up onto the first backup destination, then to the second and third, and then go back to the first. That way you always have three recent backups. Even if one of the floppy disks has a defect, or you messed up one of the backup directories, you can use one of the others.

- *Back up source files only.* The compiler translates the files that you write into files consisting of machine code. There is no need to back up the machine code files, because you can recreate them easily by running the compiler again. Focus your backup activity on those files that represent your effort. That way your backups won't fill up with files that you don't need.

- *Pay attention to the backup direction.* Backing up involves copying files from one place to another. It is important that you do this right—that is, copy from your work location to the backup location. If you do it the wrong way, you will overwrite a newer file with an older version.

- *Check your backups once in a while.* Double-check that your backups are where you think they are. There is nothing more frustrating than finding out that the backups are not there when you need them. This is particularly true if you use a backup program that stores files on an unfamiliar device (such as data tape) or in a compressed format.

- *Relax before restoring.* When you lose a file and need to restore it from backup, you are likely to be in an unhappy, nervous state. Take a deep breath and think through the recovery process before you start. It is not uncommon for an agitated computer user to wipe out the last backup when trying to restore a damaged file.

16

Java Concepts, 5th Edition

1.6 Compiling a Simple Program

You are now ready to write and run your first Java program. The traditional choice for the very first program in a new programming language is a program that displays a simple greeting: "Hello, World!". Let us follow that tradition. Here is the "Hello, World!" program in Java.

ch01/hello/HelloPrinter.java

```
1 public class HelloPrinter
2 {
3    public static void main(String[] args)
4    {
5    // Display a greeting in the console window
6
7        System.out.println ("Hello, World!");
8    }
9 }
```

Output

```
   Hello, World!
```

We will examine this program in a minute. For now, you should make a new program file and call it `HelloPrinter.java`. Enter the program instructions and compile and run the program, following the procedure that is appropriate for your compiler.

Java is *case sensitive*. You must enter upper- and lowercase letters exactly as they appear in the program listing. You cannot type `MAIN` or `PrintLn`. If you are not careful, you will run into problems—see Common Error 1.2.

Java is case sensitive. You must be careful about distinguishing between upper- and lowercase letters.

On the other hand, Java has *free-form layout*. You can use any number of spaces and line breaks to separate words. You can cram as many words as possible into each line,

```
    public class HelloPrinter{public static void
    main(String[]
```

```
args){// Display a greeting in the console window
System.out.println("Hello, World!");}}
```

You can even write every word and symbol on a separate line,

```
public
class
HelloPrinter
{
public
static
void
main
(
. . .
```

17

18

However, good taste dictates that you lay out your programs in a readable fashion. We will give you recommendations for good layout throughout this book. Appendix A contains a summary of our recommendations.

> Lay out your programs so that they are easy to read.

When you run the test program, the message

```
Hello, World!
```

will appear somewhere on the screen (see Figures 10 and 11). The exact location depends on your programming environment.

Now that you have seen the program working, it is time to understand its makeup. The first line,

```
public class HelloPrinter
```

starts a new *class*. Classes are a fundamental concept in Java, and you will begin to study them in Chapter 2. In Java, every program consists of one or more classes.

> Classes are the fundamental building blocks of Java programs.

The keyword `public` denotes that the class is usable by the "public". You will later encounter `private` features. At this point, you should simply regard the

```
public class ClassName
{
        . . .
{
```

as a necessary part of the "plumbing" that is required to write any Java program. In Java, every source file can contain at most one public class, and the name of the public class must match the name of the file containing the class. For example, the class HelloPrinter *must* be contained in a file HelloPrinter.java. It is very important that the names *and the capitalization* match exactly. You can get strange error messages if you call the class HELLOPrinter or the file helloprinter.java.
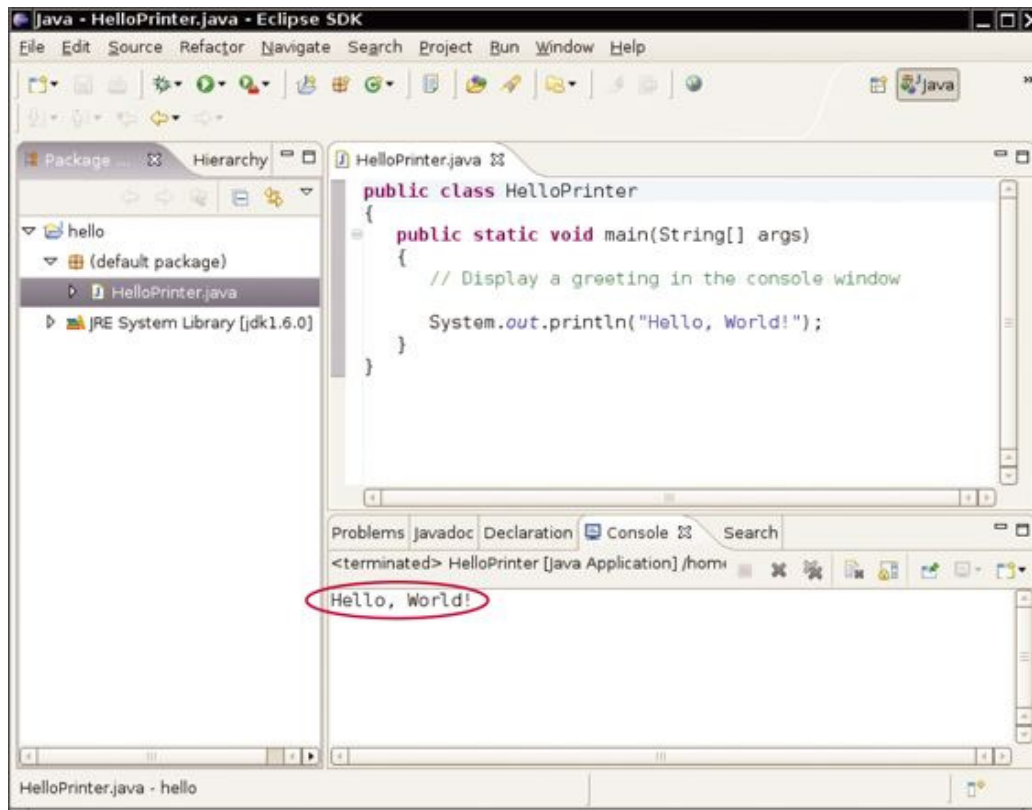
Figure 10

Running the HelloPrinter Program in a Console Window

18

Figure 11

Running the `HelloPrinter` Program in an Integrated Development Environment

The construction

```
public static void main(String[] args)
{
}
```

defines a *method* called `main`. A method contains a collection of programming instructions that describe how to carry out a particular task. Every Java application must have a `main` method. Most Java programs contain other methods besides `main`, and you will see in Chapter 3 how to write other methods.

> Every Java application contains a class with a `main` method. When the application starts, the instructions in the `main` method are executed.

The *parameter* `String[] args` is a required part of the `main` method. (It contains *command line arguments*, which we will not discuss until Chapter 11.) The keyword `static` indicates that the `main` method does not operate on an *object*. (As you will see in Chapter 2, most methods in Java do operate on objects, and `static` methods are not common in large Java programs. Nevertheless, `main` must always be `static`, because it starts running before the program can create objects.)

> Each class contains definitions of methods. Each method contains a sequence of instructions.

19

20

At this time, simply consider

```
public class ClassName
{
    public static void main(String[] args)
    {
        . . .
    }
}
```

as yet another part of the "plumbing". Our first program has all instructions inside the `main` method of a class.

The first line inside the `main` method is a *comment*

    // Display a greeting in the console window

This comment is purely for the benefit of the human reader, to explain in more detail what the next statement does. Any text enclosed between `//` and the end of the line is completely ignored by the compiler. Comments are used to explain the program to other programmers or to yourself.

> Use comments to help human readers understand your program.

The instructions or *statements* in the *body* of the `main` method—that is, the statements inside the curly braces (`{}`)—are executed one by one. Each statement ends in a semicolon (`;`). Our method has a single statement:

```
System.out.println("Hello, World!");
```

This statement prints a line of text, namely "Hello, World!". However, there are many places where a program can send that string: to a window, to a file, or to a networked computer on the other side of the world. You need to specify that the destination for the string is the *system output*—that is, a console window. The console window is represented in Java by an object called `out`. Just as you needed to place the `main` method in a `HelloPrinter` class, the designers of the Java library needed to place the `out` object into a class. They placed it in the `System` class, which contains useful objects and methods to access system resources. To use the `out` object in the `System` class, you must refer to it as `System.out`.

To use an object, such as `System.out`, you specify what you want to do to it. In this case, you want to print a line of text. The `println` method carries out this task.

You do not have to implement this method—the programmers who wrote the Java library already did that for us—but you do need to *call* the method.

Whenever you call a method in Java, you need to specify three items (see Figure 12):

A method is called by specifying an object, the method name, and the method parameters.

1. The object that you want to use (in this case, `System.out`)

2. The name of the method you want to use (in this case, `println`)

3. A pair of parentheses, containing any other information the method needs (in this case, `"Hello, World!"`). The technical term for this information is a *parameter* for the method. Note that the two periods in `System.out.println` have different meanings. The first period means "locate the out object in the `System` class". The second period means "apply the `println` method to that object".

20

Figure 12
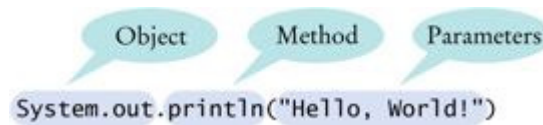
Calling a Method

A sequence of characters enclosed in quotation marks

```
"Hello, World!"
```

is called a *string*. You must enclose the contents of the string inside quotation marks so that the compiler knows you literally mean `"Hello, World!"`. There is a reason for this requirement. Suppose you need to print the word *main*. By enclosing it in quotation marks, `"main"`, the compiler knows you mean the sequence of characters `main`, not the method named `main`. The rule is simply that you must enclose all text strings in quotation marks, so that the compiler considers them plain text and does not try to interpret them as program instructions.

A string is a sequence of characters enclosed in quotation marks.

You can also print numerical values. For example, the statement

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

displays the number 7.

The `println` method prints a string or a number and then starts a new line. For example, the sequence of statements

```
System.out.println("Hello");
System.out.println("World!");
```

prints two lines of text:

```
Hello
World!
```

There is a second method, called `print`, that you can use to print an item without starting a new line. For example, the output of the two statements

```
System.out.print("00");
System.out.println(3 + 4);
```

is the single line

```
007
```

Syntax 1.1 Method Call

object.methodName (*parameters*)

Example:

```
System.out.println("Hello, Dave!")
```

Purpose:

To invoke a method on an object and supply any additional parameters

---

Self Check

12. How would you modify the `HelloPrinter` program to print the words "Hello," and "World!" on two lines?

13. Would the program continue to work if you omitted the line starting with `//`?

14. What does the following set of statements print?

```
System.out.print("My lucky number is");
System.out.println(3 + 4 + 5);
```

Common Error 1.1: Omitting Semicolons

In Java every statement must end in a semicolon. Forgetting to type a semicolon is a common error. It confuses the compiler, because the compiler uses the semicolon to find where one statement ends and the next one starts. The compiler does not use

---

line breaks or closing braces to recognize the end of statements. For example, the compiler considers

```
System.out.println("Hello")
System.out.println("World!");
```

a single statement, as if you had written

```
    System.out.println("Hello")
System.out.println("World!");
```

Then it doesn't understand that statement, because it does not expect the word `System` following the closing parenthesis after `"Hello"`. The remedy is simple. Scan every statement for a terminating semicolon, just as you would check that every English sentence ends in a period.

ADVANCED TOPIC 1.1: **Alternative Comment Syntax**

In Java there are two methods for writing comments. You already learned that the compiler ignores anything that you type between `//` and the end of the current line. The compiler also ignores any text between a `/*` and `*/`.

```
/* A simple Java program */
```

The `//` comment is easier to type if the comment is only a single line long. If you have a comment that is longer than a line, then the `/* ... */` comment is simpler:

```
/*
    This is a simple Java program that you can use to try out
    your compiler and virtual machine.
*/
```

It would be somewhat tedious to add the `//` at the beginning of each line and to move them around whenever the text of the comment changes.

In this book, we use `//` for comments that will never grow beyond a line, and `/* ... */` for longer comments. If you prefer, you can always use the `//` style. The readers of your code will be grateful for *any* comments, no matter which style you use.

1.7 Errors

Experiment a little with the `HelloPrinter` program. What happens if you make a typing error such as

```
System.ouch.println("Hello, World!");
System.out.println("Hello, World!);
System.out.println("Hello, Word!");
```

In the first case, the compiler will complain. It will say that it has no clue what you mean by `ouch`. The exact wording of the error message is dependent on the compiler, but it might be something like "Undefined symbol ouch". This is a *compile-time error* or *syntax error*. Something is wrong according to the language rules and the compiler finds it. When the compiler finds one or more errors, it refuses to translate the program to Java virtual machine instructions, and as a consequence you have no program that you can run. You must fix the error and compile again. In fact, the compiler is quite picky, and it is common to go through several rounds of fixing compile-time errors before compilation succeeds for the first time.

> A syntax error is a violation of the rules of the programming language. The compiler detects syntax errors.

If the compiler finds an error, it will not simply stop and give up. It will try to report as many errors as it can find, so you can fix them all at once. Sometimes, however, one error throws it off track. This is likely to happen with the error in the second line. Because the closing quotation mark is missing, the compiler will think that the `);` characters are still part of the string. In such cases, it is common for the compiler to emit bogus error reports for neighboring lines. You should fix only those error messages that make sense to you and then recompile.

The error in the third line is of a different kind. The program will compile and run, but its output will be wrong. It will print

```
Hello, Word!
```

This is a *run-time error* or *logic error*. The program is syntactically correct and does something, but it doesn't do what it is supposed to do. The compiler cannot find the

error. You, the programmer, must flush out this type of error. Run the program, and carefully look at its output.

> A logic error causes a program to take an action that the programmer did not intend. You must test your programs to find logic errors.

During program development, errors are unavoidable. Once a program is longer than a few lines, it requires superhuman concentration to enter it correctly without slipping up once. You will find yourself omitting semicolons or quotes more often than you would like, but the compiler will track down these problems for you.

Logic errors are more troublesome. The compiler will not find them—in fact, the compiler will cheerfully translate any program as long as its syntax is correct—but the resulting program will do something wrong. It is the responsibility of the program author to test the program and find any logic errors. Testing programs is an important topic that you will encounter many times in this book. Another important aspect of good craftsmanship is *defensive programming*: structuring programs and development processes in such a way that an error in one part of a program does not trigger a disastrous response.

The error examples that you saw so far were not difficult to diagnose or fix, but as you learn more sophisticated programming techniques, there will also be much more room for error. It is an uncomfortable fact that locating all errors in a program is very difficult. Even if you can observe that a program exhibits faulty behavior, it may not at all be obvious what part of the program caused it and how you can fix it. Special software tools (so-called *debuggers*) let you trace through a program to find *bugs*— that is, logic errors. In Chapter 6 you will learn how to use a debugger effectively.

Note that these errors are different from the types of errors that you are likely to make in calculations. If you total up a column of numbers, you may miss a minus sign or accidentally drop a carry, perhaps because you are bored or tired. Computers do not make these kinds of errors.

This book uses a three-part error management strategy. First, you will learn about common errors and how to avoid them. Then you will learn defensive programming strategies to minimize the likelihood and impact of errors. Finally, you will learn debugging strategies to flush out those errors that remain.

SELF CHECK

15. Suppose you omit the `//` characters from the `HelloPrinter.java` program but not the remainder of the comment. Will you get a compile-time error or a runtime error?

16. How can you find logic errors in a program?

COMMON ERROR 1.2: Misspelling Words

If you accidentally misspell a word, then strange things may happen, and it may not always be completely obvious from the error messages what went wrong. Here is a good example of how simple spelling errors can cause trouble:

```java
public class HelloPrinter
{
    public static void Main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

This class defines a method called `Main`. The compiler will not consider this to be the same as the `main` method, because `Main` starts with an uppercase letter and the Java language is case sensitive. Upper- and lowercase letters are considered to be completely different from each other, and to the compiler `Main` is no better match for `main` than `rain`. The compiler will cheerfully compile your `Main` method, but when the Java virtual machine reads the compiled file, it will complain about the missing `main` method and refuse to run the program. Of course, the message "missing main method" should give you a clue where to look for the error.

If you get an error message that seems to indicate that the compiler is on the wrong track, it is a good idea to check for spelling and capitalization. All Java keywords use only lowercase letters. Names of classes usually start with an uppercase letter, names of methods and variables with a lowercase letter. If you misspell the name of a symbol (for example, `ouch` instead of `out`), the compiler will complain about an "undefined symbol". That error message is usually a good clue that you made a spelling error.

1.8 The Compilation Process

Some Java development environments are very convenient to use. Enter the code in one window, click on a button to compile, and click on another button to execute your program. Error messages show up in a second window, and the program runs in a third window. With such an environment you are completely shielded from the details of the compilation process. On other systems you must carry out every step manually, by typing commands into a shell window.

No matter which compilation environment you use, you begin your activity by typing in the program statements. The program that you use for entering and modifying the program text is called an *editor*. Remember to *save* your work to disk frequently, because otherwise the text editor stores the text only in the computer's memory. If something goes wrong with the computer and you need to restart it, the contents of the primary memory (including your program text) are lost, but anything stored on the hard disk is permanent even if you need to restart the computer.

> An editor is a program for entering and modifying text, such as a Java program.

When you compile your program, the compiler translates the Java *source code* (that is, the statements that you wrote) into *class files*, which consist of virtual machine instructions and other information that is required for execution. The class files have the extension `.class`. For example, the virtual machine instructions for the `Hello-Printer` program are stored in a file `HelloPrinter.class`. As already mentioned, the compiler produces a class file only after you have corrected all syntax errors.

> The Java compiler translates source code into class files that contain instructions for the Java virtual machine.

The class file contains the translation of only the instructions that you wrote. That is not enough to actually run the program. To display a string in a window, quite a bit of low-level activity is necessary. The authors of the `System` and `PrintStream` classes (which define the `out` object and the `println` method) have implemented all 25 necessary actions and placed the required class files into a *library*. A library is a 26
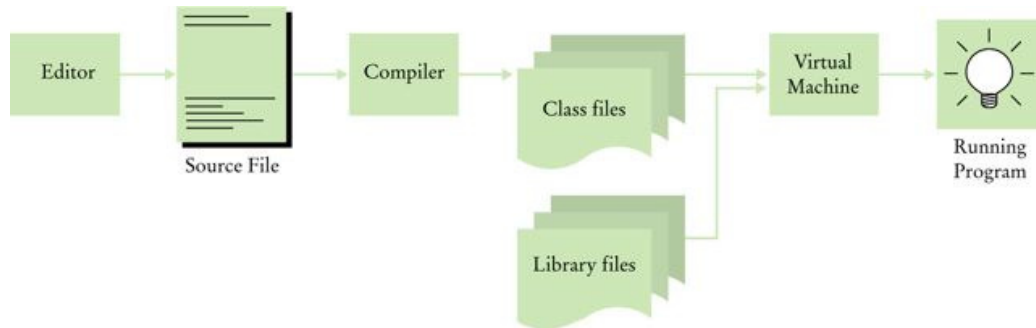
collection of code that has been programmed and translated by someone else, ready for you to use in your program.
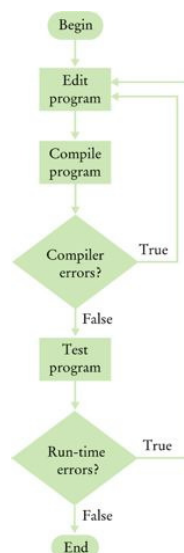
Figure 13

From Source Code to Running Program

The Java virtual machine loads the instructions for the program that you wrote, starts your program, and loads the necessary library files as they are required.

The steps of compiling and running your program are outlined in Figure 13.

Figure 14

The Edit-Compile-Test Loop

Programming activity centers around these steps. Start in the editor, writing the source file. Compile the program and look at the error messages. Go back to the editor and fix the syntax errors. When the compiler succeeds, run the program. If you find a run-time error, you must look at the source code in the editor to try to determine the reason. Once you find the cause of the error, fix it in the editor. Compile and run again to see whether the error has gone away. If not, go back to the editor. This is called the *edit–compile–test loop* (see Figure 14). You will spend a substantial amount of time in this loop when working on programming assignments.

> The Java virtual machine loads program instructions from class files and library files.

SELF CHECK

17. What do you expect to see when you load a class file into your text editor?

18. Why can't you test a program for run-time errors when it has compiler errors?

CHAPTER SUMMARY

1. A computer must be programmed to perform tasks. Different tasks require different programs.

2. A computer program executes a sequence of very basic operations in rapid succession.

3. A computer program contains the instruction sequences for all tasks that it can execute.

4. At the heart of the computer lies the central processing unit (CPU).

5. Data and programs are stored in primary storage (memory) and secondary storage (such as a hard disk).

6. The CPU reads machine instructions from memory. The instructions direct it to communicate with memory, secondary storage, and peripheral devices.

7. Generally, machine code depends on the CPU type. However, the instruction set of the Java virtual machine (JVM) can be executed on many CPUs.

8. Because machine instructions are encoded as numbers, it is difficult to write programs in machine code.

9. High-level languages allow you to describe tasks at a higher conceptual level than machine code.

10. A compiler translates programs written in a high-level language into machine code.

11. Java was originally designed for programming consumer devices, but it was first successfully used to write Internet applets.

27

28

12. Java was designed to be safe and portable, benefiting both Internet users and students.

13. Java has a very large library. Focus on learning those parts of the library that you need for your programming projects.

14. Set aside some time to become familiar with the computer system and the Java compiler that you will use for your class work.

15. Develop a strategy for keeping backup copies of your work before disaster strikes.

16. Java is case sensitive. You must be careful about distinguishing between upper-and lowercase letters.

17. Lay out your programs so that they are easy to read.

18. Classes are the fundamental building blocks of Java programs.

19. Every Java application contains a class with a main method. When the application starts, the instructions in the main method are executed.

20. Each class contains definitions of methods. Each method contains a sequence of instructions.

21. Use comments to help human readers understand your program.

22. A method is called by specifying an object, the method name, and the method parameters.

23. A string is a sequence of characters enclosed in quotation marks.

24. A syntax error is a violation of the rules of the programming language. The compiler detects syntax errors.

25. A logic error causes a program to take an action that the programmer did not intend. You must test your programs to find logic errors.

26. An editor is a program for entering and modifying text, such as a Java program.

27. The Java compiler translates source code into class files that contain instructions for the Java virtual machine.

28. The Java virtual machine loads program instructions from class files and library files.

FURTHER READING

1. http://jmol.sourceforge.net/applet/ The web site for the jmol applet for visualizing molecules.

28

29

CLASSES, OBJECTS, AND METHODS INTRODUCED IN THIS CHAPTER

Here is a list of all classes, methods, static variables, and constants introduced in this chapter. Turn to the documentation in Appendix C for more information.

```
java.io.PrintStream
    print
    println
java.lang.System
    out
```

REVIEW EXERCISES

★ **Exercise R1.1.** Explain the difference between using a computer program and programming a computer.

★ **Exercise R1.2.** What distinguishes a computer from a typical household appliance?

★ **Exercise R1.3.** Rank the storage devices that can be part of a computer system by

a. Speed

b. Cost

c. Storage capacity

★★ **Exercise R1.4.** What is the Java virtual machine?

★ **Exercise R1.5.** What is an applet?

★ **Exercise R1.6.** What is an integrated programming environment?

★ **Exercise R1.7.** What is a console window?

★★ **Exercise R1.8.** Describe *exactly* what steps you would take to back up your work after you have typed in the `HelloPrinter.java` program.

★★ **Exercise R1.9.** On your own computer or on a lab computer, find the exact location (folder or directory name) of

　　a. The sample file `HelloPrinter.java`, which you wrote with the editor

　　b. The Java program launcher `java.exe` or `java`

　　c. The library file `rt.jar` that contains the run-time library

★ **Exercise R1.10.** How do you discover syntax errors? How do you discover logic errors?

★★ **Exercise R1.11.** Write three versions of the `HelloPrinter.java` program that have different syntax errors. Write a version that has a logic error.

★★★ **Exercise R1.12.** What do the following statements print? Don't guess; write programs to find out.

    a. `System.out.println("3 + 4");`

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

    c. `System.out.println(3 + "4");`

Additional review exercises are available in WileyPLUS.

PROGRAMMING EXERCISES

★ **Exercise P1.1.** Write a program `NamePrinter` that displays your name inside a box on the console screen, like this:

Do your best to approximate lines with characters, such as |, –, and +.

★ **Exercise P1.2.** Write a program `FacePrinter` that prints a face, using text characters, hopefully better looking than this one:
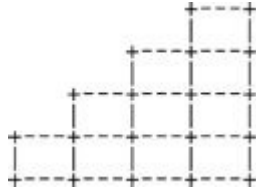

Use *comments* to indicate the statements that print the hair, ears, mouth, and so on.

★ **Exercise P1.3.** Write a program `TicTacToeBoardPrinter` that prints a tic-tac-toe board:

★ **Exercise P1.4.** Write a program `StaircasePrinter` that prints a staircase:

★★ **Exercise P1.5.** Write a program that computes the sum of the first ten positive integers, $1 + 2 + \cdots + 10$. *Hint*: Write a program of the form

```
public class Sum10
{
   public static void main(String[] args)
   {
      System.out.println(          );
   }
}
```

★★ **Exercise P1.6.** Write a program `Sum10Reciprocals` that computes the sum of the reciprocals $1/1 + 1/2 + \cdots + 1/10$. This is harder than it sounds. Try writing the program, and check the result. The program's result isn't likely to be correct. Then write the denominators as *floating-point numbers*, `1.0`, `2.0`, …, `10.0`, and run the program again. Can you explain the difference in the results? We will explore this phenomenon in Chapter 4.

★★ **Exercise P1.7.** Type in and run the following program:

```
 import javax.swing.JOptionPane;
public class DialogViewer
{
   public static void main(String[] args)
   {
      JOptionPane.showMessageDialog(null, "Hello,
World!");
      System.exit(0);
   }
}
```

Then modify the program to show the message "Hello, *your name*!".

★★ **Exercise P1.8.** Type in and run the following program:

```
  import javax.swing.JOptionPane;
public class DialogViewer
{
   public static void main(String[] args)
   {
      String name =
JOptionPane.showInputDialog("What is your name?");
      System.out.println(name);
      System.exit(0);
   }
}
```

Then modify the program to print "Hello, *name*!", displaying the name that the user typed in.

   ⬠   Additional programming exercises are available in WileyPLUS.

PROGRAMMING PROJECTS

★★★ **Project 1.1.** This project builds on Exercises P1.7 and P1.8. Your program should read the user's name, then show a sequence of two dialog boxes:

- First, an input dialog box that asks: "What would you like me to do?"

- Then a message dialog box that says: "I'm sorry, (*your name*). I'm afraid I can't do that."

ANSWERS TO SELF-CHECK QUESTIONS

1. A program that reads the data on the CD and sends output to the speakers and the screen.

2. A CD player can do one thing—play music CDs. It cannot execute programs.

3. No—the program simply executes the instruction sequences that the programmers have prepared in advance.

4. In secondary storage, typically a hard disk.

5. The central processing unit.

6. 21 100

7. No—a compiler is intended for programmers, to translate high-level programming instructions into machine code.

8. Safety and portability.

9. No one person can learn the entire library—it is too large.

10. Programs are stored in files, and files are stored in folders or directories.

11. You back up your files and folders.

12. `System.out.println("Hello,"); System.out.println(" World!");`

13. Yes—the line starting with `//` is a comment, intended for human readers. The compiler ignores comments.

14. The printout is `My lucky number is12`. It would be a good idea to add a space after the `is`.

15. A compile-time error. The compiler will not know what to do with the word `Display`.

16. You need to run the program and observe its behavior.

17. A sequence of random characters, some funny-looking. Class files contain virtual machine instructions that are encoded as binary numbers.

18. When a program has compiler errors, no class file is produced, and there is nothing to run.