JAVA PROGRAMMING

FROM PROBLEM ANALYSIS TO PROGRAM DESIGN

EDITION 5

D.S. Malik

COURSE TECHNOLOGY
CENGAGE Learning™

Java Programming: From Problem Analysis to Program Design, Fifth Edition
D.S. Malik

Executive Editor: Marie Lee
Acquisitions Editor: Brandi Shailer
Senior Product Manager: Alyssa Pratt
Editorial Assistant: Jacqueline Lacaire
Content Project Manager: Lisa Weidenfeld
Associate Marketing Manager: Shanna Shelton
Art Director: Faith Brosnan
Proofreader: Andrea Schein
Indexer: Alexandra Nickerson
Print Buyer: Julio Esperas
Cover Designer: Roycroft Design/ www.roycroftdesign.com
Cover Photo: © photolibrary/Richard Cummins
Compositor: Integra

Some of the product names and company names used in this book have been used for identification purposes only and may be trademarks or registered trademarks of their respective manufacturers and sellers.

Any fictional data related to persons or companies or URLs used throughout this book is intended for instructional purposes only. At the time this book was printed, any such data was fictional and not belonging to any real persons or companies.

Course Technology, a part of Cengage Learning, reserves the right to revise this publication and make changes from time to time in its content without notice.

The programs in this book are for instructional purposes only. They have been tested with care, but are not guaranteed for any particular intent beyond educational purposes. The author and the publisher do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs.

Library of Congress Control Number: 2010940363

ISBN-13: 978-1-111-53053-2
ISBN-10: 1-111-53053-x

Course Technology
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil and Japan. Locate your local office at:
www.cengage.com/global

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Course Technology, visit
www.cengage.com/coursetechnology

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com**

Printed in the United States of America
1 2 3 4 5 6 7 16 15 14 13 12 11 10

CHAPTER 1

An Overview of Computers and Programming Languages

IN THIS CHAPTER, YOU WILL:

- Learn about different types of computers
- Explore the hardware and software components of a computer system
- Learn about the language of a computer
- Learn about the evolution of programming languages
- Examine high-level programming languages
- Discover what a compiler is and what it does
- Examine how a Java program is processed
- Learn about the Internet and World Wide Web
- Learn what an algorithm is and explore problem-solving techniques
- Become familiar with structured and object-oriented programming design methodologies

Introduction

Terms such as "the Internet," which was unfamiliar just a few years ago, are now common. Elementary school students regularly "surf" the Internet and use computers to design their classroom projects. Many people use the Internet to look up information and to communicate with others. These Internet activities are all made possible by the availability of different software, also known as computer programs. Software is developed by using programming languages. The Java programming language is especially well suited for developing software to accomplish specific tasks. Our main objective is to teach you how to write programs in the Java programming language. Before you begin programming, it is useful if you understand some of the basic terminology and different components of a computer. We begin with an overview of the history of computers.

An Overview of the History of Computers

The first device known to carry out calculations was the abacus. The abacus was invented in Asia but was used in ancient Babylon, China, and throughout Europe until the late middle ages. The abacus uses a system of sliding beads on a rack for addition and subtraction. In 1642, the French philosopher and mathematician Blaise Pascal invented the calculating device called the Pascaline. It had eight movable dials on wheels that could calculate sums up to eight figures long. Both the abacus and Pascaline could perform only addition and subtraction operations. Later in the seventeenth century, Gottfried von Leibniz invented a device that was able to add, subtract, multiply, and divide. In 1819, Joseph Jacquard, a French weaver, discovered that the weaving instructions for his looms could be stored on cards with holes punched in them. While the cards moved throughout the loom in sequence, needles passed through the holes and picked up threads of the correct color and texture. A weaver could rearrange the cards and change the pattern being woven. In essence, the cards programmed a loom to produce patterns in cloth. The weaving industry seems to have little in common with the computer industry. However, the idea of storing information by punching holes on a card turned out to be of great importance in the later development of computers.

In the early and mid-1800s, Charles Babbage, an English mathematician and physical scientist, designed two calculating machines—the difference engine and the analytical engine. The difference engine could automatically perform complex operations, such as squaring numbers. Babbage built a prototype of the difference engine, but did not build the actual device. The first complete difference engine was completed in London in 2002, 153 years after it was designed. It consists of 8,000 parts, weighs five tons, and measures 11 feet long. A replica of the difference engine was completed in 2008 and is on display at the Computer History Museum in Mountain View, California (*http://www.computerhistory.org/babbage/*). Most of Babbage's work is known through the writings of his colleague Ada Augusta, Countess of Lovelace. Augusta is considered to be the first computer programmer.

At the end of the 19th century, U.S. Census officials needed help in accurately tabulating the census data. Herman Hollerith invented a calculating machine that ran on electricity and used punched cards to store data. Hollerith's machine was immensely successful. Hollerith founded the Tabulating Machine Company, which later became the computer and technology corporation known as IBM.

The first computer-like machine was the Mark I. It was built, in 1944, jointly by IBM and Harvard University under the leadership of Howard Aiken. Punched cards were used to feed data into the machine. Mark I was 52 feet long, weighed 50 tons, and had 750,000 parts. In 1946, ENIAC (Electronic Numerical Integrator and Calculator) was built at the University of Pennsylvania. It contained 18,000 vacuum tubes and weighed some 30 tons.

The computers that we know today use the design rules given by John von Neumann in the late 1940s. His design included components such as arithmetic logic unit, control unit, memory, and input/output devices. These components are described in the next section. Von Neumann computer design makes it possible to store the programming instruction and the data in the same memory space. In 1951, the UNIVAC (Universal Automatic Computer) was built and sold to the U.S. Census Bureau.

In 1956, the invention of the transistors resulted in smaller, faster, more reliable, and more energy-efficient computers. This era also saw the emergence of the software development industry with the introduction of FORTRAN and COBOL, two early programming languages. In the next major technological advancement, transistors were replaced by tiny integrated circuits or "chips." Chips are smaller and cheaper than transistors and can contain thousands of circuits on a single chip. They give computers tremendous processing speed.

In 1970, the microprocessor, an entire CPU on a single chip, was invented. In 1977, Stephen Wozniak and Steven Jobs designed and built the first Apple computer in their garage. In 1981, IBM introduced its personal computer (PC). In the 1980s, clones of the IBM PC made the personal computer even more affordable. By the mid-1990s, people from many walks of life were able to afford them. Computers continue to become faster and less expensive as technology advances.

Modern-day computers are very powerful, reliable, and easy to use. They can accept spoken-word instructions and imitate human reasoning through artificial intelligence. Expert systems assist doctors in making diagnoses. Mobile computing applications are growing significantly. Using hand-held devices, delivery drivers can access global positioning satellites (GPS) to verify customer locations for pickups and deliveries. Cell phones can check your e-mail, make airline reservations, see how stocks are performing, and access your bank accounts.

Although there are several categories of computers, such as mainframe, midsize, and micro, all computers share some basic elements.

Elements of a Computer System

A computer is an electronic device capable of performing commands. The basic commands that a computer performs are input (get data), output (display results), storage, and performance of arithmetic and logical operations. There are two main components of a computer system—hardware and software. In the next few sections, we give a brief overview of these components. Let's look at hardware first.

Hardware

Major hardware components include the central processing unit (CPU); main memory (MM), also called random access memory (RAM); input/output devices; and secondary storage. Some examples of input devices are the keyboard, mouse, and secondary storage. Examples of output devices are the monitor, printer, and secondary storage.

CENTRAL PROCESSING UNIT AND MAIN MEMORY

The **central processing unit (CPU)** is the "brain" of the computer and the single most expensive piece of hardware in a computer. The more powerful the CPU, the faster the computer. Arithmetic and logical operations are carried out inside the CPU. Figure 1-1(a) shows some hardware components.

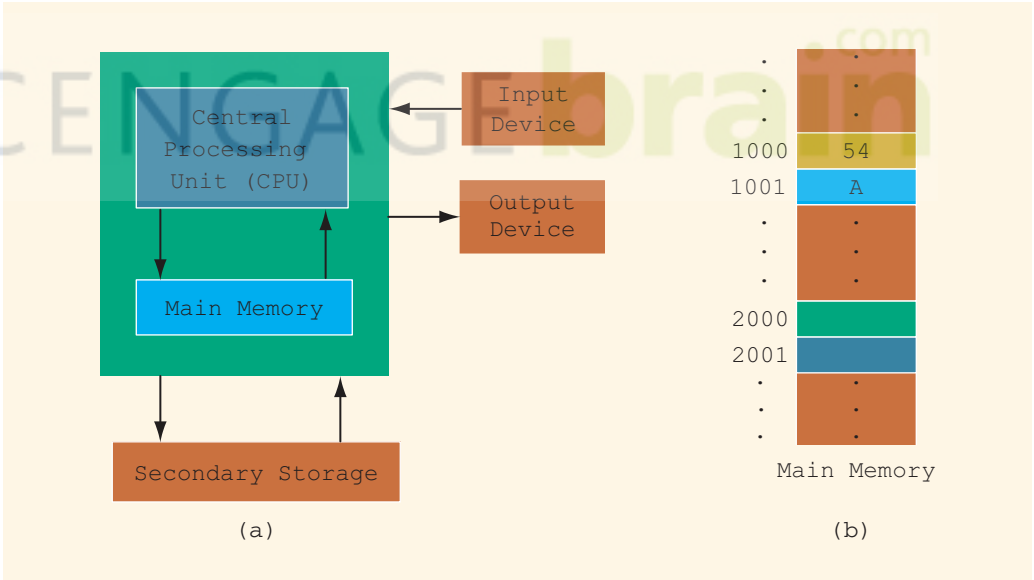


FIGURE 1-1 Hardware components of a computer and main memory

Main memory, or random access memory (RAM), is connected directly to the CPU. All programs must be loaded into main memory before they can be executed. Similarly,

all data must be brought into main memory before a program can manipulate it. When the computer is turned off, everything in main memory is lost.

Main memory is an ordered sequence of cells, called **memory cells**. Each cell has a unique location in main memory, called the **address** of the cell. These addresses help you access the information stored in the cell. Figure 1-1(b) shows main memory with some data.

Today's computers come with main memory consisting of millions to billions of cells. Although Figure 1-1(b) shows data stored in cells, the content of a cell can be either a programming instruction or data. Moreover, this figure shows the data as numbers and letters. However, as explained later in this chapter, main memory stores everything as sequences of `0`s and `1`s. The memory addresses are also expressed as sequences of `0`s and `1`s.

SECONDARY STORAGE

Because programs and data must be stored in main memory before processing, and because everything in main memory is lost when the computer is turned off, information stored in main memory must be transferred to some other device for longer-term storage. A device that stores longer-term information (unless the device becomes unusable or you change the information by rewriting it) is called **secondary storage**. To be able to transfer information from main memory to secondary storage, these components must be connected directly to each other. Examples of secondary storage are hard disks, floppy disks, flash memory, ZIP disks, CD-ROMs, and tapes.

INPUT/OUTPUT DEVICES

For a computer to perform a useful task, it must be able to take in data and programs and display the results of the manipulation of the data. The devices that feed data and programs into computers are called **input devices**. The keyboard, mouse, and secondary storage are examples of input devices. The devices that the computer uses to display and store results are called **output devices**. A monitor, printer, and secondary storage are examples of output devices. Figure 1-2 shows some input and output devices.

FIGURE 1-2 Some input and output devices

Software

Software consists of programs written to perform specific tasks. For example, you use word-processing programs to write letters, papers, and books. The two types of programs are system programs and application programs.

System programs control the computer. The system program that loads first when you turn on your PC is called the **operating system**. Without an operating system, the computer is useless. The operating system monitors the overall activity of the computer and provides services, such as memory management, input/output activities, and storage management. The operating system has a special program that organizes secondary storage so that you can access information conveniently. The operating system is the program that runs the application programs. **Application programs** perform specific tasks. Word processors, spreadsheets, and games are examples of application programs. Both operating systems and application programs are written in programming languages.

Language of a Computer

When you press A on your keyboard, the computer displays A on the screen, but what is actually stored inside the computer's main memory? What is the language of the computer? How does it store whatever you type on the keyboard?

Remember that a computer is an electronic device. Electrical signals move along channels inside the computer. There are two types of electrical signals: analog and digital. **Analog signals** are continuous waveforms used to represent things, such as sound. Audio tapes, for example, store data in analog signals. **Digital signals** represent information with a sequence of `0`s and `1`s. A `0` represents a low voltage, and a `1` represents a high voltage. Digital signals are more reliable carriers of information than analog signals and can be copied from one device to another with exact precision. You might have noticed that when you make a copy of an audio tape, the sound quality of the copy is not as good as that on the original tape. Computers use digital signals.

Because digital signals are processed inside a computer, the language of a computer, called **machine language**, is a sequence of `0`s and `1`s. The digit `0` or `1` is called a **binary digit**, or **bit**. Sometimes a sequence of `0`s and `1`s is referred to as a **binary code** or a **binary number**.

Bit: A binary digit `0` or `1`.

A sequence of eight bits is called a **byte**. Moreover, $2^{10} = 1024$ bytes and is called a **kilobyte (KB)**. Table 1-1 summarizes the terms used to describe the various numbers of bytes.

TABLE 1-1 Binary Units

Unit	Symbol	Bits/Bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	1024 KB = 2^{10} KB = 2^{20} bytes = 1,048,576 bytes
Gigabyte	GB	1024 MB = 2^{10} MB = 2^{30} bytes = 1,073,741,824 bytes
Terabyte	TB	1024 GB = 2^{10} GB = 2^{40} bytes = 1,099,511,627,776 bytes
Petabyte	PB	1024 TB = 2^{10} TB = 2^{50} bytes = 1,125,899,906,842,624 bytes
Exabyte	EB	1024 PB = 2^{10} PB = 2^{60} bytes = 1,152,921,504,606,846,976 bytes
Zettabyte	ZB	1024 EB = 2^{10} EB = 2^{70} bytes = 1,180,591,620,717,411,303,424 bytes

Every letter, number, or special symbol (such as `*` or `{`) on your keyboard is encoded as a sequence of bits, each having a unique representation. The most commonly used encoding scheme on personal computers is the seven-bit **American Standard Code for Information Interchange (ASCII)**. The ASCII data set consists of 128 characters, numbered 0 through 127. (Note that $2^7 = 128$ and $2^8 = 256$.) That is, in the ASCII data set, the position of the first character is 0, the position of the second character is 1, and so on. In this scheme, `A` is encoded as `1000001`. In fact, `A` is the 66th character in the ASCII character code, but its position is 65 because the position of the first character is 0. Furthermore, `1000001` is the binary representation of `65`. The character `3` is encoded as `0110011`. For a complete list of the printable ASCII character set, refer to Appendix C.

NOTE

The number system that we use in our daily life is called the **decimal system** or **base 10**. Because everything inside a computer is represented as a sequence of 0s and 1s, that is, binary numbers, the number system that a computer uses is called binary or **base 2**. We indicated in the preceding paragraph that the number 1000001 is the binary representation of 65. Appendix D describes how to convert a number from base 10 to base 2 and vice versa. Appendix D also describes how to convert a number between base 2 and base 16 (hexadecimal) and between base 2 and base 8 (octal).

Inside the computer, every character is represented as a sequence of eight bits, that is, as a byte. Because ASCII is a seven-bit code, you must add `0` to the left of the ASCII encoding of a character. Hence, inside the computer, the character `A` is represented as `01000001`, and the character `3` is represented as `00110011`.

Other encoding schemes include Unicode, which is a more recent development. **Unicode** consists of 65,536 characters. To store a Unicode character, you need two bytes. Java uses the Unicode character set. Therefore, in Java, every character is represented as a sequence of 16 bits, that is, 2 bytes. In Unicode, the character `A` is represented as `0000000001000001`.

The ASCII character set is a subset of Unicode; the first 128 characters of Unicode are the same as the characters in ASCII. If you are dealing with only the English language, the ASCII character set is sufficient to write Java programs. The advantage of the Unicode character set is that symbols from languages other than English can be handled easily.

Evolution of Programming Languages

The most basic computer language, machine language, provides program instructions in bits. Even though most computers perform the same kinds of operations, the designers of different CPUs sometimes choose different sets of binary codes to perform those operations. Therefore, the machine language of one computer is not necessarily the same as the machine language of another computer. The only consistency among computers is that in any computer, all data are stored and manipulated as a binary code.

Early computers were programmed in machine language. To see how instructions are written in machine language, suppose you want to use the equation:

```
wages = rate · hours
```

to calculate weekly wages. Assume that the memory locations of `rate`, `hours`, and `wages` are `010001`, `010010`, and `010011`, respectively. Further suppose that the binary code `100100` stands for load, `100110` stands for multiplication, and `100010` stands for store. In machine language, you might need the following sequence of instructions to calculate the weekly wages:

```
100100 010001
100110 010010
100010 010011
```

To represent the weekly wages equation in machine language, the programmer had to remember the machine language codes for various operations. Also, to manipulate data, the programmer had to remember the locations of the data in main memory. Remembering specific codes made programming difficult and error prone.

Assembly languages were developed to make the programmer's job easier. In **assembly language**, an instruction is an easy-to-remember form called a **mnemonic**. Table 1-2 shows some examples of instructions in assembly language and their corresponding machine language code.

TABLE 1-2 Examples of Instructions in Assembly Language and Machine Language

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

Using assembly language instructions, you can write the equation to calculate the weekly wages as follows:

```
LOAD   rate
MULT   hours
STOR   wages
```

As you can see, it is much easier to write instructions in assembly language. However, a computer cannot execute assembly language instructions directly. The instructions first have to be translated into machine language. A program called an **assembler** translates the assembly language instructions into machine language.

Assembler: A program that translates a program written in assembly language into an equivalent program in machine language.

Moving from machine language to assembly language made programming easier, but a programmer was still forced to think in terms of individual machine instructions. The next step toward making programming easier was to devise **high-level languages** that were closer to spoken languages, such as English and Spanish. Basic, FORTRAN, COBOL, Pascal, C, C++, and Java are all high-level languages. You will learn the high-level language Java in this book.

In Java, you write the weekly wages equation as follows:

```
wages = rate * hours;
```

The instruction written in Java is much easier to understand and is self-explanatory to a novice user who is familiar with basic arithmetic. As in the case of assembly language, however, the computer cannot directly execute instructions written in a high-level language. To run on a computer, these Java instructions first need to be translated into an intermediate language called **bytecode** and then interpreted into a particular machine language. A program called a **compiler** translates instructions written in Java into bytecode.

Compiler: A program that translates a program written in a high-level language into the equivalent machine language. (In the case of Java, this machine language is the bytecode.)

Recall that the computer understands only machine language. Moreover, different types of CPUs use different machine languages. To make Java programs **machine independent**, that is, able to run on many different types of computer platforms, the designers of Java introduced a hypothetical computer called the **Java Virtual Machine (JVM)**. In fact, bytecode is the machine language for the JVM.

NOTE

In languages such as C and C++, the compiler translates the source code directly into the machine language of your computer's CPU. For such languages, a different compiler is needed for each type of CPU. Therefore, programs in these languages are not easily portable from one type of machine to another. The source code must be recompiled for each type of CPU. To make Java programs machine independent and easily portable, and to allow them to run on a Web browser, the designers of Java introduced the Java Virtual Machine (JVM) and bytecode as the (machine) language of this machine. It is easier to translate a bytecode into a particular type of CPU. This concept is covered further in the following section, Processing a Java Program.

Processing a Java Program

Java has two types of programs—applications and applets. The following is an example of a Java application program:

```
public class MyFirstJavaProgram
{
    public static void main(String[] args)
    {
        System.out.println("My first Java program.");
    }
}
```

At this point you need not be too concerned with the details of this program. However, if you run (execute) this program, it will display the following line on the screen:

```
My first Java program.
```

Recall that a computer can understand only machine language. Therefore, in order to run this program successfully, the code must first be translated into the machine language. In this section we review the steps required to execute programs written in Java.

To process a program written in Java, you carry out the following steps, as illustrated in Figure 1-3.

1. You use a text editor, such as Notepad, to create (that is, type) a program in Java following the rules, or syntax, of the language. This program is called the **source program**. The program must be saved in a text file named `ClassName.java`, where `ClassName` is the name of the Java class contained in the file. For example, in the Java program

given above, the name of the (`public`) class containing the Java program is `MyFirstJavaProgram`. Therefore, this program must be saved in the text file named `MyFirstJavaProgram.java`. Otherwise an error will occur.

Source program: A program written in a high-level language.

2. You must verify that the program obeys the rules of the programming language—that is, the program must be syntactically correct—and translate the program into the equivalent bytecode. The compiler checks the source program for syntax errors and, if no error is found, translates the program into bytecode. The bytecode is saved in the file with the `.class` extension. For example, the bytecode for `MyFirstJavaProgram.java` is stored in the `MyFirstJavaProgram.class` file by the compiler.
3. To run a Java application program, the `.class` file must be loaded into computer memory. To run a Java applet, you must use either a Web browser or an applet viewer, a stripped-down Web browser for running applets. The programs that you write in Java are typically developed using an **integrated development environment (IDE)**. The IDE contains many programs that are useful in creating your program. For example, it contains the necessary code to display the results of the program and several mathematical functions to make the programmer's job somewhat easier. Because certain code is already available to you, you can use this code rather than writing your own. You can also develop your own libraries (called *packages* in Java). (Note that in Java, typically, a package is a set of related classes. So, typically, a Java program is a collection of classes. We will explain this further in Chapters 2 and 8. At this point, you need not be too concerned with these details.) In general, to successfully run a Java program, the bytecode for classes used in the program must be connected. The program that automatically does this in Java is known as the **loader**.
4. The next step is to execute the Java program. In addition to connecting the bytecode from various classes, the loader also loads your Java program's bytecode into main memory. As the classes are loaded into main memory, the *bytecode verifier* verifies that the bytecode for the classes is valid and does not violate Java's security restrictions. Finally, a program called an **interpreter** translates each bytecode instruction into your computer's machine language, and then executes it.

   Interpreter: A program that reads and translates each bytecode instruction into your computer's machine language, and then executes it.

Note that the Java interpreter translates and executes one bytecode instruction at a time. It does not first translate the entire bytecode into your computer's machine language. As noted earlier, in languages such as C++, a different compiler is needed for each type of CPU, whereas a Java compiler translates a Java source program into bytecode, the machine language of JVM, which is independent of any particular type of CPU.

The Java interpreter translates each bytecode instruction into a particular type of CPU machine language and then executes the instruction. Thus, in the case of the Java language, a different type of interpreter is needed for a particular type of CPU. However, interpreters are programs that are simpler than compilers. Because the Java interpreter translates one bytecode instruction at a time, Java programs run more slowly.

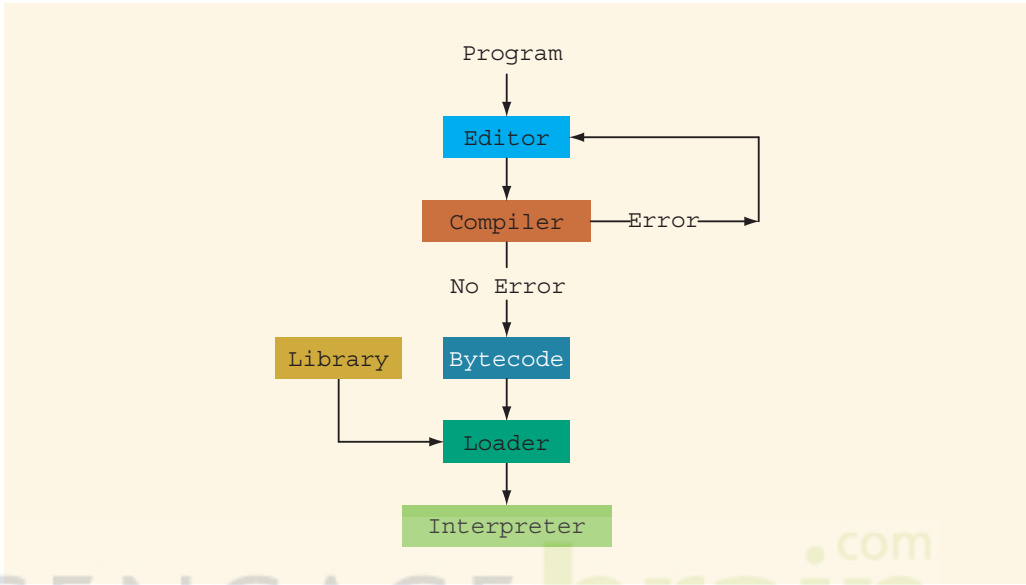


FIGURE 1-3 Processing a Java program

As a programmer, one of your primary concerns is with Step 1. That is, you must learn, understand, and master the rules of the programming language to create source programs. Programs are developed using an IDE. Well-known IDEs used to create programs in Java include JBuilder (from Borland), CodeWarrior (Metrowerks), and jGrasp (Auburn University). These IDEs contain an editor to create the program, a compiler to check the program for syntax errors, a program to load the object codes of the resources used from the IDE, and a program to execute the program. These IDEs are also quite user friendly. When you compile your program, the compiler not only identifies the syntax errors, but also typically suggests how to correct them.

NOTE Other software that can be used to develop Java programs include Eclipse, TextPad, JCreator, BlueJ, and DrJava.

In Chapter 2, after being introduced to some basic elements of Java, you will see how a Java program is created.

Internet, World Wide Web, Browser, and Java

We often hear the terms Internet, World Wide Web (or simply, Web) and Web browser (or simply, browser). What do these terms mean, and what is Java's connection with them?

The *Internet* is an interconnection of networks that allows computers around the world to communicate with each other. In 1969, the U.S. Department of Defense's Advanced Research Project Agency (ARPA) funded research projects to investigate and develop techniques and technologies to interlink networks. The objective was to develop communication protocols so that networked computers could communicate with each other. This was called the *internetting* project, and the funding resulted into ARPANET, which eventually became known as the "Internet."

Over the last four decades, the Internet has grown manyfold. In 1973, approximately 25 computers were connected via the Internet. This number grew to 700,000 computers by 1991, and to over 10,000,000 by 2000. Each day, more and more computers are getting connected via the Internet.

The terms Internet and World Wide Web are often used interchangeably. However, there is a difference between the two. The Internet allows computers to be connected and communicate with each other. On the other hand, the *World Wide Web* (WWW), or Web, uses software programs that enable computer users to access documents and files (including images, audio, and video) on almost any subject over the Internet with the click of a mouse. Undoubtedly, the Internet has become one of the world's leading communication mechanisms. Computers around the world communicate via the Internet; the World Wide Web makes that communication a fun activity.

The primary language for the Web is known as *Hypertext Markup Language* (HTML). It is a simple language for laying out and linking documents, as well as for viewing images and listening to sound. However, HTML is not capable of interacting with the user, except to collect information via simple forms. Therefore, Web pages are essentially static. As noted previously, Java has two types of programs—applications and applets. In terms of programming, both types are similar. Application programs are stand-alone programs that can run on your computer. Java applets are programs that run from a *Web browser* and make the Web responsive and interactive. Two well-known *browsers* are Mozilla Firefox and Internet Explorer. Java applets can run in either browser. Moreover, through the use of applets, the Web becomes responsive, interactive, and fun to use. (Note that to run applets, the browser you use must be Java enabled.)

Programming with the Problem Analysis–Coding–Execution Cycle

Programming is a process of problem solving. Different people use different techniques to solve problems. Some techniques are clearly outlined and easy to follow; they solve the problem and give insight into how the solution was reached. Such problem-solving techniques can be easily modified if the domain of the problem changes.

To be a skillful problem solver, and, therefore, to become a skillful programmer, you must use good problem-solving techniques. One common problem-solving technique includes analyzing a problem, outlining the problem requirements, and designing steps, called an **algorithm**, to solve the problem.

Algorithm: A step-by-step problem-solving process in which a solution is arrived at in a finite amount of time.

In the programming environment, the problem-solving process involves the following steps:

1. Analyze the problem and outline the problem and its solution requirements.
2. Design an algorithm to solve the problem.
3. Implement the algorithm in a programming language, such as Java.
4. Verify that the algorithm works.
5. Maintain the program by using and improving it, and modifying it if the problem domain changes.

Figure 1-4 summarizes this programming process.

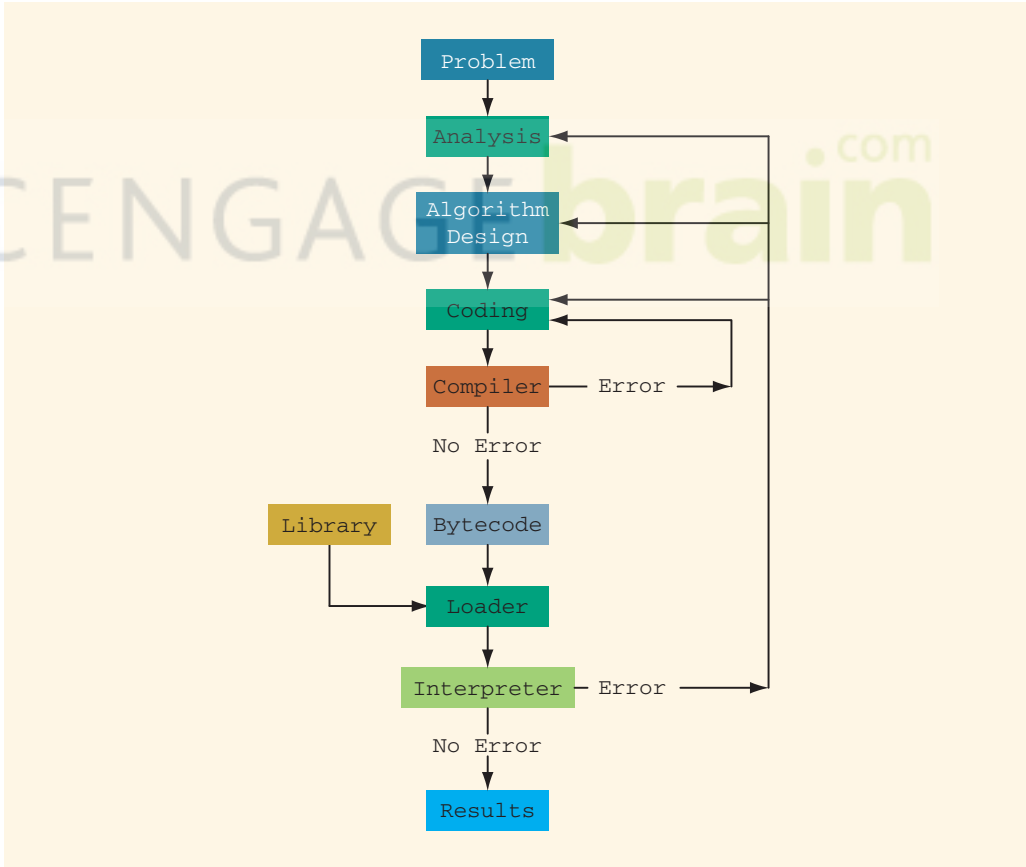


FIGURE 1-4 Problem analysis–coding–execution cycle

To develop a program to solve a problem, you start by analyzing the problem, then outlining the problem and the options for a solution. You then design the algorithm; write the program instructions in a high-level language, or code the program; and enter the program into a computer system.

Analyzing the problem is the first, and most important, step in the process. This step requires that you do the following:

- Thoroughly understand the problem.
- Understand the problem requirements. Requirements can include whether the program requires interaction with the user, whether it manipulates data, whether it produces output, and what the output looks like. If the program manipulates data, the programmer must know what the data are and how they are represented. To do this, you need to look at sample data.
- If the program produces output, you should know how the results should be generated and formatted.
- If the problem is complex, divide the problem into subproblems and repeat Steps 1 and 2 by analyzing each subproblem and understanding each subproblem's requirements. You also need to know how the subproblems relate to each other.

After you carefully analyze the problem, the next step is to design an algorithm to solve it. If you broke the problem into subproblems, you need to design an algorithm for each subproblem. Once you design an algorithm, you must check it for correctness. You can sometimes do this using sample data; other times, you might need to perform some mathematical analysis to test the algorithm's correctness. You also need to integrate the subproblem solutions.

Once you have designed the algorithm and verified its correctness, the next step is to convert the algorithm into a high-level language. You use a text editor to enter the program into a computer, making sure that the program follows the language's syntax. To verify the correctness of the syntax, you run the code through a compiler. If the compiler generates error messages, you must identify the errors in the code, resolve them, and then run the code through the compiler again. When all syntax errors are removed, the compiler generates the machine code (bytecode in Java).

The final step is to execute the program. The compiler guarantees only that the program follows the language's syntax; it does not guarantee that the program will run correctly. During execution, the program might terminate abnormally due to logical errors, such as division by zero. Even if the program terminates normally, it may still generate erroneous results. Under these circumstances, you may have to reexamine the code, the algorithm, or even your analysis of the problem.

Your overall programming experience will benefit if you spend enough time to thoroughly complete the problem analysis before attempting to write the programming instructions. Usually, you do this work on paper using a pen or pencil. Taking this careful approach to programming has a number of advantages. It is much easier to discover errors in a program that is well analyzed and well designed. Furthermore, a

thoroughly analyzed and carefully designed program is much easier to follow and modify. Even the most experienced programmers spend a considerable amount of time analyzing a problem and designing an algorithm.

Throughout this book, you will learn not only the rules of writing programs in Java, but also problem-solving techniques. Each chapter discusses several programming problems, each of which is clearly marked as a Programming Example. The Programming Examples teach techniques to analyze and solve the problems and also help you understand the concepts discussed in the chapter. To gain the full benefit of this book, we recommend that you work through the Programming Examples at the end of each chapter.

EXAMPLE 1-1

In this example, we design an algorithm to find the perimeter and area of a rectangle.

To find the perimeter and area of a rectangle, you need to know the rectangle's length and width. The perimeter and area of the rectangle are then given by the following formulas:

```
perimeter = 2 · (length + width)
area = length · width
```

The algorithm to find the perimeter and area of the rectangle is:

1. Get the length of the rectangle.
2. Get the width of the rectangle.
3. Find the perimeter using the following equation:

   ```
   perimeter = 2 · (length + width)
   ```
4. Find the area using the following equation:

   ```
   area = length · width
   ```

EXAMPLE 1-2

In this example, we design an algorithm that calculates the monthly paycheck of a salesperson at a local department store.

Every salesperson has a base salary. The salesperson also receives a bonus at the end of each month, based on the following criteria: If the salesperson has been with the store for five years or less, the bonus is $10 for each year that he or she has worked there. If the salesperson has been with the store for more than five years, the bonus is $20 for each year that he or she has worked there. The salesperson can earn an additional bonus as follows: If the total sales made by the salesperson for the month are greater than or equal to $5,000 but less than $10,000, he or she receives a 3% commission on the sale. If the total sales made by the salesperson for the month are at least $10,000, he or she receives a 6% commission on the sale.

To calculate a salesperson's monthly paycheck, you need to know the base salary, the number of years that the salesperson has been with the company, and the total sales made by the salesperson for that month. Suppose `baseSalary` denotes the base salary, `noOfServiceYears` denotes the number of years that the salesperson has been with the store, `bonus` denotes the bonus, `totalSales` denotes the total sales made by the salesperson for the month, and `additionalBonus` denotes the additional bonus.

You can determine the bonus as follows:

```
if (noOfServiceYears is less than or equal to five)
    bonus = 10 · noOfServiceYears
otherwise
    bonus = 20 · noOfServiceYears
```

Next, you can determine the additional bonus of the salesperson as follows:

```
if (totalSales is less than 5000)
    additionalBonus = 0
otherwise
      if (totalSales is greater than or equal to 5000 and
                        totalSales is less than 10000)
            additionalBonus = totalSales · (0.03)
      otherwise
            additionalBonus = totalSales · (0.06)
```

Following the above discussion, you can now design the algorithm to calculate a salesperson's monthly paycheck:

1. Get `baseSalary`.
2. Get `noOfServiceYears`.
3. Calculate `bonus` using the following formula:

```
if (noOfServiceYears is less than or equal to five)
     bonus = 10 · noOfServiceYears
otherwise
     bonus = 20 · noOfServiceYears
```

4. Get `totalSales`.
5. Calculate `additionalBonus` using the following formula:

```
if (totalSales is less than 5000)
    additionalBonus = 0
otherwise
   if (totalSales is greater than or equal to 5000 and
        totalSales is less than 10000)
      additionalBonus = totalSales · (0.03)
otherwise
      additionalBonus = totalSales · (0.06)
```

6. Calculate `payCheck` using the following equation:

```
payCheck = baseSalary + bonus + additionalBonus
```

EXAMPLE 1-3

In this example, we design an algorithm to play a number-guessing game.

The objective is to randomly generate an integer greater than or equal to `0` and less than `100`. Then, prompt the player (user) to guess the number. If the player guesses the number correctly, output an appropriate message. Otherwise, check whether the guessed number is less than the random number. If the guessed number is less than the random number generated, output the message, "Your guess is lower than the number. Guess again!"; otherwise, output the message, "Your guess is higher than the number. Guess again!". Then, prompt the player to enter another number. The player is prompted to guess the random number until the player enters the correct number.

The first step is to generate a random number, as described above. Java provides the means to do so, which is discussed in Chapter 5. Suppose `num` stands for the random number and `guess` stands for the number guessed by the player.

After the player enters the `guess`, you can compare the `guess` with the random number as follows:

```
if (guess is equal to num)
    Print "You guessed the correct number."
otherwise
    if guess is less than num
       Print "Your guess is lower than the number. Guess again!"
otherwise
       Print "Your guess is higher than the number. Guess again!"
```

You can now design an algorithm as follows:

1. Generate a random number and call it `num`.
2. *Repeat* the following steps until the player has guessed the correct number:
   a. Prompt the player to enter `guess`.
   b.
```
if (guess is equal to num)
    Print "You guessed the correct number."
otherwise
    if guess is less than num
       Print "Your guess is lower than the number. Guess again!"
otherwise
       Print "Your guess is higher than the number. Guess again!"
```

In Chapter 5, we write a program that uses this algorithm to play the number-guessing game.

The type of coding used in Examples 1-1 to 1-3 is called **pseudocode**, which is an "outline" of a program that could be translated into actual code. Pseudocode is not written in a particular language, nor does it have syntax rules; it is mainly a technique to show the programming steps.

Programming Methodologies

Two popular approaches to programming design are the structured approach and the object-oriented approach, which are outlined below.

Structured Programming

Dividing a problem into smaller subproblems is called **structured design**. Each subproblem is then analyzed, and a solution for the subproblem is obtained. The solutions to all the subproblems are then combined to solve the overall problem. This process of implementing a structured design is called **structured programming**. The structured design approach is also known as **top-down design**, **bottom-up design**, **stepwise refinement**, and **modular programming**.

Object-Oriented Programming

Object-oriented design (OOD) is a widely used programming methodology. In OOD, the first step in the problem-solving process is to identify the components called **objects**, which form the basis of the solution, and to determine how these objects interact with one another. For example, suppose you want to write a program that automates the video rental process for a local video store. The two main objects in this problem are the video and the customer.

After identifying the objects, the next step is to specify for each object the relevant data and possible operations to be performed on that data. For example, for a video object, the *data* might include:

- movie name
- starring actors
- producer
- production company
- number of copies in stock

Some of the operations on a video object might include:

- checking the name of the movie
- reducing the number of copies in stock by one after a copy is rented
- incrementing the number of copies in stock by one after a customer returns a copy

This illustrates that each **object** consists of data and the operations on those data. An object combines data and operations on that data into a single unit. In OOD, the final program is a collection of interacting objects. A programming language that implements OOD is called an **object-oriented programming (OOP)** language. You will learn about the many advantages of OOD in later chapters.

Because an object consists of data and operations on the data, before you can design and use objects, you need to learn how to represent data in computer memory, how to manipulate data, and how to implement operations. In Chapter 2, you will learn the basic data types of Java and discover how to represent and manipulate data in computer memory. Chapter 3 discusses how to input data into a Java program and output the results generated by a Java program.

To create operations, you write algorithms and implement them in a programming language. Because a data element in a complex program usually has many operations, to separate operations from each other and use them effectively and in a convenient manner, you use **methods** to implement algorithms. You will learn the details of methods in Chapter 7. Certain algorithms require that a program make decisions, a process called selection. Other algorithms might require that certain statements be repeated until certain conditions are met, a process called repetition. Still other algorithms might require both selection and repetition. You will learn about selection and repetition mechanisms, called control structures, in Chapters 4 and 5.

Finally, to work with objects, you need to know how to combine data and operations on that data into a single unit. In Java, the mechanism that allows you to combine data and operations on the data into a single unit is called a **class**. In Chapter 8, you will learn how to create your own classes.

In Chapter 9, using a mechanism called an array, you will learn how to manipulate data when data items are of the same type, such as the items in a list of sales figures. As you can see, you need to learn quite a few things before working with the OOD methodology.

For some problems, the structured approach to program design is very effective. Other problems are better addressed by OOD. For example, if a problem requires manipulating sets of numbers with mathematical functions, you might use the structured design approach and outline the steps required to obtain the solution. The Java library supplies a wealth of functions that you can use to manipulate numbers effectively. On the other hand, if you want to write a program that would make a candy machine operational, the OOD approach is more effective. Java was designed especially to implement OOD. Furthermore, OOD works well and is used in conjunction with structured design. Chapter 6 explains how to use an existing class to create a Graphical User Interface (GUI) and then gives several examples explaining how to solve problems using OOD concepts.

Both the structured design and OOD approaches require that you master the basic components of a programming language to be an effective programmer. In the next few chapters, you will learn the basic components of Java required by either approach to programming.

QUICK REVIEW

1. A computer is an electronic device capable of performing arithmetic and logical operations.
2. A computer system has two kinds of components: hardware and software.
3. The central processing unit (CPU) and the main memory are examples of hardware components.
4. All programs must be brought into main memory before they can be executed.
5. When the power to the computer is switched off, everything in main memory is lost.
6. Secondary storage provides permanent storage for information. Hard disks, floppy disks, flash-memory, ZIP disks, CD-ROMs, and tapes are examples of secondary storage.
7. Input to the computer is done via an input device. Two common input devices are the keyboard and the mouse.
8. The computer sends output to an output device, such as the computer monitor.
9. Software refers to programs run by the computer.
10. The operating system monitors the overall activity of the computer and provides services.
11. Application programs perform a specific task.
12. The most basic language of a computer is a sequence of `0`s and `1`s called machine language. Every computer directly understands its own machine language.
13. A bit is a binary digit, `0` or `1`.
14. A sequence of `0`s and `1`s is called a binary code or a binary number.
15. A byte is a sequence of eight bits.
16. One kilobyte (KB) is $2^{10} = 1024$ bytes; one megabyte (MB) is $2^{20} = 1{,}048{,}576$ bytes; one gigabyte (GB) is $2^{30} = 1{,}073{,}741{,}824$ bytes; one terabyte (TB) is $2^{40} = 1{,}099{,}511{,}627{,}776$ bytes; one petabyte (PB) is $2^{50} = 1{,}125{,}899{,}906{,}842{,}624$ bytes; one exabyte (EB) is $2^{60} = 1{,}152{,}921{,}504{,}606{,}846{,}976$ bytes; and one zettabyte (ZB) is $2^{70} = 1{,}180{,}591{,}620{,}717{,}411{,}303{,}424$ bytes.
17. Assembly language uses easy-to-remember instructions called mnemonics.
18. Assemblers are programs that translate a program written in assembly language into machine language.
19. To run a Java program on a computer, the program must first be translated into an intermediate language called bytecode and then interpreted into a particular machine language.

20. To make Java programs machine independent, the designers of the Java language introduced a hypothetical computer called the Java Virtual Machine (JVM).
21. Bytecode is the machine language for the JVM.
22. Compilers are programs that translate a program written in a high-level language into an equivalent machine language. In the case of Java, this machine language is the bytecode.
23. In Java, the necessary steps to process a program are edit, compile, load, and execute.
24. A Java loader transfers into main memory the bytecode of the classes needed to execute the program.
25. An interpreter is a program that reads, translates each bytecode instruction into the machine language of your computer, and then executes it.
26. The Internet is a network of networks through which computers around the world are connected.
27. The World Wide Web, or Web, uses software programs that allow computer users to view documents on almost any subject over the Internet with the click of a mouse.
28. Java application programs are stand-alone programs that can run on your computer. Java applets are programs that run from a Web browser, or simply a browser.
29. A problem-solving process for programming has five steps: analyze the problem, design an algorithm, implement the algorithm in a programming language, verify that the algorithm works, and maintain the program.
30. An algorithm is a step-by-step problem-solving process in which a solution is arrived at in a finite amount of time.
31. The two basic approaches to programming design are structured design and object-oriented design.
32. In structured design, a problem is divided into smaller subproblems. Each subproblem is solved, and the subproblem solutions are integrated.
33. In object-oriented design (OOD), the programmer identifies components called objects, which form the basis of the solution, and determines how these objects interact with one another. In OOD, a program is a collection of interacting objects.
34. An object consists of data and the operations on those data.

EXERCISES

1. Mark the following statements as true or false.
   a. The first device known to carry out calculations was the Pascline.
   b. Modern-day computers can accept spoken-word instructions, but cannot imitate human reasoning.
   c. In Unicode, every character is coded as a sequence of sixteen bits.
   d. The arithmetic operations are performed inside the CPU and, if an error is found, it outputs the logical errors.
   e. A sequence of 0s and 1s is called a decimal code.
   f. A Java compiler is a program that translates a Java program into bytecode.
   g. Bytecode is the machine language of the JVM.
   h. The CPU stands for command performing unit.
   i. RAM stands for readily available memory.
   j. A program written in a high-level programming language is called a source program.
   k. ZB stands for zero byte.
   l. The first step in the problem-solving process is to analyze the problem.
2. Name two input devices.
3. Name two output devices.
4. Why is secondary storage needed?
5. What is the function of an operating system?
6. What are the two types of programs?
7. What are the differences between machine languages and high-level languages?
8. What is a source program?
9. What kind of errors are reported by a compiler?
10. Why do you need to translate a program written in a high-level language into machine language?
11. Why would you prefer to write a program in a high-level language rather than a machine language?
12. What are the advantages of problem analysis and algorithm design over directly writing a program in a high-level language?
13. Design an algorithm to find the weighted average of four test scores. The four test scores and their respective weights are given in the following format:

```
testScore1 weightTestScore1
...
```

For example, a sample data is as follows:

```
75 0.20
95 0.35
85 0.15
65 0.30
```

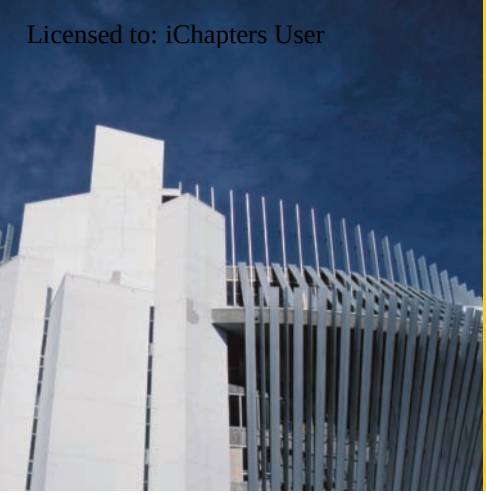
14. Given the radius, in inches, and price of a pizza, design an algorithm and write the pseudocode to find the price of the pizza per square inch.
15. To make a profit, the prices of the items sold in a furniture store are marked up by 80%. After marking up the prices each item is put on sale at a discount of 10%. Design an algorithm to find the selling price of an item sold at the furniture store. What information do you need to find the selling price?
16. Suppose a, b, and c denote the lengths of the sides of a triangle. Then, the area of the triangle can be calculated using the formula:

$$\sqrt{s(s-a)(s-b)(s-c)}$$

where $s = (1/2)(a + b + c)$. Design an algorithm that uses this formula to find the area of a triangle. What information do you need to find the area?

17. Suppose that the cost of sending an international fax is calculated as follows: Service charges \$3.00, \$0.20 per page for the first 10 pages, and \$0.10 for each additional page. Design an algorithm that asks the user to enter the number of pages to be faxed. The algorithm then uses the number of pages to be faxed to calculate the amount due.
18. You are given a list of students' names and their test scores. Design an algorithm that does the following:
    a. Calculates the average test scores.
    b. Determines and prints the names of all the students whose test score is below the average test score.
    c. Determines the highest test score.
    d. Prints the names of all the students whose test score is the same as the highest test score.

    (You must divide this problem into subproblems as follows: The first subproblem determines the average test score. The second subproblem determines and prints the names of all the students whose test score is below the average test score. The third subproblem determines the highest test score. The fourth subproblem prints the names of all the students whose test score is the same as the highest test score. The main algorithm combines the solutions of the subproblems.)



APPENDIX E
ANSWERS TO ODD-NUMBERED EXERCISES

Chapter 1

1. a. False; b. False; c. True; d. False; e. False; f. True; g. True; h. False; i. False; j. True; k. False; l. True
3. Monitor and printer.
5. An operating system monitors the overall activity of the computer and provides services. Some of these services include memory management, input/output activities, and storage management.
7. In machine language the programs are written using the binary codes while in high-level language the programs are closer to the natural language. For execution, a high-level language program is translated into the machine language while a machine language need not be translated into any other language.
9. Syntax errors.
11. Instructions in a high-level language are closer to the natural language, such as English, and therefore, are easier to understand and learn than the machine language.
13. To find the weighted average of four test scores, first you need to know each test score and its weight. Next, you multiply each test score by its weight and then add these numbers to get the average. Therefore:
    1. Get `testScore1, weightTestScore1`
    2. Get `testScore2, weightTestScore2`
    3. Get `testScore3, weightTestScore3`
    4. Get `testScore4, weightTestScore4`
    5.
```
sum = testScore1 * weightTestScore1 +
      testScore2 * weightTestScore2 +
      testScore3 * weightTestScore3 +
      testScore4 * weightTestScore4;
```
15. To calculate the selling price of an item, we need to know the original price (the price the store pays to buy it) of the item. We can then the use the following formula to find the selling price:

```
sellingPrice = (originalPrice + originalPrice × 0.80) × 0.90
```

The algorithm is as follows:

a. Get `originalPrice`

b. Calculate the `sellingPrice` using the formula:

```
sellingPrice = (originalPrice + originalPrice × 0.80) × 0.90
```

The information needed to calculate the selling price is the original price and the marked-up percentage.

17. Suppose that `numOfPages` denotes the number of pages to be faxed and `billingAmount` denotes the total charges for the pages faxed. To calculate the total charges, you need to know the number of pages faxed.

If `numOfPages` is less than or equal to 10, the billing amount is service charges plus (`numOfPages × 0.20`); otherwise, billing amount is service charges plus `10 × 0.20` plus (`numOfPages - 10`) × `0.10`.

You can now write the algorithm as follows:

a. Get `numOfPages`.

b. Calculate billing amount using the formula:

```
if (numOfPages is less than or equal to 10)
    billingAmount = 3.00 + (numOfPages × 0.20);
otherwise
    billingAmount = 3.00 + 10 × 0.20 + (numOfPages - 10) × 0.10;
```

This page contains answers for this chapter only