

CHAPTER 1

INTRODUCTION



© JanPietruszka/iStockphoto.

CHAPTER GOALS

- To learn about computers and programming
- To compile and run your first Java program
- To recognize compile-time and run-time errors
- To describe an algorithm with pseudocode

CHAPTER CONTENTS

- 1.1 COMPUTER PROGRAMS** 2
- 1.2 THE ANATOMY OF A COMPUTER** 3
 - C&S** Computers Are Everywhere 5
- 1.3 THE JAVA PROGRAMMING LANGUAGE** 6
- 1.4 BECOMING FAMILIAR WITH YOUR PROGRAMMING ENVIRONMENT** 7
 - PT1** Backup Copies 10
- 1.5 ANALYZING YOUR FIRST PROGRAM** 11
 - SYN** Java Program 12
 - CE1** Omitting Semicolons 13
- 1.6 ERRORS** 14
 - CE2** Misspelling Words 15
- 1.7 PROBLEM SOLVING: ALGORITHM DESIGN** 15
 - HT1** Describing an Algorithm with Pseudocode 19
 - WE1** Writing an Algorithm for Tiling a Floor 21



© JanPietruszka/iStockphoto.

Just as you gather tools, study a project, and make a plan for tackling it, in this chapter you will gather up the basics you need to start learning to program. After a brief introduction to computer hardware, software, and programming in general, you will learn how to write and run your first Java program. You will also learn how to diagnose and fix programming errors, and how to use pseudocode to describe an algorithm—a step-by-step description of how to solve a problem—as you plan your computer programs.

1.1 Computer Programs

Computers execute very basic instructions in rapid succession.

A computer program is a sequence of instructions and decisions.

Programming is the act of designing and implementing computer programs.

You have probably used a computer for work or fun. Many people use computers for everyday tasks such as electronic banking 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.

The flexibility of a computer is quite an amazing phenomenon. The same machine can balance your checkbook, lay out 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. Computers can carry out a wide range of tasks because they execute different programs, each of which directs the computer to work on a specific task.

The computer itself is a machine that stores data (numbers, words, pictures), interacts with devices (the monitor, the sound system, the printer), and executes programs. A **computer program** tells a computer, in minute detail, the sequence of steps that are needed to fulfill a task. The physical computer and peripheral devices are collectively called the **hardware**. The programs the computer executes are called the **software**.

Today's computer programs are so sophisticated that it is hard to believe that they are composed of extremely primitive instructions. A typical instruction may be one of the following:

- Put a red dot at a given screen position.
- Add up two numbers.
- If this value is negative, continue the program at a certain instruction.

The computer user has the illusion of smooth interaction because a program contains a huge number of such instructions, and because the computer can execute them at great speed.

The act of designing and implementing computer programs is called **programming**. In this book, you will learn how to program a computer—that is, how to direct the computer to execute tasks.

To write a computer game with motion and sound effects or a word processor that supports fancy fonts and pictures is a complex task that requires a team of many highly-skilled programmers. Your first programming efforts will be more mundane. The concepts and skills you learn in this book form an important foundation, and you should not be disappointed if your first programs do not rival the sophisticated software that is familiar to you. Actually, you will find that there is an immense thrill even in simple programming tasks. It is an amazing experience to see the computer precisely and quickly carry out a task that would take you hours of drudgery, to

make small changes in a program that lead to immediate improvements, and to see the computer become an extension of your mental powers.



1. What is required to play music on a computer?
2. Why is a CD player less flexible than a computer?
3. What does a computer user need to know about programming in order to play a video game?

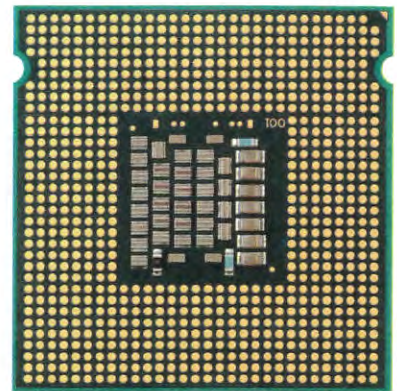
1.2 The Anatomy of a Computer

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

At the heart of the computer lies the **central processing unit (CPU)** (see Figure 1). The inside wiring of the CPU is enormously complicated. For example, the Intel Core processor (a popular CPU for personal computers at the time of this writing) is composed of several hundred million structural elements, called *transistors*.

The CPU performs program control and data processing. That is, the CPU locates and executes the program instructions; it carries out arithmetic operations such as addition, subtraction, multiplication, and division; it fetches data from external memory or devices and places processed data into storage.

There are two kinds of storage. Primary storage, or memory, is made from electronic circuits that can store data, provided they are supplied with electric power. **Secondary storage**, usually a **hard disk** (see Figure 2) or a solid-state drive, provides slower and less expensive storage that persists without



© Amorphis/iStockphoto.

Figure 1 Central Processing Unit

The central processing unit (CPU) performs program control and data processing.

Storage devices include memory and secondary storage.



PhotoDisc, Inc./Getty Images, Inc.

Figure 2 A Hard Disk

electricity. A hard disk consists of rotating platters, which are coated with a magnetic material. A solid-state drive uses electronic components that can retain information without power, and without moving parts.

To interact with a human user, a computer requires peripheral devices. The computer transmits information (called *output*) to the user through a display screen, speakers, and printers. The user can enter information (called *input*) for the computer by using a keyboard or a pointing device such as a mouse.

Some computers are self-contained units, whereas others are interconnected through **networks**. Through the network cabling, the computer can read data and programs from central storage locations or send data to other computers. To 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 3 gives a schematic overview of the architecture of a personal computer. Program instructions and data (such as text, numbers, audio, or video) reside in secondary storage or elsewhere on the network. When a program is started, its instructions are brought into memory, where the CPU can read them. The CPU reads and executes one instruction at a time. As directed by these instructions, the CPU reads data, modifies it, and writes it back to memory or secondary storage. Some program instructions will cause the CPU to place dots on the display screen or printer or to vibrate the speaker. As these actions happen many times over and at great speed, the human user will perceive images and sound. Some program instructions read user input from the keyboard, mouse, touch sensor, or microphone. The program analyzes the nature of these inputs and then executes the next appropriate instruction.

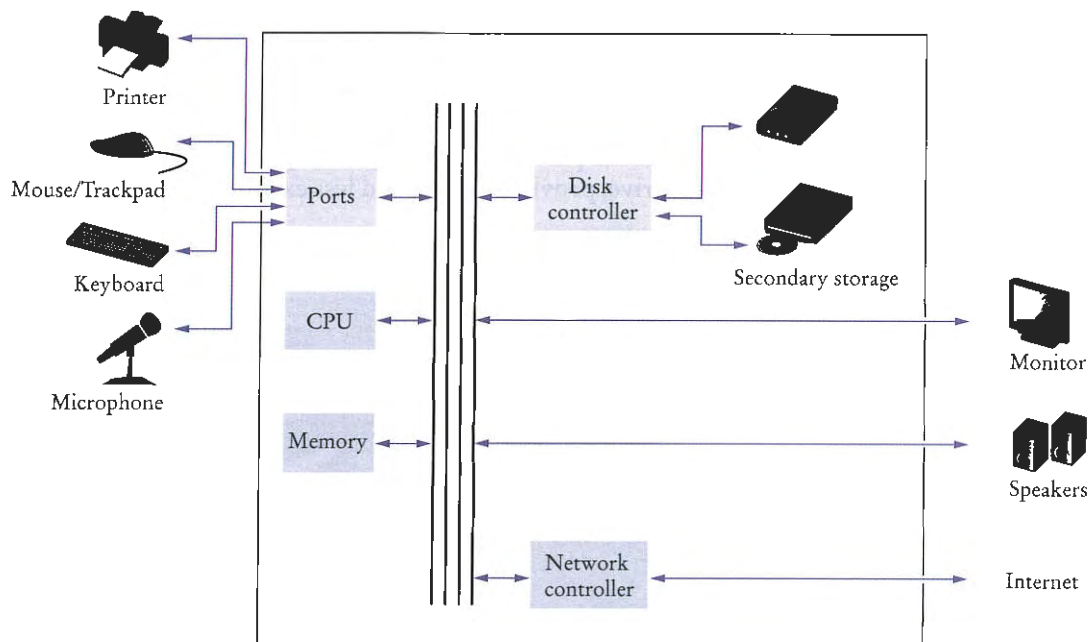


Figure 3 Schematic Design of a Personal Computer

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?
6. A modern smartphone is a computer, comparable to a desktop computer. Which components of a smartphone correspond to those shown in Figure 3?

Practice It Now you can try these exercises at the end of the chapter: R1.2, R1.3.



Computing & Society 1.1 Computers Are Everywhere

When computers were first invented in the 1940s, a computer filled an entire room. The photo below shows the ENIAC (electronic numerical integrator and computer), completed in 1946 at the University of Pennsylvania. The ENIAC was used by the military to compute the trajectories of projectiles. Nowadays, computing facilities of search engines, Internet shops, and social networks fill huge buildings called data centers. At the other end of the spectrum, computers are all around us. Your cell phone has a computer inside, as do many credit cards and fare cards for public transit. A modern car has several computers—to control the engine, brakes, lights, and the radio.

The advent of ubiquitous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today carried out by computer-controlled robots, operated by a few people who know how to work with those computers. Books, music, and movies nowadays are often consumed on computers, and computers are almost always involved in their production. The book that you are reading right now could not have



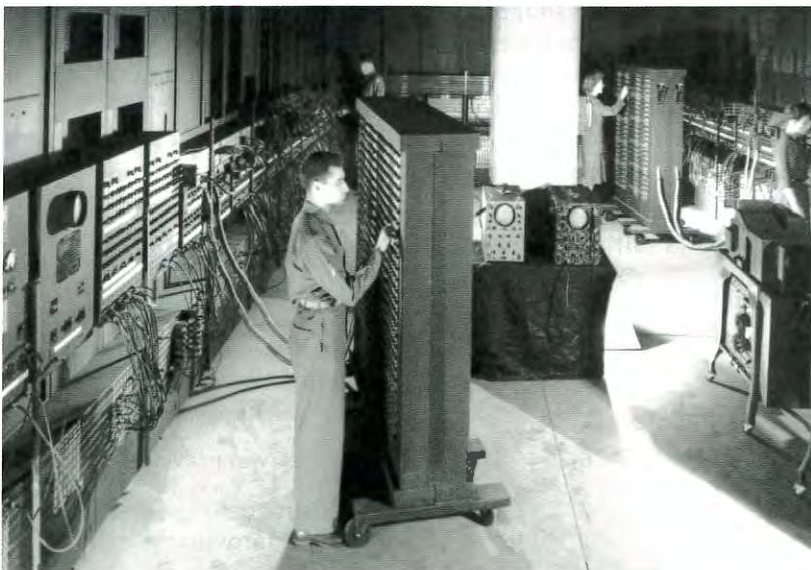
© Maurice Savage/Alamy Limited.

This transit card contains a computer.

been written without computers.

Knowing about computers and how to program them has become an essential skill in many careers. Engineers design computer-controlled cars and medical equipment that preserve lives. Computer scientists develop programs that help people come together to support social causes. For example, activists used social networks to share videos showing abuse by repressive regimes, and this information was instrumental in changing public opinion.

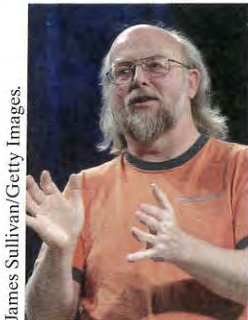
As computers, large and small, become ever more embedded in our everyday lives, it is increasingly important for everyone to understand how they work, and how to work with them. As you use this book to learn how to program a computer, you will develop a good understanding of computing fundamentals that will make you a more informed citizen and, perhaps, a computing professional.



© UPPA/Photoshot.

The ENIAC

1.3 The Java Programming Language



James Sullivan/Getty Images

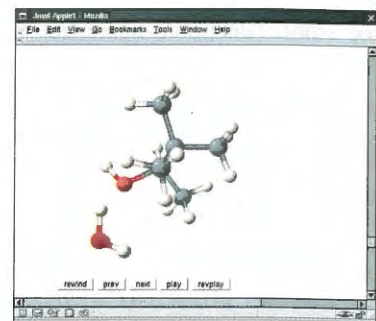
James Gosling

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

In order to write a computer program, you need to provide a sequence of instructions that the CPU can execute. A computer program consists of a large number of simple CPU instructions, and it is tedious and error-prone to specify them one by one. For that reason, **high-level programming languages** have been created. In a high-level language, you specify the actions that your program should carry out. A **compiler** translates the high-level instructions into the more detailed instructions (called **machine code**) required by the CPU. Many different programming languages have been designed for different purposes.

In 1991, a group led by James Gosling and Patrick Naughton at Sun Microsystems designed a programming language, code-named “Green”, for use in consumer devices, such as intelligent television “set-top” boxes. The language was designed to be simple, secure, and usable for many different processor types. No customer was ever found for this technology.

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, together with a browser that ran **applets**—Java code that can be located anywhere on the Internet. The figure at right shows a typical example of an applet.



An Applet for Visualizing Molecules

Since then, Java has grown at a phenomenal rate. Programmers have embraced the language because it is easier to use 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 allow Java programmers to target hardware ranging from smart cards to the largest Internet servers.

Because Java was designed for the Internet, it has two attributes that make it very suitable for beginners: safety and portability.

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

Table 1 Java Versions (since Version 1.0 in 1996)

Version	Year	Important New Features	Version	Year	Important New Features
1.1	1997	Inner classes	5	2004	Generic classes, enhanced for loop, auto-boxing, enumerations, annotations
1.2	1998	Swing, Collections framework	6	2006	Library improvements
1.3	2000	Performance enhancements	7	2011	Small language changes and library improvements
1.4	2002	Assertions, XML support	8	2014	Function expressions, streams, new date/time library

Java was designed so that anyone can execute programs in their browser without fear. The safety features of the Java language ensure that a program is terminated if it tries to do something unsafe. Having a safe environment is also helpful for anyone learning Java. When you make an error that results in unsafe behavior, your program is terminated and you receive an accurate error report.

The other benefit of Java is portability. The same Java program will run, without change, on Windows, UNIX, Linux, or Macintosh. In order to achieve portability, the Java compiler does not translate Java programs directly into CPU instructions. Instead, compiled Java programs contain instructions for the Java **virtual machine**, a program that simulates a real CPU. Portability is another benefit for the beginning student. You do not have to learn how to write programs for different platforms.

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 to write even the simplest programs. This is not a problem for professional programmers, but it can be a nuisance 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 more complete detail in a later chapter.

Java has been extended many times during its life—see Table 1. In this book, we assume that you have Java version 7 or later.

Finally, you cannot hope to learn all of Java in one course. 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.

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.

Java programs are distributed as instructions for a virtual machine, making them platform-independent.

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



7. What are the two most important benefits of the Java language?
8. How long does it take to learn the entire Java library?

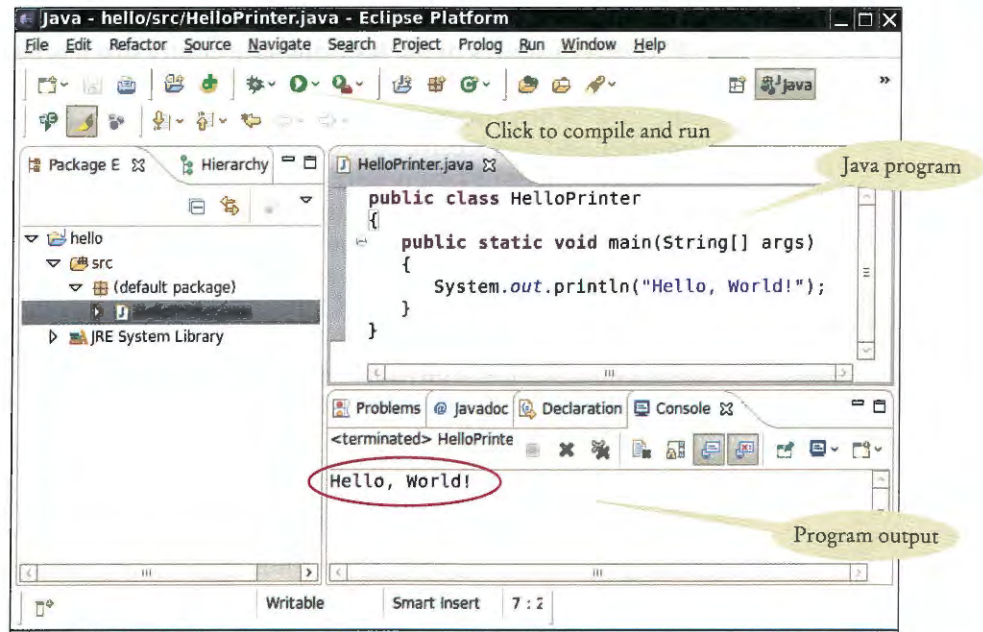
Practice It Now you can try this exercise at the end of the chapter: R1.5.

1.4 Becoming Familiar with Your Programming Environment

Set aside time to become familiar with the programming environment that you will use for your class work.

Many students find that the tools they need as programmers are very different from the software with which they are familiar. You should spend some time making yourself familiar with your programming environment. Because computer systems vary widely, this book can only give an outline of the steps you need to follow. It is a good idea to participate in a hands-on lab, or to ask a knowledgeable friend to give you a tour.

Figure 4
Running the
HelloPrinter
Program in an
Integrated
Development
Environment



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

Step 1 Start the Java development environment.

Computer systems differ greatly in this regard. On many computers there is an **integrated development environment** in which you can write and test your programs. On other computers you first launch an **editor**, a program that functions like a word processor, in which you can enter your Java instructions; you then open a *console window* and type commands to execute your program. You need to find out how to get started with your environment.

Step 2 Write a simple 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:

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

We will examine this program in the next section.

No matter which programming environment you use, you begin your activity by typing the program statements into an editor window.

Create a new file and call it `HelloPrinter.java`, using the steps that are appropriate for your environment. (If your environment requires that you supply a project name in addition to the file name, use the name `hello` for the project.) Enter the program instructions *exactly* as they are given above. Alternatively, locate the electronic copy in this book's companion code and paste it into your editor.

Figure 5
Running the HelloPrinter
Program in a Console Window

```

Terminal
File Edit View Terminal Tabs Help
~$ cd BigJava/ch01/hello
~/BigJava/ch01/hello$ javac HelloPrinter.java
~/BigJava/ch01/hello$ java HelloPrinter
Hello, World!
~/BigJava/ch01/hello$

```

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

As you write this program, pay careful attention to the various symbols, and keep in mind that 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 on page 15.

Step 3 Run the program.

The process for running a program depends greatly on your programming environment. You may have to click a button or enter some commands. When you run the test program, the message

Hello, World!

will appear somewhere on the screen (see Figures 4 and 5).

In order to run your program, the Java compiler translates your **source files** (that is, the statements that you wrote) into **class files**. (A class file contains instructions for the Java virtual machine.) After the compiler has translated your **source code** into virtual machine instructions, the virtual machine executes them. During execution, the virtual machine accesses a library of pre-written code, including the implementations of the `System` and `PrintStream` classes that are necessary for displaying the program's output. Figure 6 summarizes the process of creating and running a Java program. In some programming environments, the compiler and virtual machine are essentially invisible to the programmer—they are automatically executed whenever you ask to run a Java program. In other environments, you need to launch the compiler and virtual machine explicitly.

Step 4 Organize your work.

As a programmer, you write programs, try them out, and improve them. You store your programs in **files**. Files are stored in **folders** or **directories**. A folder can contain

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



Figure 6
From Source Code
to Running Program

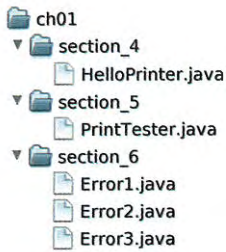


Figure 7
A Folder Hierarchy

files as well as other folders, which themselves can contain more files and folders (see Figure 7). This hierarchy can be quite large, and you need not be concerned with all of its branches. However, you should create folders for organizing your work. It is a good idea to make a separate folder for your programming coursework. Inside that folder, make a separate folder for each program.

Some programming environments place your programs into a default location if you don't specify a folder yourself. In that case, you need to find out where those files are located.

Be sure that you understand where your files are located in the folder hierarchy. This information is essential when you submit files for grading, and for making backup copies (see Programming Tip 1.1).



9. Where is the `HelloPrinter.java` file stored on your computer?
10. What do you do to protect yourself from data loss when you work on programming projects?

Practice It Now you can try this exercise at the end of the chapter: R1.6.

Programming Tip 1.1



Backup Copies

You will spend many hours creating and improving Java programs. It is easy to delete a file by accident, and occasionally files are lost because of a computer malfunction. Retyping the contents of lost files is frustrating and time-consuming. It is therefore crucially important that you learn how to safeguard files and get in the habit of doing so *before* disaster strikes. Backing up files on a memory stick is an easy and convenient storage method for many people. Another increasingly popular form of backup is Internet file storage. Here are a few pointers to keep in mind:



© Tatiana Popova/Stockphoto.

- *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 could have saved easily. I recommend that you back up your work once every thirty minutes.
- *Rotate backups.* Use more than one directory for backups, and rotate them. That is, first back up onto the first directory. Then back up onto the second directory. Then use the third, and then go back to the first. That way you always have three recent backups. If your recent changes made matters worse, you can then go back to the older version.
- *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 to find out that the backups are not there when you need them.
- *Relax, then restore.* When you lose a file and need to restore it from a 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.

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

1.5 Analyzing Your First Program



© Amanda Rohde/Stockphoto.

In this section, we will analyze the first Java program in detail. Here again is the source code:

section_5/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 }
```

The line

```
public class HelloPrinter
```

indicates the declaration of a **class** called `HelloPrinter`.

Every Java program consists of one or more classes. We will discuss classes in more detail in Chapters 2 and 3.

The word `public` denotes that the class is usable by the “public”. You will later encounter private features.

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 named `HelloPrinter.java`.

The construction

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

declares 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.

The term `static` is explained in more detail in Chapter 8, and the meaning of `String[] args` is covered in Chapter 11. At this time, simply consider

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

as a part of the “plumbing” that is required to create a Java program. Our first program has all instructions inside the `main` method of the class.

The `main` method contains one or more instructions called **statements**. Each statement ends in a semicolon (;). When a program runs, the statements in the `main` method are executed one by one.

Classes are the fundamental building blocks of Java programs.

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

Each class contains declarations of methods. Each method contains a sequence of instructions.

Syntax 1.1 Java Program

```

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

```

Every Java program contains a main method with this header.

Every program contains at least one class. Choose a class name that describes the program action.

The statements inside the main method are executed when the program runs.

Each statement ends in a semicolon. See page 13.

Be sure to match the opening and closing braces.

Replace this statement when you write your own programs.

In our example program, the `main` method has a single statement:

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

This statement prints a line of text, namely “Hello, World!”. In this statement, we *call* a method which, for reasons that we will not explain here, is specified by the rather long name `System.out.println`.

We do not have to implement this method—the programmers who wrote the Java library already did that for us. We simply want the method to perform its intended task, namely to print a value.

Whenever you call a method in Java, you need to specify

1. The method you want to use (in this case, `System.out.println`).
2. Any values the method needs to carry out its task (in this case, “Hello, World!”). The technical term for such a value is an **argument**. Arguments are enclosed in parentheses. Multiple arguments are separated by commas.

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 `m a i n`, 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.

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

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

evaluates the expression `3 + 4` and displays the number 7.

A method is called by specifying the method and its arguments.

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

The `System.out.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, `System.out.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
```



FULL CODE EXAMPLE

Go to wiley.com/go/bje06code to download a program to demonstrate print commands.



SELF CHECK

11. How do you modify the `HelloPrinter` program to greet you instead?
12. How would you modify the `HelloPrinter` program to print the word "Hello" vertically?
13. Would the program continue to work if you replaced line 7 with this statement?
`System.out.println(Hello);`
14. What does the following set of statements print?
`System.out.print("My lucky number is");`
`System.out.println(3 + 4 + 5);`
15. What do the following statements print?
`System.out.println("Hello");`
`System.out.println("");`
`System.out.println("World");`

Practice It Now you can try these exercises at the end of the chapter: R1.7, R1.8, E1.5, E1.8.

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. However, do not add a semicolon at the end of `public class Hello` or `public static void main`. These lines are not statements.

1.6 Errors

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

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

A compile-time error is a violation of the programming language rules that is detected by the compiler.

In the first case, the compiler will complain. It will say that it has no clue what you mean by `ou`. The exact wording of the error message is dependent on your development environment, but it might be something like “Cannot find symbol `ou`”. This is a **compile-time error**. Something is wrong according to the rules of the language and the compiler finds it. For this reason, compile-time errors are often called **syntax errors**. When the compiler finds one or more errors, it refuses to translate the program into 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.

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, an error throws the compiler off track. Suppose, for example, you forget the quotation marks around a string: `System.out.println(Hello, World!)`. The compiler will not complain about the missing quotation marks. Instead, it will report “Cannot find symbol `Hello`”. Unfortunately, the compiler is not very smart and it does not realize that you meant to use a string. It is up to you to realize that you need to enclose strings in quotation marks.

The error in the second line above 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**. The program is syntactically correct and does something, but it doesn’t do what it is supposed to do. Because run-time errors are caused by logical flaws in the program, they are often called **logic errors**.

This particular run-time error did not include an error message. It simply produced the wrong output. Some kinds of run-time errors are so severe that they generate an **exception**: an error message from the Java virtual machine. For example, if your program includes the statement

```
System.out.println(1 / 0);
```

you will get a run-time error message “Division by zero”.

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

Run-time 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—



Programmers spend a fair amount of time fixing compile-time and run-time errors.

© Martin Carlsson/iStockphoto.

A run-time error causes a program to take an action that the programmer did not intend.

FULL CODE EXAMPLE

Go to wiley.com/go/bj6e6code to download three programs that illustrate errors.

but the resulting program will do something wrong. It is the responsibility of the program author to test the program and find any run-time errors.



16. Suppose you omit the "" characters around `Hello, World!` from the `HelloPrinter.java` program. Is this a compile-time error or a run-time error?
17. Suppose you change `println` to `println` in the `HelloPrinter.java` program. Is this a compile-time error or a run-time error?
18. Suppose you change `main` to `hello` in the `HelloPrinter.java` program. Is this a compile-time error or a run-time error?
19. When you used your computer, you may have experienced a program that “crashed” (quit spontaneously) or “hung” (failed to respond to your input). Is that behavior a compile-time error or a run-time error?
20. Why can’t you test a program for run-time errors when it has compiler errors?

Practice It Now you can try these exercises at the end of the chapter: R1.9, R1.10, R1.11.

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:

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

This class declares 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 or virtual machine is on the wrong track, check for spelling and capitalization. If you misspell the name of a symbol (for example, `ou` instead of `out`), the compiler will produce a message such as “cannot find symbol `ou`”. That error message is usually a good clue that you made a spelling error.

1.7 Problem Solving: Algorithm Design

You will soon learn how to program calculations and decision making in Java. But before we look at the mechanics of implementing computations in the next chapter, let’s consider how you can describe the steps that are necessary for finding the solution to a problem.

1.7.1 The Algorithm Concept

You may have run across advertisements that encourage you to pay for a computerized service that matches you up with a love partner. Think how this might work. You fill out a form and send it in. Others do the same. The data are processed by a computer program. Is it reasonable to assume that the computer can perform the task of finding the best match for you? Suppose your younger brother, not the computer, had all the forms on his desk. What instructions could you give him? You can't say, "Find the best-looking person who likes inline skating and browsing the Internet". There is no objective standard for good looks, and your brother's opinion (or that of a computer program analyzing the photos of prospective partners) will likely be different from yours. If you can't give written instructions for someone to solve the problem, there is no way the computer can magically find the right solution. The computer can only do what you tell it to do. It just does it faster, without getting bored or exhausted.

For that reason, a computerized match-making service cannot guarantee to find the optimal match for you. Instead, you may be presented with a set of potential partners who share common interests with you. That is a task that a computer program can solve.

In order for a computer program to provide an answer to a problem that computes an answer, it must follow a sequence of steps that is

- Unambiguous
- Executable
- Terminating

The step sequence is *unambiguous* when there are precise instructions for what to do at each step and where to go next. There is no room for guesswork or personal opinion. A step is *executable* when it can be carried out in practice. For example, a computer can list all people that share your hobbies, but it can't predict who will be your life-long partner. Finally, a sequence of steps is *terminating* if it will eventually come to an end. A program that keeps working without delivering an answer is clearly not useful.

A sequence of steps that is unambiguous, executable, and terminating is called an **algorithm**. Although there is no algorithm for finding a partner, many problems do have algorithms for solving them. The next section gives an example.



Finding the perfect partner is not a problem that a computer can solve.

© mammamaart/iStockphoto.

An algorithm for solving a problem is a sequence of steps that is unambiguous, executable, and terminating.



An algorithm is a recipe for finding a solution.

© Claudiad/iStockphoto.

1.7.2 An Algorithm for Solving an Investment Problem

Consider the following investment problem:

You put \$10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

Could you solve this problem by hand? Sure, you could. You figure out the balance as follows:

year	interest	balance
0		10000
1	$10000.00 \times 0.05 = 500.00$	$10000.00 + 500.00 = 10500.00$
2	$10500.00 \times 0.05 = 525.00$	$10500.00 + 525.00 = 11025.00$
3	$11025.00 \times 0.05 = 551.25$	$11025.00 + 551.25 = 11576.25$
4	$11576.25 \times 0.05 = 578.81$	$11576.25 + 578.81 = 12155.06$

You keep going until the balance is at least \$20,000. Then the last number in the year column is the answer.

Of course, carrying out this computation is intensely boring to you or your younger brother. But computers are very good at carrying out repetitive calculations quickly and flawlessly. What is important to the computer is a description of the steps for finding the solution. Each step must be clear and unambiguous, requiring no guesswork. Here is such a description:

Start with a year value of 0, a column for the interest, and a balance of \$10,000.

year	interest	balance
0		10000

Repeat the following steps while the balance is less than \$20,000

Add 1 to the year value.

Compute the interest as $\text{balance} \times 0.05$ (i.e., 5 percent interest).

Add the interest to the balance.

year	interest	balance
0		10000
1	500.00	10500.00
14	942.82	19799.32
(15)	989.96	20789.28

Report the final year value as the answer.

These steps are not yet in a language that a computer can understand, but you will soon learn how to formulate them in Java. This informal description is called **pseudocode**. We examine the rules for writing pseudocode in the next section.

1.7.3 Pseudocode

Pseudocode is an informal description of a sequence of steps for solving a problem.

There are no strict requirements for pseudocode because it is read by human readers, not a computer program. Here are the kinds of pseudocode statements and how we will use them in this book:

- Use statements such as the following to describe how a value is set or changed:

total cost = purchase price + operating cost

Multiply the balance value by 1.05.

Remove the first and last character from the word.

- Describe decisions and repetitions as follows:

If total cost 1 < total cost 2

While the balance is less than \$20,000

For each picture in the sequence

Use indentation to indicate which statements should be selected or repeated:

For each car

operating cost = 10 x annual fuel cost

total cost = purchase price + operating cost

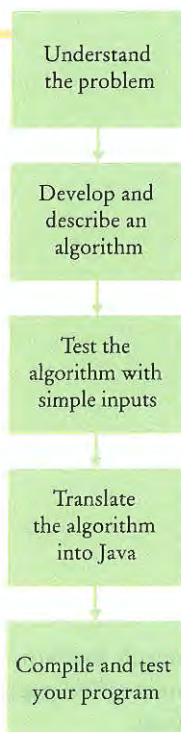
Here, the indentation indicates that both statements should be executed for each car.

- Indicate results with statements such as:

Choose car1.

Report the final year value as the answer.

1.7.4 From Algorithms to Programs



In Section 1.7.2, we developed pseudocode for finding how long it takes to double an investment. Let's double-check that the pseudocode represents an algorithm; that is, that it is unambiguous, executable, and terminating.

Our pseudocode is unambiguous. It simply tells how to update values in each step. The pseudocode is executable because we use a fixed interest rate. Had we said to use the actual interest rate that will be charged in years to come, and not a fixed rate of 5 percent per year, the instructions would not have been executable. There is no way for anyone to know what the interest rate will be in the future. It requires a bit of thought to see that the steps are terminating: With every step, the balance goes up by at least \$500, so eventually it must reach \$20,000.

Therefore, we have found an algorithm to solve our investment problem, and we know we can find the solution by programming a computer. The existence of an algorithm is an essential prerequisite for programming a task. You need to first discover and describe an algorithm for the task before you start programming (see Figure 8). In the chapters that follow, you will learn how to express algorithms in the Java language.

Figure 8 The Software Development Process



21. Suppose the interest rate was 20 percent. How long would it take for the investment to double?
22. Suppose your cell phone carrier charges you \$29.95 for up to 300 minutes of calls, and \$0.45 for each additional minute, plus 12.5 percent taxes and fees. Give an algorithm to compute the monthly charge from a given number of minutes.
23. Consider the following pseudocode for finding the most attractive photo from a sequence of photos:
- Pick the first photo and call it "the best so far".*
- For each photo in the sequence*
- If it is more attractive than the "best so far"*
- Discard "the best so far".*
- Call this photo "the best so far".*
- The photo called "the best so far" is the most attractive photo in the sequence.*
- Is this an algorithm that will find the most attractive photo?
24. Suppose each photo in Self Check 23 had a price tag. Give an algorithm for finding the most expensive photo.
25. Suppose you have a random sequence of black and white marbles and want to rearrange it so that the black and white marbles are grouped together. Consider this algorithm:
- Repeat until sorted*
- Locate the first black marble that is preceded by a white marble, and switch them.*
- What does the algorithm do with the sequence $\circ\bullet\circ\bullet\bullet$? Spell out the steps until the algorithm stops.
26. Suppose you have a random sequence of colored marbles. Consider this pseudocode:
- Repeat until sorted*
- Locate the first marble that is preceded by a marble of a different color, and switch them.*
- Why is this not an algorithm?

Practice It Now you can try these exercises at the end of the chapter: R1.16, E1.4, P1.1.

HOW TO 1.1



Describing an Algorithm with Pseudocode

This is the first of many “How To” sections in this book that give you step-by-step procedures for carrying out important tasks in developing computer programs.

Before you are ready to write a program in Java, you need to develop an algorithm—a method for arriving at a solution for a particular problem. Describe the algorithm in pseudocode—a sequence of precise steps formulated in English. To illustrate, we’ll devise an algorithm for this problem:

Problem Statement You have the choice of buying one of two cars. One is more fuel efficient than the other, but also more expensive. You know the price and fuel efficiency (in miles per gallon, mpg) of both cars. You plan to keep the car for ten years. Assume a price of \$4 per gallon of gas and usage of 15,000 miles per year. You will pay cash for the car and not worry about financing costs. Which car is the better deal?



Step 1 Determine the inputs and outputs.

In our sample problem, we have these inputs:

- **purchase price1** and **fuel efficiency1**, the price and fuel efficiency (in mpg) of the first car
- **purchase price2** and **fuel efficiency2**, the price and fuel efficiency of the second car

We simply want to know which car is the better buy. That is the desired output.

Step 2 Break down the problem into smaller tasks.

For each car, we need to know the total cost of driving it. Let's do this computation separately for each car. Once we have the total cost for each car, we can decide which car is the better deal.

The total cost for each car is **purchase price + operating cost**.

We assume a constant usage and gas price for ten years, so the operating cost depends on the cost of driving the car for one year.

The operating cost is **10 x annual fuel cost**.

The annual fuel cost is **price per gallon x annual fuel consumed**.

The annual fuel consumed is **annual miles driven / fuel efficiency**. For example, if you drive the car for 15,000 miles and the fuel efficiency is 15 miles/gallon, the car consumes 1,000 gallons.

Step 3 Describe each subtask in pseudocode.

In your description, arrange the steps so that any intermediate values are computed before they are needed in other computations. For example, list the step

total cost = purchase price + operating cost

after you have computed **operating cost**.

Here is the algorithm for deciding which car to buy:

For each car, compute the total cost as follows:

annual fuel consumed = annual miles driven / fuel efficiency

annual fuel cost = price per gallon x annual fuel consumed

operating cost = 10 x annual fuel cost

total cost = purchase price + operating cost

If total cost1 < total cost2

Choose car1.

Else

Choose car2.

Step 4 Test your pseudocode by working a problem.

We will use these sample values:

Car 1: \$25,000, 50 miles/gallon

Car 2: \$20,000, 30 miles/gallon

Here is the calculation for the cost of the first car:

annual fuel consumed = annual miles driven / fuel efficiency = 15000 / 50 = 300

annual fuel cost = price per gallon x annual fuel consumed = 4 x 300 = 1200

operating cost = 10 x annual fuel cost = 10 x 1200 = 12000

total cost = purchase price + operating cost = 25000 + 12000 = 37000

Similarly, the total cost for the second car is \$40,000. Therefore, the output of the algorithm is to choose car 1.

The following Worked Example demonstrates how to use the concepts in this chapter and the steps in the How To to solve another problem. In this case, you will see how to develop an algorithm for laying tile in an alternating pattern of colors. You should read the Worked Example to review what you have learned, and for help in tackling another problem.

In future chapters, Worked Examples are provided for you on the book's companion Web site. A brief description of the problem tackled in the example will appear with a reminder to download it from www.wiley.com/go/bje06examples. You will find any code related to the Worked Example included with the book's companion code for the chapter. When you see the Worked Example description, download the example and the code to learn how the problem was solved.

WORKED EXAMPLE 1.1

Writing an Algorithm for Tiling a Floor



Problem Statement Write an algorithm for tiling a rectangular bathroom floor with alternating black and white tiles measuring 4×4 inches. The floor dimensions, measured in inches, are multiples of 4.

Step 1 Determine the inputs and outputs.

The inputs are the floor dimensions (length \times width), measured in inches. The output is a tiled floor.

Step 2 Break down the problem into smaller tasks.

A natural subtask is to lay one row of tiles. If you can solve that task, then you can solve the problem by laying one row next to the other, starting from a wall, until you reach the opposite wall.

How do you lay a row? Start with a tile at one wall. If it is white, put a black one next to it. If it is black, put a white one next to it. Keep going until you reach the opposite wall. The row will contain $\text{width} / 4$ tiles.

Step 3 Describe each subtask in pseudocode.

In the pseudocode, you want to be more precise about exactly where the tiles are placed.

Place a black tile in the northwest corner.

While the floor is not yet filled, repeat the following steps:

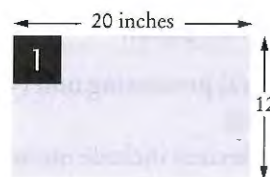
Repeat this step $\text{width} / 4 - 1$ times:

Place a tile east of the previously placed tile. If the previously placed tile was white, pick a black one; otherwise, a white one.

Locate the tile at the beginning of the row that you just placed. If there is space to the south, place a tile of the opposite color below it.

Step 4 Test your pseudocode by working a problem.

Suppose you want to tile an area measuring 20×12 inches. The first step is to place a black tile in the northwest corner.



© rban/Stockphoto.

Next, alternate four tiles until reaching the east wall. ($\text{width} / 4 - 1 = 20 / 4 - 1 = 4$)



There is room to the south. Locate the tile at the beginning of the completed row. It is black. Place a white tile south of it.



Complete the row.



There is still room to the south. Locate the tile at the beginning of the completed row. It is white. Place a black tile south of it.



Complete the row.



Now the entire floor is filled, and you are done.

CHAPTER SUMMARY

Define “computer program” and programming.

- Computers execute very basic instructions in rapid succession.
- A computer program is a sequence of instructions and decisions.
- Programming is the act of designing and implementing computer programs.

Describe the components of a computer.



- The central processing unit (CPU) performs program control and data processing.
- Storage devices include memory and secondary storage.

Describe the process of translating high-level languages to machine code.



- Java was originally designed for programming consumer devices, but it was first successfully used to write Internet applets.
- Java was designed to be safe and portable, benefiting both Internet users and students.
- Java programs are distributed as instructions for a virtual machine, making them platform-independent.
- Java has a very large library. Focus on learning those parts of the library that you need for your programming projects.

Become familiar with your Java programming environment.

- Set aside time to become familiar with the programming environment that you will use for your class work.
- An editor is a program for entering and modifying text, such as a Java program.
- Java is case sensitive. You must be careful about distinguishing between upper- and lowercase letters.
- The Java compiler translates source code into class files that contain instructions for the Java virtual machine.
- Develop a strategy for keeping backup copies of your work before disaster strikes.

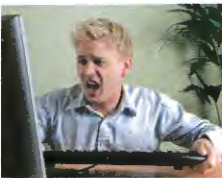


Describe the building blocks of a simple program.



- Classes are the fundamental building blocks of Java programs.
- Every Java application contains a class with a `main` method. When the application starts, the instructions in the `main` method are executed.
- Each class contains declarations of methods. Each method contains a sequence of instructions.
- A method is called by specifying the method and its arguments.
- A string is a sequence of characters enclosed in quotation marks.

Classify program errors as compile-time and run-time errors.



- A compile-time error is a violation of the programming language rules that is detected by the compiler.
- A run-time error causes a program to take an action that the programmer did not intend.

Write pseudocode for simple algorithms.

- An algorithm for solving a problem is a sequence of steps that is unambiguous, executable, and terminating.
- Pseudocode is an informal description of a sequence of steps for solving a problem.



STANDARD LIBRARY ITEMS INTRODUCED IN THIS CHAPTER

```
java.io.PrintStream
  print
  println
```

```
java.lang.System
  out
```

REVIEW EXERCISES

- **R1.1** Explain the difference between using a computer program and programming a computer.
- **R1.2** Which parts of a computer can store program code? Which can store user data?
- **R1.3** Which parts of a computer serve to give information to the user? Which parts take user input?
- **R1.4** A toaster is a single-function device, but a computer can be programmed to carry out different tasks. Is your cell phone a single-function device, or is it a programmable computer? (Your answer will depend on your cell phone model.)
- **R1.5** Explain two benefits of using Java over machine code.
- **R1.6** 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.

- **R1.7** What does this program print?

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("39 + 3");
        System.out.println(39 + 3);
    }
}
```

- **R1.8** What does this program print? Pay close attention to spaces.

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

- **R1.9** What is the compile-time error in this program?

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


- ■ **R1.10** Write three versions of the `HelloPrinter.java` program that have different compile-time errors. Write a version that has a run-time error.
- **R1.11** How do you discover syntax errors? How do you discover logic errors?
- ■ ■ **R1.12** The cafeteria offers a discount card for sale that entitles you, during a certain period, to a free meal whenever you have bought a given number of meals at the regular price. The exact details of the offer change from time to time. Describe an algorithm that lets you determine whether a particular offer is a good buy. What other inputs do you need?
- ■ **R1.13** Write an algorithm to settle the following question: A bank account starts out with \$10,000. Interest is compounded monthly at 6 percent per year (0.5 percent per month). Every month, \$500 is withdrawn to meet college expenses. After how many years is the account depleted?
- ■ ■ **R1.14** Consider the question in Exercise R1.13. Suppose the numbers (\$10,000, 6 percent, \$500) were user selectable. Are there values for which the algorithm you developed would not terminate? If so, change the algorithm to make sure it always terminates.
- ■ ■ **R1.15** In order to estimate the cost of painting a house, a painter needs to know the surface area of the exterior. Develop an algorithm for computing that value. Your inputs are the width, length, and height of the house, the number of windows and doors, and their dimensions. (Assume the windows and doors have a uniform size.)
- ■ **R1.16** In How To 1.1, you made assumptions about the price of gas and annual usage to compare cars. Ideally, you would like to know which car is the better deal without making these assumptions. Why can't a computer program solve that problem?
- ■ **R1.17** Suppose you put your younger brother in charge of backing up your work. Write a set of detailed instructions for carrying out his task. Explain how often he should do it, and what files he needs to copy from which folder to which location. Explain how he should verify that the backup was carried out correctly.
- **R1.18** Write pseudocode for an algorithm that describes how to prepare Sunday breakfast in your household.
- ■ **R1.19** The ancient Babylonians had an algorithm for determining the square root of a number a . Start with an initial guess of $a / 2$. Then find the average of your guess g and a / g . That's your next guess. Repeat until two consecutive guesses are close enough. Write pseudocode for this algorithm.

PRACTICE EXERCISES

- **E1.1** Write a program that prints a greeting of your choice, perhaps in a language other than English.
- ■ **E1.2** Write a program that prints the sum of the first ten positive integers, $1 + 2 + \dots + 10$.
- ■ **E1.3** Write a program that prints the product of the first ten positive integers, $1 \times 2 \times \dots \times 10$. (Use `*` to indicate multiplication in Java.)
- ■ **E1.4** Write a program that prints the balance of an account after the first, second, and third year. The account has an initial balance of \$1,000 and earns 5 percent interest per year.
- **E1.5** Write a program that displays your name inside a box on the screen, like this: Dave
Do your best to approximate lines with characters such as `|` `-` `+`.

- ■ ■ E1.6 Write a program that prints your name in large letters, such as

```

* * ** **** ***** * *
* * * * * * * * * *
***** * * **** ***** * *
* * ***** * * * * *
* * * * * * * * * *

```

- ■ E1.7 Write a program that prints your name in Morse code, like this:

```

.... .- .-. .-. .-.--

```

Use a separate call to `System.out.print` for each letter.

- ■ E1.8 Write a program that prints a face similar to (but different from) the following:

```

/////
+*****+
(| o o |)
 | ^ |
 | '- |
+-----+

```

- ■ E1.9 Write a program that prints an imitation of a Piet Mondrian painting. (Search the Internet if you are not familiar with his paintings.) Use character sequences such as `@@@` or `:::` to indicate different colors, and use `-` and `|` to form lines.
- ■ E1.10 Write a program that prints a house that looks exactly like the following:

```

+
+ +
+ +
+-----+
| .-. |
| | | |
+--+--+

```

- ■ ■ E1.11 Write a program that prints an animal speaking a greeting, similar to (but different from) the following:

```

 ^ ^ ^ ^
( ' ' ) / Hello \
( - ) < Junior |
 | | | \ Coder! /
( _ | ) -----

```

- E1.12 Write a program that prints three items, such as the names of your three best friends or favorite movies, on three separate lines.
- E1.13 Write a program that prints a poem of your choice. If you don't have a favorite poem, search the Internet for "Emily Dickinson" or "e e cummings".
- ■ E1.14 Write a program that prints the United States flag, using `*` and `=` characters.
- ■ E1.15 Type in and run the following program. Then modify it to show the message "Hello, *your name!*".

```

import javax.swing.JOptionPane;

public class DialogViewer
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(null, "Hello, World!");
    }
}

```

```
    }
}
```

- ■ **E1.16** Type in and run the following program. Then modify it to print “Hello, *name!*”, displaying the name that the user typed in.

```
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);
    }
}
```

- ■ ■ **E1.17** Modify the program from Exercise E1.16 so that the dialog continues with the message “My name is Hal! What would you like me to do?” Discard the user’s input and display a message such as

I'm sorry, Dave. I'm afraid I can't do that.

Replace Dave with the name that was provided by the user.

- ■ **E1.18** Type in and run the following program. Then modify it to show a different greeting and image.

```
import java.net.URL;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class Test
{
    public static void main(String[] args) throws Exception
    {
        URL imageUrl = new URL(
            "http://horstmann.com/java4everyone/duke.gif");
        JOptionPane.showMessageDialog(null, "Hello", "Title",
            JOptionPane.PLAIN_MESSAGE, new ImageIcon(imageUrl));
    }
}
```

- **Business E1.19** Write a program that prints a two-column list of your friends’ birthdays. In the first column, print the names of your best friends; in the second, print their birthdays.

- **Business E1.20** In the United States there is no federal sales tax, so every state may impose its own sales taxes. Look on the Internet for the sales tax charged in five U.S. states, then write a program that prints the tax rate for five states of your choice.

Sales Tax Rates	

Alaska:	0%
Hawaii:	4%
. . .	

- **Business E1.21** To speak more than one language is a valuable skill in the labor market today. One of the basic skills is learning to greet people. Write a program that prints a two-column list with the greeting phrases shown in the table. In the first column, print the phrase in English, in the second column, print the phrase in a language of your choice. If you don’t speak a language other than English, use an online translator or ask a friend.

List of Phrases to Translate

Good morning.

It is a pleasure to meet you.

Please call me tomorrow.

Have a nice day!

PROGRAMMING PROJECTS

- ■ P1.1 You want to decide whether you should drive your car to work or take the train. You know the one-way distance from your home to your place of work, and the fuel efficiency of your car (in miles per gallon). You also know the one-way price of a train ticket. You assume the cost of gas at \$4 per gallon, and car maintenance at 5 cents per mile. Write an algorithm to decide which commute is cheaper.
- ■ P1.2 You want to find out which fraction of your car's use is for commuting to work, and which is for personal use. You know the one-way distance from your home to work. For a particular period, you recorded the beginning and ending mileage on the odometer and the number of work days. Write an algorithm to settle this question.
- ■ ■ P1.3 The value of π can be computed according to the following formula:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

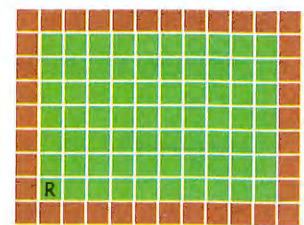
Write an algorithm to compute π . Because the formula is an infinite series and an algorithm must stop after a finite number of steps, you should stop when you have the result determined to six significant digits.

- **Business P1.4** Imagine that you and a number of friends go to a luxury restaurant, and when you ask for the bill you want to split the amount and the tip (15 percent) between all. Write pseudocode for calculating the amount of money that everyone has to pay. Your program should print the amount of the bill, the tip, the total cost, and the amount each person has to pay. It should also print how much of what each person pays is for the bill and for the tip.

- ■ P1.5 Write an algorithm to create a tile pattern composed of black and white tiles, with a fringe of black tiles all around and two or three black tiles in the center, equally spaced from the boundary. The inputs to your algorithm are the total number of rows and columns in the pattern.

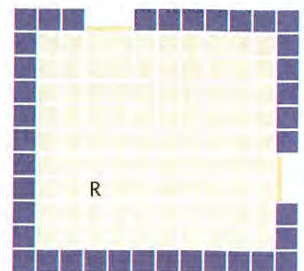


- ■ ■ P1.6 Write an algorithm that allows a robot to mow a rectangular lawn, provided it has been placed in a corner, like this:
The robot can:



- Move forward by one unit.
- Turn left or right.
- Sense the color of the ground one unit in front of it.

- ■ ■ P1.7 Consider a robot that is placed in a room. The robot can:
 - Move forward by one unit.
 - Turn left or right.
 - Sense what is in front of it: a wall, a window, or neither.



Write an algorithm that enables the robot, placed anywhere in the room, to count the number of windows. For example, in the room at right, the robot (marked as R) should find that it has two windows.

© Skip ODonnell/Stockphoto.



■ ■ ■ **P1.8** Consider a robot that has been placed in a maze. The right-hand rule tells you how to escape from a maze: Always have the right hand next to a wall, and eventually you will find an exit. The robot can:

- Move forward by one unit.
- Turn left or right.
- Sense what is in front of it: a wall, an exit, or neither.

Write an algorithm that lets the robot escape the maze. You may assume that there is an exit that is reachable by the right-hand rule. Your challenge is to deal with situations in which the path turns. The robot can't see turns. It can only see what is directly in front of it.

■ ■ ■ **Business P1.9** Suppose you received a loyalty promotion that lets you purchase one item, valued up to \$100, from an online catalog. You want to make the best of the offer. You have a list of all items for sale, some of which are less than \$100, some more. Write an algorithm to produce the item that is closest to \$100. If there is more than one such item, list them all. Remember that a computer will inspect one item at a time—it can't just glance at a list and find the best one.

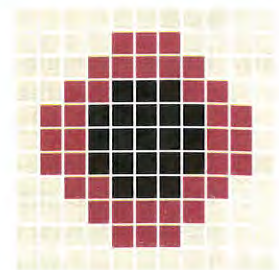
■ ■ ■ **Science P1.10** A television manufacturer advertises that a television set has a certain size, measured diagonally. You wonder how the set will fit into your living room. Write an algorithm that yields the horizontal and vertical size of the television. Your inputs are the diagonal size and the aspect ratio (the ratio of width to height, usually 16 : 9 for television sets).



© Don Bayley/Stockphoto.

■ ■ ■ **Science P1.11** Cameras today can correct “red eye” problems caused when the photo flash makes eyes look red. Write pseudocode for an algorithm that can detect red eyes. Your input is a pattern of colors, such as that at right.

You are given the number of rows and columns. For any row or column number, you can query the color, which will be red, black, or something else. If you find that the center of the black pixels coincides with the center of the red pixels, you have found a red eye, and your output should be “yes”. Otherwise, your output is “no”.










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. Nothing.
4. In secondary storage, typically a hard disk.
5. The central processing unit.
6. A smartphone has a CPU and memory, like any computer. A few smartphones have keyboards. Generally, the touchpad is used instead of a mouse. Secondary storage is in the form of a solid state drive. Of course, smartphones have a display, speaker, and microphone. The network connection uses the wireless radio to connect to a cell tower.

7. Safety and portability.
8. No one person can learn the entire library—it is too large.
9. The answer varies among systems. A typical answer might be `/home/dave/cs1/hello/HelloPrinter.java` or `c:\Users\Dave\Workspace\hello\HelloPrinter.java`
10. You back up your files and folders.
11. Change `World` to your name (here, Dave):

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

```
System.out.println("H");
System.out.println("e");
System.out.println("!");
System.out.println("!");
System.out.println("o");
```
13. No. The compiler would look for an item whose name is `Hello`. You need to enclose `Hello` in quotation marks:

```
System.out.println("Hello");
```
14. The printout is `My lucky number is12`. It would be a good idea to add a space after the `is`.
15. `Hello`
a blank line
`World`
16. This is a compile-time error. The compiler will complain that it does not know the meanings of the words `Hello` and `World`.
17. This is a compile-time error. The compiler will complain that `System.out` does not have a method called `println`.
18. This is a run-time error. It is perfectly legal to give the name `hello` to a method, so the compiler won't complain. But when the program is run, the virtual machine will look for a main method and won't find one.
19. It is a run-time error. After all, the program had been compiled in order for you to run it.
20. When a program has compiler errors, no class file is produced, and there is nothing to run.
21. 4 years:
0 10,000
1 12,000
2 14,400
3 17,280
4 20,736
22. Is the number of minutes at most 300?
a. If so, the answer is $\$29.95 \times 1.125 = \33.70 .
b. If not,
1. Compute the difference: (number of minutes) – 300.
2. Multiply that difference by 0.45.
3. Add \$29.95.
4. Multiply the total by 1.125. That is the answer.
23. No. The step *If it is more attractive than the "best so far"* is not executable because there is no objective way of deciding which of two photos is more attractive.
24. *Pick the first photo and call it "the most expensive so far".*
For each photo in the sequence
If it is more expensive than "the most expensive so far"
Discard "the most expensive so far".
Call this photo "the most expensive so far".
The photo called "the most expensive so far" is the most expensive photo in the sequence.
25. The first black marble that is preceded by a white one is marked in blue:

Switching the two yields

The next black marble to be switched is

yielding

The next steps are



Now the sequence is sorted.
26. The sequence doesn't terminate. Consider the input `0●0●0`. The first two marbles keep getting switched.