CHAPTER 1

Introduction to Programming and the Java Language

CHAPTER CONTENTS

Introduction

Computer applications touch almost every aspect of our lives. They run automated teller machines, the grocery store's checkout register, the appointment calendar at your doctor's office, airport kiosks for flight check-in, a restaurant's meal-ordering system, and online auctions, just to name a few applications. On your personal computer, you may run a word processor, virus detection software, a spreadsheet, computer games, and an image processing system.

Someone, usually a team of programmers, wrote those applications. If you're reading this text, you're probably curious about what's involved in writing applications, and you would like to write a few yourself. Perhaps you have an idea for the world's next great application or computer game.

In this text, we'll cover the basics of writing applications. Specifically, we'll use the Java programming language. Keep in mind, however, that becoming a good programmer requires more than mastering the rules, or **syntax**, of a programming language. You also must master basic programming techniques. These are established methods for performing common programming operations, such as calculating a total, finding an average, or arranging a group of items in order.

You also must master good software engineering principles so that you design code that is readable, easily maintained, and reusable. By readable, we mean that someone else should be able to read your program and figure out what it does and how it does it. Writing readable code is especially important for programmers who want to advance in their careers, because it allows someone else to take over the maintenance of your program while you move on to bigger and better responsibilities. Ease of maintenance is also an important aspect of programming, because the specifications for any program are continually changing. How many programs can you name that have had only one version? Not many. Well-designed code allows you and others to incorporate prewritten and pretested modules into your program, thus reducing the time to develop a program and yielding code that is more robust and has fewer bugs. One useful feature of the Java programming language is the large supply of prewritten code that you are free to use in your programs.

Programming is an exciting activity. It's very satisfying to decompose a complex task into computer instructions and watch your program come alive. It can be frustrating, however, when your program either doesn't run at all or produces the wrong output.

Writing correct programs is critical. Someone's life or life savings may depend on the correctness of your program. Reusing code helps in developing correct programs,

but you must also master effective testing techniques to verify that the output of your program is correct.

In this text, we'll concentrate not only on the syntax of the Java language, but also on basic programming techniques, good software engineering principles, and effective testing techniques.

Before you can write programs, however, it's important to understand the platform on which your program will run. A platform refers to the computer hardware and the operating system. Your program will use the hardware for inputting data, for performing calculations, and for outputting results. The operating system will start your program running and will provide your program with essential resources, such as memory, and services, such as reading and writing files.

1.1   Basic Computer Concepts

1.1.1   Hardware

As shown in Figure 1.1, a computer typically includes the following components:

- a CPU, or central processing unit, which executes the instructions of a program
- a memory unit, which holds the instructions and data of a program while it is executing
- a hard disk, used to store programs and data so that they can be loaded into memory and accessed by the CPU
- a keyboard used for input of data
- a monitor, used to display output from a program
- an Ethernet port and wireless networking transceiver for connecting to the Internet or a Local Area Network (LAN)
- other components (not shown) such as a graphics card and a DVD drive

For example, if you were to go to a computer store in search of the latest personal computer, you might be shown a computer with this set of specifications:

- a 3.4-GHz Intel Core i3-3240 processor
- a touch screen
- 3 MB of cache memory
- 8 GB of RAM (Random Access Memory)
- a 1 TB (Terabyte) hard disk

Motherboard
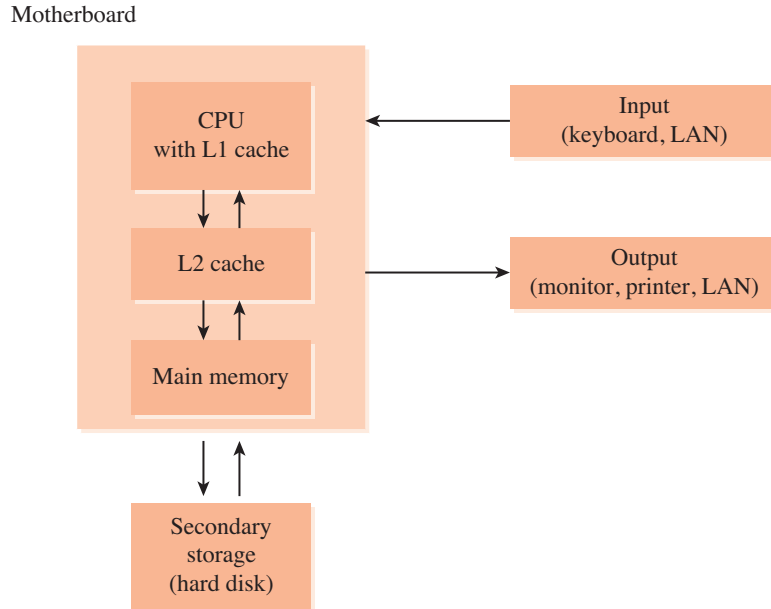
In these specifications, the Intel Core i3-3240 is the CPU. Other processors used as CPUs in personal computers and servers include the AMD Athlon, the Oracle SPARC, and the IBM POWER processor.

CPUs consist of an Arithmetic Logic Unit (ALU) [also called an Integer Unit (IU)], which performs basic integer arithmetic and logical operations; a Floating Point Unit (FPU), which performs floating-point arithmetic; a set of hardware registers for holding data and memory addresses; and other supporting hardware, including a control unit to sequence the instructions. Each CPU comes with its own set of instructions, which are the operations that it can perform. The instructions typically perform arithmetic and logic operations, move data from one location to another, and change the flow of the program (that is, determine which instruction is to be executed next).

A program consists of many instructions. The first step in executing a program is loading it into memory. Each instruction in the program is executed using a **Fetch-Decode-Execute Cycle**. The **Program Counter** keeps track of the next instruction to be fetched. The CPU fetches the next instruction from memory and places it into the **Instruction Register**. The instruction is decoded (is the instruction a move, load, store, etc.?). Then, the instruction is executed. This Fetch-Decode-Execute Cycle repeats until the program ends.

The speed of a CPU is related to its clock cycle, typically rated in GHz (gigahertz); at the time of this edition, a high-end CPU speed would be rated at 3.4 GHz. The clock speed, measured in clock cycles per second, determines how fast a processor can perform operations such as fetching, decoding, and executing instructions. A CPU rated

at 2.5 GHz executes 2.5 billion clock cycles a second. Current RISC processors feature pipelining, which allows the CPU to process several instructions at once, so that while one instruction is executing, the processor can decode the next instruction and fetch the next instruction after that. This greatly improves performance of applications.

Memory or storage devices, such as L2 cache, memory, or hard disks, are typically rated in terms of their capacity, expressed in bytes. A **byte** is eight binary digits, or **bits**. A single bit's value is 0 or 1. Depending on the type of memory or storage device, the capacity will be stated in kilobytes, megabytes, gigabytes, or even tera-bytes. The sizes of these units are shown in Table 1.1.

For the CPU to execute at its rated speed, however, instructions and data must be available to the CPU at that speed as well. Instructions and data come directly from the L1 cache, which is memory directly located on the CPU chip. Since the L1 cache is located on the CPU chip, it runs at the same speed as the CPU. However, the L1 cache, which can be several Mbytes, is typically much smaller than main memory, and eventually the CPU will need to process more instructions and data than can be held in the L1 cache at one time.

At that point, the CPU typically brings data from what is called the L2 cache, which is located on separate memory chips connected to the CPU. A typical speed for the L2 cache would be a few nanoseconds (billionths of a second) access time, and this will considerably slow down the rate at which the CPU can execute instructions. L2 cache size today is typically 3 to 8 Mbytes, and again, the CPU will eventually need more space for instructions and data than the L2 cache can hold at one time.

At that point, the CPU will bring data and instructions from main memory, also located outside, but connected to, the CPU chip. This will slow down the CPU even more, because main memory typically has an access time of about 20 to 50 nanosec-onds. Main memory, though, is significantly larger in size than the L1 and L2 caches, typically anywhere between 3 and 8 Gbytes. When the CPU runs out of space again, it will have to get its data from the hard disk, which is typically 1 Tbyte or more, but with an access time in the milliseconds (thousandths of a second) range.

TABLE 1.1 Memory Units and Their Sizes

Memory Unit	Size
KB, or Kbytes, or kilobytes	About 1,000 bytes (exactly 2^{10} or 1,024 bytes)
MB, or Mbytes, or megabytes	About 1 million bytes (exactly 2^{20} or 1,048,576 bytes)
GB, or Gbytes, or gigabytes	About 1 billion bytes (exactly 2^{30} or 1,073,741,824 bytes)
TB, or Tbytes, or terabytes	About 1 trillion bytes (exactly 2^{40} or 1.09951×10^{12} bytes)

As you can see from these numbers, a considerable amount of speed is lost when the CPU goes from main memory to disk, which is why having sufficient memory is very important for the overall performance of applications.

Another factor that should be taken into consideration is cost per kilobyte. Typically the cost per kilobyte decreases significantly stepping down from L1 cache to hard disk, so high performance is often traded for low price.

Main memory (also called RAM) uses DRAM, or Dynamic Random Access Memory technology, which maintains data only when power is applied to the memory and needs to be refreshed regularly in order to retain data. The L1 and L2 caches use SRAM, or Static Random Access Memory technology, which also needs power but does not need to be refreshed in order to retain data. Memory capacities are typically stated in powers of 2. For instance, 256 Kbytes of memory is 2^{18} bytes, or 262,144 bytes.

Memory chips contain cells, each cell containing a bit, which can store either a 0 or a 1. Cells can be accessed individually or as a group of typically 4, 8, or 16 cells. For instance, a 32-Kbit RAM chip organized as $8K \times 4$ is composed of exactly 2^{13}, or 8,192 units, each unit containing four cells. This RAM chip will have four data output pins (or lines) and 13 access pins (or lines), enabling access to all 8,192 units because each access pin can have a value of 0 or 1. Table 1.2 compares the features of various memory types.

1.1.2   Operating Systems

An operating system (OS) is a software program that

- controls the peripheral devices (for instance, it manages the file system)
- supports multitasking, by scheduling multiple programs or tasks to execute during the same interval
- allocates memory to each program, so that there is no conflict among the memory of any programs running at the same time
- prevents the user from damaging the system. For instance, it prevents user programs from overwriting the operating system or another program's memory

The operating system loads, or **boots**, when the computer system is turned on and is intended to run as long as the computer is running.

Examples of operating systems are macOS, Microsoft Windows, Linux, and iOS and Android for mobile devices.

TABLE 1.2   A Comparison of Memory Types

Device	Location	Type	Speed	Capacity (MB)	Cost/KB
L1 cache	On-chip	SRAM	Very fast	Very small	Very high
L2 cache	Off-chip	SRAM	Fast	Small	High
Memory	Off-chip	DRAM	Moderate	Moderate	Moderate
Hard disk	Separate	Disk media	Slow	Large	Small

1.1.3   Application Software

Application software consists of the programs written to perform specific tasks. These programs are run by the operating system, or as is typically said, they are run "on top of" the operating system. Examples of applications are word processors, such as Microsoft Word or Apache OpenOffice; spreadsheets, such as Microsoft Excel; database management systems, such as Oracle or Microsoft SQL Server; Internet browsers, such as Mozilla Firefox, Google Chrome, or Microsoft Edge; and most of the programs you will write during your study of Computer Science.

1.1.4   Computer Networks and the Internet

Computer Networks

Computer networks connect two or more computers. A common network used by many corporations and universities is a LAN, or Local Area Network. A typical LAN connects several computers that are geographically close to one another, often in the same building, and allows them to share resources, such as a printer, a database, or a file system. In a LAN, most user computers are called **clients**, and one or more computers act as **servers**. The server controls access to resources on the network and can supply services to the clients, such as answering database requests, storing and serving files, or managing email. The computers on the network can exchange data through physical cables or through a wireless network.

The Internet

The Internet is a network of networks, connecting millions of computers around the world. The Internet evolved from ARPANET, a 1969 U.S. military research project whose goal was to design a method for computers to communicate. Most computers on the Internet are clients, typically requesting resources, such as webpages, through an Internet browser. These resources are provided by web servers, which store webpages and respond to these requests.

Every machine on the Internet has a unique ID called its IP address (IP stands for Internet Protocol). Special computers called **routers** find a path through the Internet from one computer to another using these IP addresses. A computer can have a static IP address, which is dedicated to that machine, or a dynamic IP address, which is assigned to the computer when it connects to the Internet. An IP address is made up of four octets, whose values in decimal notation are between 0 and 255. For instance, 58.203.151.103 could represent such an IP address. In binary notation, this IP address is 111010.11001011.10010111.1100111. Later in this chapter, we will learn how to convert a decimal number, such as 103, to its binary equivalent, 1100111.

Most people are familiar with URL (Uniform Resource Locator) addresses that look like *http://www.oracle.com/technetwork/java/index.html*. A URL is actually an Internet domain name as well as the path on that domain to a specific web page. In this URL, the domain name is *www.oracle.com*. The page requested is *index.html*, which is located in the *java* folder under the *technetwork* folder.

Domain name resolution servers, which implement the Domain Name System (DNS), convert domain names to IP addresses, so that Internet users don't need to know the IP addresses of websites they want to visit. The World Wide Web Consortium (W3C), an international group developing standards for Internet access, prefers the term Uniform Resource Identifier (URI) rather than URL, because URI covers future Internet addressing schemes.

Skill Practice
with these end-of-chapter questions

1.7.1   Multiple Choice Exercises

Questions 1, 2, 3, 4

1.7.3   General Questions

Questions 21, 22, 23

1.7.4   Technical Writing

Questions 31, 32, 33

1.2   Practice Activity: Displaying System Configuration

We have explored hardware and operating systems in general. Now, let's discover some information about the hardware and operating system on your computer. Depending on whether you're using a Windows operating system or a Mac OS

operating system, choose the appropriate directions that follow to display the operating system's name, the CPU type, and how much memory the computer has.

1.2.1   Displaying Windows Configuration Information

To display system configuration information on a computer running Windows 10, run *msinfo32.exe* from the command line. From the *Start* menu, type *cmd* to start the Command Prompt program and then type *msinfo32* into the Command Prompt window. You will get a display similar to the one in Figure 1.2, although the information displayed varies, depending on your hardware and the version of Windows you are running.

Item	Value
OS Name	Microsoft Windows 10 Enterprise
Version	10.0.14393 Build 14393
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	PC07WK39
System Manufacturer	LENOVO
System Model	20FAS3QE00
System Type	x64-based PC
System SKU	LENOVO_MT_20FA_BU_Think_FM_ThinkPad T460s
Processor	Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz, 2808 Mhz, 2 Core(s), 4 Logical Processor(s)
BIOS Version/Date	LENOVO N1CET47W (1.15), 8/8/2016
SMBIOS Version	2.8
Embedded Controller Version	1.09
BIOS Mode	Legacy
BaseBoard Manufacturer	LENOVO
BaseBoard Model	Not Available
BaseBoard Name	Base Board
Platform Role	Mobile
Secure Boot State	Unsupported
PCR7 Configuration	Binding Not Possible
Windows Directory	C:\windows
System Directory	C:\windows\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.14393.206"
User Name	UNIVERSITY\USER
Time Zone	Eastern Daylight Time
Installed Physical Memory (RAM)	12.0 GB
Total Physical Memory	11.4 GB
Available Physical Memory	8.41 GB
Total Virtual Memory	13.8 GB
Available Virtual Memory	10.2 GB
Page File Space	2.38 GB
Page File	C:\pagefile.sys
Device Guard Virtualization based security	Not enabled
Hyper-V - VM Monitor Mode Extensions	Yes
Hyper-V - Second Level Address Translation Extensions	Yes
Hyper-V - Virtualization Enabled in Firmware	Yes
Hyper-V - Data Execution Protection	Yes

Figure 1.2

Windows System Information

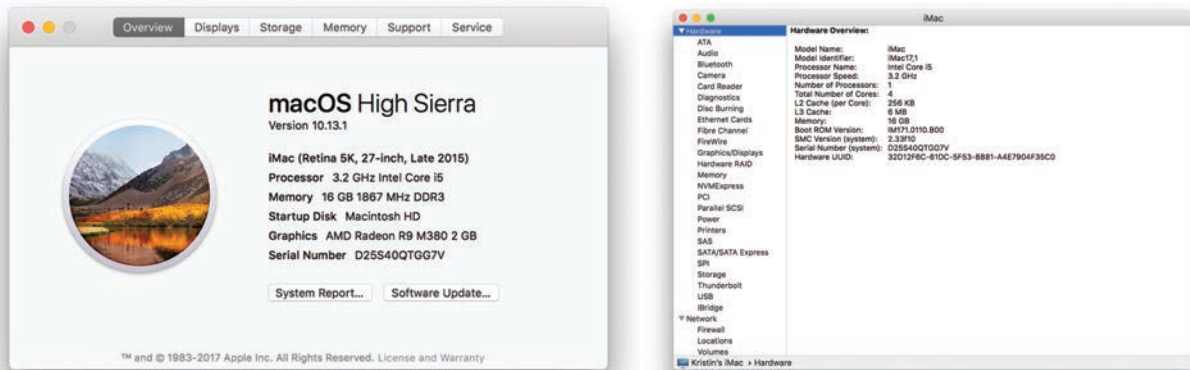Screen shots reprinted with permission from Apple Inc.

Figure 1.3

Mac Operating System Information

Screen shots reprinted with permission from Apple Inc.

As you can see in Figure 1.2, this computer is running Intel™ Core™ i7-6600U CPU processor running at 2.6 GHz, and the computer has 12 Gbytes of memory, 8.41 Gbytes of which is not being used at the time of the display.

1.2.2   Displaying Mac OS Configuration Information

To display system information on a Mac OS computer, click on the apple icon on the top left of the screen and select "About This Mac." You will see something similar to the display in Figure 1.3, which displays the version of the MAC OS operating system that is running on the computer.

As you can see, this iMac is running macOS High Sierra version 10.13.1. The processor is an Intel Core i5 running at 3.2 GHz and the iMac has 16 Gbytes of memory. Your information may be different from that shown in the figure, depending on your hardware and operating system version.

DISCUSSION QUESTIONS   **?**

1. Compare the system information on several computers. Is it the same or different from computer to computer? Explain why the information is the same or different.

2. In the sample display for Windows 10, the computer has 12 Gbytes of memory, but only 8.41 Gbytes of memory is available. Why do you think some memory is not available?

3. Compare your computer to the ones on the previous pages shown here. Which do you think would have better performance? Explain your answer.

1.3   Data Representation

1.3.1   Binary Numbers

As mentioned earlier, a CPU understands only binary numbers, whose digits consist of either 0 or 1. All data is stored in a computer's memory as binary digits. A bit holds one binary digit. A byte holds eight binary digits.

Binary numbers are expressed in the base 2 system, because there are only 2 values in that system, 0 and 1. By contrast, most people are used to the decimal, or base 10, system, which uses the values 0 through 9.

There are other number systems, such as the octal, or base 8, system, which uses the digits from 0 to 7, and the hexadecimal, or base 16, system, which uses the digits 0 to 9 and the letters A to F.

As we know it in the decimal system, the number 359 is composed of the following three digits:

3, representing the hundreds, or 10^2

5, representing the tens, or 10^1

9, representing the ones, or 10^0

Therefore, we can write 359 as

$$359 = 3*10^2 + 5*10^1 + 9*10^0$$

Thus, the decimal number 359 is written as a linear combination of powers of 10 with coefficients from the base 10 alphabet, that is, the digits from 0 to 9. Similarly, the binary number 11011 is written as a linear combination of powers of 2 with coefficients from the base 2 alphabet, that is, the digits 0 and 1.

For example, the binary number 11011 can be written as

$$11011 = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$

Table 1.3 lists the binary equivalents for the decimal numbers 0 through 8, while Table 1.4 lists the decimal equivalents of the first 15 powers of 2.

Note that in Table 1.3, as we count in increments of 1, the last digit alternates between 0 and 1. In fact, we can see that for even numbers, the last digit is always 0 and for odd numbers, the last digit is always 1.

Because computers store numbers as binary, and people recognize numbers as decimal values, conversion between the decimal and binary number systems often takes place inside a computer.

TABLE 1.3   Binary Equivalents of Decimal Numbers 0 Through 8

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000

TABLE 1.4   Powers of 2 and Their Decimal Equivalents

2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1

Let's try a few conversions. To convert a binary number to a decimal number, multiply each digit in the binary number by $2^{position-1}$, counting the rightmost position as position 1 and moving left through the binary number. Then add the products together.

Using this method, let's calculate the equivalent of the binary number 11010 in our decimal system.

```
11010 = 1*2⁴ + 1*2³ + 0*2² + 1*2¹ + 0*2⁰
      = 16 + 8 + 0 + 2 + 0
      = 26
```

Now let's examine how to convert a decimal number to a binary number. Let's convert the decimal number 359 into its binary number equivalent. As we can see from the way we rewrote 11011, a binary number can be written as a sum of powers of 2 with coefficients 0 and 1.

The strategy to decompose a decimal number into a sum of powers of 2 is simple: first find the largest power of 2 that is smaller than or equal to the decimal number,

subtract that number from the decimal number, then do the same with the remainder, and so on, until you reach 0.

The largest power of 2 that is smaller than 359 is 256, or 2^8 (the next larger power of 2 would be 512, which is larger than 359). Subtracting 256 from 359 gives us 103 ($359 - 256 = 103$), so we now have

$$359 = 2^8*1 + 103$$

Now we apply the same procedure to 103. The largest power of 2 that is smaller than 103 is 64, or 2^6. That means that there is no factor for 2^7, so that digit's value is 0. Subtracting 64 from 103 gives us 39.

Now we have

$$359 = 2^8*1 + 2^7*0 + 2^6*1 + 39$$

Repeating the procedure for 39, we find that the largest power of 2 smaller than 39 is 32 or 2^5. Subtracting 32 from 39 gives us 7.

So we now have

$$359 = 2^8*1 + 2^7*0 + 2^6*1 + 2^5*1 + 7$$

Repeating the procedure for 7, the largest power of 2 smaller than 7 is 2^2, or 4. That means that there are no factors for 2^4 or 2^3, so the value for each of those digits is 0. Subtracting 4 from 7 gives us 3, so we have

$$359 = 2^8*1 + 2^7*0 + 2^6*1 + 2^5*1 + 2^4*0 + 2^3*0 + 2^2*1 + 3$$

Repeating the procedure for 3, the largest power of 2 smaller than 3 is 2, or 2^1, and we have:

$$359 = 2^8*1 + 2^7*0 + 2^6*1 + 2^5*1 + 2^4*0 + 2^3*0 + 2^2*1 + 2^1*1 + 1$$

1 is a power of 2; it is 2^0, so we finally have

$$359 = 2^8*1 + 2^7*0 + 2^6*1 + 2^5*1 + 2^4*0 + 2^3*0 + 2^2*1 + 2^1*1 + 2^0*1$$

Removing the power of 2 multipliers, 359 can be represented in the binary system as

$$359 = 2^8*1 + 2^7*0 + 2^6*1 + 2^5*1 + 2^4*0 + 2^3*0 + 2^2*1 + 2^1*1 + 2^0*1$$
$$= 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1$$

or

`1 0110 0111`

CODE IN ACTION

To see a step-by-step demonstration of converting between decimal and binary numbers, look for the movie within the online resources. Click on the link to start the movie.

In a computer program, we will use both positive and negative numbers. Appendix D explains how negative numbers, such as −34, are represented in the binary system. In a computer program, we also use floating-point numbers, such as 3.75. Appendix E explains how floating-point numbers are represented using the binary system.

1.3.2   Using Hexadecimal Numbers to Represent Binary Numbers

As you can see, binary numbers can become rather long. With only two possible values, 0 and 1, it takes 16 binary digits to represent the decimal value +32,768. For that reason, the hexadecimal, or base 16, system is often used as a shorthand representation of binary numbers. The hexadecimal system uses 16 digits: 0 to 9 and A to F. The letters A to F represent the values 10, 11, 12, 13, 14, and 15.

The maximum value that can be represented in four binary digits is $2^4 - 1$, or 15. The maximum value of a hexadecimal digit is also 15, which is represented by the letter F. So you can reduce the size of a binary number by using hexadecimal digits to represent each group of four binary digits.

Table 1.5 displays the hexadecimal digits along with their binary equivalents.

To represent the following binary number in hexadecimal, you simply substitute the appropriate hex digit for each set of four binary digits.

```
0001 1010 1111 1001 1011 0011 1011 1110
  1    A    F    9    B    3    B    E
```

Here's an interesting sequence of hexadecimal numbers. The first 32 bits of every Java applet are:

```
1100 1010 1111 1110 1011 1010 1011 1110
```

Translated into hexadecimal, that binary number becomes:

```
CAFE BABE
```

TABLE 1.5   Hexadecimal Digits and Equivalent Binary Values

Hex Digit	Binary Value	Hex Digit	Binary Value
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

1.3.3   Representing Characters with the Unicode Character Set

Java represents characters using the Unicode Worldwide Character Standard, or simply Unicode. Each Unicode character is represented as 16 bits, or two bytes. This means that the Unicode character set can encode 65,536 characters.

The Unicode character set was developed by the Unicode Consortium, which consists of computer manufacturers, software vendors, the governments of several nations, and others. The consortium's goal was to support an international character set, including the printable characters on the standard QWERTY keyboard, as well as international characters such as é or λ.

Many programming languages store characters using the ASCII (American Standard Code for Information Interchange) character set, which uses 7 bits to encode each character, and thus, can represent only 128 characters. For compatibility with the ASCII character set, the first 128 characters in the Unicode character set are the same as the ASCII character set.

Table 1.6 shows a few examples of Unicode characters and their decimal equivalents.

For more information on the Unicode character set, see Appendix C or visit the Unicode Consortium's website at *http://www.Unicode.org*.

TABLE 1.6   Selected Unicode Characters and Their Decimal Equivalents

Unicode Character	Decimal Value	Hex Value
NUL, the null character (a nonprintable character)	0	0000
*	42	002A
1	49	0031
2	50	0032
A	65	0041
B	66	0042
a	97	0061
b	98	0062
}	125	007D
delete (a nonprintable character)	127	007F

Skill Practice
with these end-of-chapter questions

1.7.1   Multiple Choice Exercises

Questions 5, 6, 7, 8

1.7.2   Converting Numbers

Questions 15, 16, 17, 18, 19, 20

1.7.3   General Questions

Questions 24, 25, 26

1.4   Programming Languages

1.4.1   High- and Low-Level Languages

Programming languages can be categorized into three types:

- machine language
- assembly language
- high-level language

In the early days of computing, programmers often used machine language or assembly language. Machine language uses binary codes, or strings of 0s and 1s, to execute the instruction set of the CPU and to refer to memory addresses. This method of programming is extremely challenging and time consuming. Also, the code written in machine language is not portable to other computer architectures because each CPU has its own set of instructions. Machine language's early popularity can be attributed largely to the fact that programmers had no other choices. However, programmers rarely use machine language today.

Assembly languages are one step above machine language, using symbolic names for memory addresses and mnemonics for processor instructions—for example: *BEQ* (branch if equal), *SW* (store), or *LW* (load). An Assembler program converts the code to machine language before it is executed. Like machine language, assembly languages are also CPU-dependent and are not portable among computers with different processors (for instance, between Intel and SPARC). Assembly language is easier to write than machine language but still requires a significant effort and, thus, is usually used only when the program requires features, such as direct hardware access, that are not supported by a high-level language.

High-level languages, such as Java, C++, Swift, PHP, and Python, are closer to the English language than they are to machine language, making them a lot easier to use for software development and more portable among CPU architectures. For this reason, programmers have embraced high-level languages for more and more applications.

Characteristics of high-level languages, such as Java, are

- The languages are highly symbolic. Programmers write instructions using keywords and special characters and use symbolic names for data.

- The languages are somewhat portable (some more portable than others) among different CPUs.

- Some high-level programming languages, such as C++, Java, and Python, can be used for general-purpose applications, and some high-level programming languages can be used for specialized applications, such as Swift for iPhone development and PHP for Internet applications.

High-level languages are compiled, interpreted, or a combination of both. A program written in a compiled language, such as C++, is converted by a compiler into machine code, then the machine code is executed.

By contrast, a program written using an interpreted language, such as Perl, is read and converted to machine code, line by line, at execution time. Typically, a program written in an interpreted language will run more slowly than its equivalent written in a compiled language.

Java uses a combination of a compiler and an interpreter. A Java program is first compiled into processor-independent **byte codes**, then the byte code file is interpreted at run time by software called the **Java Virtual Machine (JVM)**.

1.4.2   An Introduction to Object-Oriented Programming

Initial high-level languages, such as Fortran or C, were procedural. Typically, programmers wrote task-specific code in separate procedures, or functions, and invoked these procedures from other sections of the program in order to perform various tasks. The program's data was generally shared among the procedures.

In the mid-1970s, the first object-oriented programming language, Smalltalk, was introduced, enabling programmers to write code with a different approach. Whereas procedures or functions dealt mainly with basic data types such as integers, real numbers, or single characters, Smalltalk provided the programmer with a new tool: classes and objects of those classes.

A class enables the programmer to encapsulate data and the functions to manipulate that data into one package. A class essentially defines a template, or model, from which objects are created. Creating an object is called **instantiation**. Thus, objects are created—instantiated—according to the design of the class.

A class could represent something in real life, such as a person. The class could have various attributes such as, in the example of a "person" class, a first name, a last name, and an age. The class would also provide code, called **methods**, that allow the creator of the object to set and retrieve the values of the attributes.

One big advantage to object-oriented programming is that well-written classes can be reused in new programs, thereby reducing future development time.

Smalltalk was somewhat successful, but had a major deficiency: its syntax was unlike any syntax already known by most programmers. Most programmers who knew the procedural C programming language were attracted by the object-oriented features of Smalltalk but were reluctant to use it because its syntax was so different from C's syntax. C++ added object-oriented features to C, but also added complexity.

Meanwhile, the Internet was growing by leaps and bounds and gaining popularity daily. Web developers used HTML to develop webpages and soon felt the need to incorporate programming features not only on the server side, but also directly on the client side. Fortunately, Java appeared on the scene.

1.4.3   The Java Language

On May 23, 1995, Sun Microsystems introduced Java, originally named Oak, as a free, object-oriented language targeted at embedded applications for consumer

devices. A Java Virtual Machine was incorporated immediately into the Netscape Navigator Internet browser, and as the Internet grew, small Java programs, known as applets, began to appear on webpages in increasing numbers. Java syntax is basically identical (with some minor exceptions) to that of C++, and soon programmers all over the world started to realize the benefits of using Java. Those benefits include

- syntax identical to that of C++, except that Java eliminates some of C++'s more complex features
- object orientation
- Internet-related features, such as servlets, which are run by the web server
- an extensive library of classes that can be reused readily, including classes for providing a Graphical User Interface and Java Database Connectivity (JDBC) for communicating with a database
- portability among every platform that supports a Java Virtual Machine
- built-in networking
- open source availability of the Java Development Kit

As we mentioned earlier, a Java program is first compiled into processor-independent byte codes, then the byte codes are interpreted at run time by the Java Virtual Machine (JVM). As its name implies, the JVM simulates a virtual processor with its own instruction set, registers, and instruction pointer. Thus, to run a Java program, we only need a JVM. Fortunately, JVMs are available on every major computing platform.

Because Java programs are interpreted at run time, they typically run more slowly than their C++ counterparts. However, many platforms provide Java compilers that convert source code directly to machine code. This results in greater execution speed, but with an accompanying loss of portability. Just-in-Time (JIT) compilers are also available. These JITs compile code at run time so that subsequent execution of the same code runs much faster.

Java programs can be written as servlets, Android apps, or applications.

Java servlets are invoked by the web server and run on the server, without being downloaded to the client. Typically, servlets dynamically generate web content by reading and writing to a database using JDBC (Java Database Connectivity).

The Android Open Source project has chosen Java to be the language for developing apps for Android devices, such as cell phones, tablets, and smart TVs.

Java applications run standalone on a client computer. In this text, we will write Java applications.

Oracle Corporation, which acquired Sun Microsystems in January 2010, provides a valuable Java website (*www.oracle.com/technetwork/java*), which has information on using the prewritten classes, a tutorial on Java, and many more resources for the Java programmer. We will refer you to that site often in this text.

1.5   An Introduction to Programming

1.5.1   Programming Basics

In many ways, programming is like solving a puzzle. We have a task to perform and we know the operations that a computer can perform (input, calculations, comparisons, rearranging of items, and output). As programmers, our job is to decompose a task into individual, ordered steps of inputting, calculating, comparing, rearranging, and outputting.

For example, suppose our task is to find the sum of two numbers. First, our program needs to read (input) the numbers into the computer. Next, our program needs to add the two numbers together (calculate). Finally, our program needs to write (output) the sum.

Notice that this program consists of steps, called **instructions**, which are performed in order ("First," "Next," "Finally"). Performing operations in order, one after another, is called **sequential processing**.

The order in which instructions are executed by the computer is critical in programming. You can't calculate the sum of two numbers before you have read the two numbers, and you can't output a sum before you have calculated it. Programming, therefore, requires the programmer to specify the ordering of instructions, which is called the **flow of control** of the program. There are four different ways that the flow of control can progress through a program: sequential execution, method call, selection, and looping. We've just seen sequential execution, and we'll discuss the other types of flow of control in the next section.

Because getting the flow of control correct is essential to getting a program to produce correct output, programmers use a tool called **pseudocode** (pronounced *sue dough code*) to help them design the flow of control before writing the code.

1.5.2   Program Design with Pseudocode

Pseudocode, from *pseudo*, which means "appearing like," is a method for expressing a program's order of instructions in the English language, rather than a

programming language. In this way, the programmer can concentrate on designing a program without also being bogged down in the syntax of the particular programming language.

The pseudocode for calculating the sum of two numbers would look like Example 1.1.

```
read first number
read second number
set total to (first number + second number)
output total
```

EXAMPLE 1.1   Pseudocode for Summing Two Numbers

Fortunately, the rules for writing pseudocode are not rigid. Essentially, you can use any wording that works for you.

Let's look at another example. Suppose our program needs to calculate the square root of a number. The instructions for calculating a square root are rather complex; fortunately, Java provides prewritten code that computes the square root of a number. The prewritten code is called a **method**, and our program can execute that code by **calling the method**. As part of the method call, we tell the method which number's square root we want to calculate. This is called **passing an argument to the method**. When the method finishes executing its instructions, control is passed back to our program just after the method call and our program can use the square root of the number calculated by the method. This is called **returning a value**. Another way of looking at method calls is to consider what happens when we're reading a book and find a word we don't understand. We mark our place in the book and look up the word in a dictionary. When we're finished looking up the word, we go back to our place in the book and continue reading.

Example 1.2 shows the pseudocode for calculating the square root of an integer.

```
read a number
call the square root method, passing the number and receiving the square root
output the square root of the number
```

EXAMPLE 1.2   Using a Method Call to Calculate a Square Root

The order of operations is still input, calculate, and output, but we're calling a method to perform the calculation for us.

Now suppose our task is to determine whether a number is positive or negative. First, our program should input the number into the computer. Next, we need to determine whether the number is positive or negative. We know that numbers greater than or equal to 0 are positive and numbers less than 0 are negative, so our program should compare the number to 0. Finally, our program should write a message indicating whether the number is positive or negative.

Like Examples 1.1 and 1.2, the operations are input, calculate, and output, in that order. However, depending on whether the number is positive or negative, our program should write a different message. If the number is greater than or equal to 0, the program should write a message that the number is positive, but if the number is less than 0, the program should write a message that the number is negative. Code used to handle this situation is called **selection**; the program selects which code to execute based on the value of the data.

The pseudocode for this program could be written as that shown in Example 1.3.

```
read a number
if the number is greater than or equal to 0
  write "Number is positive."
else
  write "Number is negative."
```

EXAMPLE 1.3   Using Selection

Notice the indentation for the code that will be selected based on the comparison of the number with 0. Programmers use indentation to make it easier to see the flow of control of the program.

Now let's get a little more complicated. Suppose our program needs to find the sum of a group of numbers. This is called **accumulating**. To accomplish this, we can take the same approach as if we were adding a group of numbers using a calculator. We start with a total of 0 and add each number, one at a time, to the running total. When we have no more numbers to add, the running total is the total of all the numbers.

Translating this into pseudocode, we get the code shown in Example 1.4.

```
set total to 0
read a number
while there was a number to read, repeat next two instructions
  add number to total
  read the next number
write total
```

EXAMPLE 1.4   Accumulating a Total

The indented code will be repeated for each number read until there are no more numbers. This repeated execution of the same code is called **looping**, or **iteration**, and is used extensively in programming whenever the same processing needs to be performed on each item in a set.

Accumulating a total and determining whether a number is positive or negative are just two of many commonly performed operations. In programming,

we will often perform tasks for which there are standard methods of process-ing, called **algorithms**. For example, the algorithm for accumulation is to set a total to 0, use looping to add each item to the total, then output the total. More generally, we can think of an algorithm as a strategy to solve a problem. Earlier in the chapter, we used an algorithm to convert a decimal number to its binary representation.

Other common programming tasks are counting items, calculating an average, sort-ing items into order, and finding the minimum and maximum values. In this text, we will learn the standard algorithms for performing these common operations. Once we learn these algorithms, our programming job will become easier. When we recognize that a program requires any of these tasks, we can simply plug in the appropriate algorithm with some minor modifications.

Programming, in large part, is simply reducing a complex task to a set of subtasks that can be implemented by combining standard algorithms that use sequential pro-cessing, method calls, selection, and looping.

The most difficult part of programming, however, is recognizing which algorithms to apply to the problem at hand. This requires analytical skills and the ability to see pat-terns. Throughout this text, we will point out common patterns wherever possible.

1.5.3   Developing a Java Application

Writing a Java application consists of several steps: writing the code, compil-ing the code, and executing the application. Java source code is stored in a text file with the extension *.java*. Compiling the code creates one or more *.class* files, which contain processor-independent byte codes. The Java Virtual Machine (JVM) translates the byte codes into machine-level instructions for the processor on which the Java application is running. Thus, if a Java application is running on an Intel i7 processor, the JVM translates the byte codes into the i7's instruc-tion set.

Oracle provides a Java SE Development Toolkit (JDK) on its website (*www.oracle .com/technetwork/java*), which is downloadable free of charge. The JDK contains a compiler, JVM, and jshell, which is a tool for immediate execution of Java code. In addition, the JDK contains a broad range of prewritten Java classes that program-mers can use in their Java applications.

If you are downloading and installing Java yourself, be sure to follow the directions on the Oracle website, including the directions for setting the path for *javac*, the Java compiler. You need to set the path correctly so that you can run the Java compiler from any folder on your computer.

SOFTWARE ENGINEERING TIP
Looking for patterns will help you determine the appropriate algorithms for your programs.

To develop an application using the JDK, write the source code using any text editor, such as Notepad on Windows, TextEdit on Macs, or the vi editor on Linux. To compile the code, invoke the compiler from the command line:

```
javac ClassName.java
```

where *ClassName.java* is the name of the source file.

If our program, written in the file *ClassName.java*, compiles correctly, a new file, *ClassName.class*, will be created in our current directory.

To run the application, we invoke the JVM from the command line:

```
java ClassName
```

Typically, programmers use an Integrated Development Environment (IDE) to develop applications rather than invoking the compiler and JVM from the command line. An IDE consists of a program editor, a compiler, and a run-time environment, integrated via a Graphical User Interface. The advantage to using an IDE is that errors in the Java code that are found by the compiler or the JVM can be linked directly to the program editor at the line in the source file that caused the error. Additionally, the Graphical User Interface enables the programmer to switch among the editor, compiler, and execution of the program without launching separate applications.

Some of the many available IDEs include Eclipse from the Eclipse Foundation, Inc.; JGRASP, developed at Auburn University; NetBeans, downloadable from Oracle; and TextPad from Helios Software Solutions. Some IDEs are freely available, while others require a software license fee.

Skill Practice
with these end-of-chapter questions

1.7.1   Multiple Choice Exercises

Questions 9, 10, 11, 12, 13, 14

1.7.3   General Questions

Questions 27, 28, 29, 30

1.7.4   Technical Writing

Question 34

1.5.4 Programming Activity 1: Writing a First Java Application

Let's create our first Java program. This program prints the message, "Programming is not a spectator sport!"

Start by launching your IDE and open a new editor window. This is where you will write the code for the program.

Before you type any code, however, let's name the document. You do this by saving the document as *FirstProgram.java*. Be sure to capitalize the F and the P and keep the other letters lowercase. Java is case-sensitive, so Java considers *firstprogram.java* or even *Firstprogram.java* to be a different name. Also, note that there is no space between *First* and *Program*.

Keeping case sensitivity in mind, type in the program shown in Example 1.5.

```java
1 // First program in Java
2 // Anderson, Franceschi
3
4 public class FirstProgram
5 {
6   public static void main( String [ ] args )
7   {
8     System.out.println( "Programming is not a spectator sport!" );
9
10    System.exit( 0 );
11  }
12 }
```

EXAMPLE 1.5   A First Program in Java

At this point, we ask that you just type the program as you see it here, except for the line numbers, which are not part of the program. Line numbers are displayed in this example to allow easy reference to a particular line in the code. We'll explain a little about the program now; additional details will become clear as the semester progresses.

The first two lines, which start with two forward slashes, are comments. They will not be compiled or executed; they are simply information for the programmer and are used to increase the readability of the program.

Line 4 defines the class name as *FirstProgram*. Notice that the class name must be spelled exactly the same way—including capitalization—as the file name, *FirstProgram.java*.

The curly braces in lines 5 and 12 mark the beginning and ending of the *FirstProgram* class, and the curly braces in lines 7 and 11 mark the beginning and ending of the *main method*. Every Java application must define a class and a *main* method. Execution of a Java application always begins with the code inside *main*. So when this application begins, it will execute line 8, which writes the message "*Programming*

COMMON ERROR TRAP
Java is case-sensitive. The class name and the file name must match exactly, including capitalization.

is not a spectator sport!" to the system console. Next, it executes line 10, *System.exit(0)*, which exits the program. Including this line is optional; if you omit this line, the application will still exit normally.

As you type the program, notice that your IDE automatically colors your text to help you distinguish comments, *String* literals ("*Programming is not a spectator sport!*"), Java class names (*String, System*), and keywords (*public, class, static*), which are reserved for specific uses in Java. Curly braces, brackets, and parentheses, which have syntactical meaning in Java, are usually displayed in color as well. Your IDE may use different colors than those shown in Example 1.5.

When you have completed typing the code in Example 1.5, compile it. If everything is typed correctly, the compiler will create a *FirstProgram.class* file, which contains the byte codes for the program.

If you received any compiler errors, check that you have entered the code exactly as it is written in Example 1.5. We give you tips on finding and fixing the errors in the next section.

If you got a clean compile with no errors, congratulations! You're ready to execute the application. This will invoke the JVM and pass it the *First Program.class* file created by the compiler. If all is well, you will see the message, *Programming is not a spectator*

Figure 1.4

Output from Example 1.5

```
Programming is not a spectator sport!
```

sport!, displayed on the Java console, which is the text window that opens automatically. Figure 1.4 shows the correct output of the program.

Debugging Techniques

If the compiler found syntax errors in the code, these are called **compiler errors**, not because the compiler caused them, but because the compiler found them. When the compiler detects errors in the code, it writes diagnostic information about the errors.

For example, try typing *println* with a capital P (as *Println*), and recompiling. The compiler displays the following message:

```
FirstProgram.java:8: error: cannot find symbol
    System.out.Println( "Programming is not a spectator sport!" );
              ^
  symbol: method Println(String)
  location: variable out of type PrintStream
1 error
```

The first line identifies the file name that contains the Java source code, as well as the line number in the source code where the error occurred. In this case, the error occurred on line 8. The second line displays line 8 from the source code with a caret (^) pointing to *Println* as being the cause of the error. The symbol and location information in the third and fourth lines indicate that the *Println* method is unknown. Remember that Java is case-sensitive, so *println* and *Println* are considered to be different. As you gain experience with Java, these error messages will become more meaningful to you. With most IDEs, double-clicking on the first line in the error message transfers you to the source code window with your cursor positioned on line 8 so you can correct the error.

Many times, the compiler will find more than one error in the source code. When that happens, don't panic. Often, a single problem, such as a missing semicolon or curly brace, can cause multiple compiler errors.

For example, after correcting the preceding error, try deleting the left curly brace in line 7, then recompiling. The compiler reports four errors:

```
FirstProgram.java:6: error: ';' expected
        public static void main( String [ ] args )
                                                  ^
FirstProgram.java:10: error: <identifier> expected
        System.exit( 0 );
                  ^
FirstProgram.java:10: error: illegal start of type
        System.exit( 0 );
                     ^
FirstProgram.java:12: error: class, interface or enum expected
}
^
4 errors
```

As you can see, the compiler messages do not always report the problem exactly. When you receive a compiler message, looking at the surrounding lines will often help you find the error. Depending on your IDE, you might see messages other than those shown here because some IDEs attempt to interpret the error messages from the compiler to provide more relevant information on the errors.

It is sometimes easier to fix one error at a time and recompile after each fix, because the first fix might eliminate many of the reported errors.

When all the compiler errors are corrected, you're ready to execute the program.

It is possible to get a clean bill of health from the compiler, yet the program still won't run. To demonstrate this, try eliminating the brackets in line 6 after the word

SOFTWARE ENGINEERING TIP
Because one syntax error can cause multiple compiler errors, correct only the obvious errors and recompile after each correction.

String. If you then compile the program, no errors are reported. But when you try to run the program, you get a **run-time error**.

Instead of *Programming is not a spectator sport!*, the following message is displayed on the Java console:

```
Error: Main method not found in class FirstProgram, please define the main
method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

This means that the *main* method header (line 6) was not typed correctly.

Thus, we've seen that two types of errors can occur while you are developing a Java program: compiler errors, which are usually caused by language syntax errors or misspellings, and run-time errors, which are often caused by problems using the prewritten classes. Run-time errors can also be caused by exceptions that the JVM detects as it is running, such as an attempt to divide by zero.

Testing Techniques

Once your program compiles cleanly and executes without run-time errors, you may be tempted to conclude that your job is finished. Far from it—you must also verify the results, or output, of the program.

In the sample program, it's difficult to get incorrect results—other than misspelling the message or omitting the spaces between the words. But any nontrivial program should be tested thoroughly before declaring it production ready.

To test a program, consider all the possible inputs and the corresponding correct outputs. It often isn't feasible to test every possible input, so programmers usually test **boundary conditions**, which are the values that sit on the boundaries of producing different output for a program.

For example, to test the code that determines whether an integer is negative or nonnegative, you would feed the program −1 and 0. These are the boundaries of negative and nonnegative integers. In other words, the boundary between negative and nonnegative integers is between −1 and 0.

When a program does not produce the correct output, we say the program contains **logic errors**. By testing your program thoroughly, you can discover and correct most logic errors. Table 1.7 shows types of program errors and their usual causes.

We'll talk more about testing techniques throughout the text.

TABLE 1.7   Types of Program Errors and Their Causes

Type of Error	Usual Causes
Compiler errors	Incorrect language syntax or misspellings
Run-time errors	Incorrect use of classes
Logic errors	Incorrect program design or incorrect implementation of the design

? **DISCUSSION QUESTIONS**

1. In the Debugging Techniques section, we saw that making one typo could generate several compiler errors. Why do you think that happens?

2. Explain why testing boundary conditions is an efficient way to verify a program's correctness.

3. Did any errors occur while you were developing the first application? If so, explain whether they were compiler or run-time errors and what you did to fix them.

1.5.5 Making a JAR File

After we finish writing, compiling, and testing our application, we may want to distribute it. Java programs are typically distributed as **JARs**, which stands for Java ARchive. The JAR format allows the compression and aggregation of multiple files into one. The *.jar* extension is, by default, associated with the Java Virtual Machine. Thus, a user can double-click on a JAR file and launch its corresponding application directly. Most IDEs include a tool to easily create a JAR file. But you can also create a JAR file from the command line. The general syntax to create a JAR file is:

```
jar cf jarfile inputfile(s)
```

- The *c* and *f* characters are options. The *c* option means that we want to create a JAR file. The *f* option means that we want to send the output to a file.

- *jarfile* is the name of the JAR file that we want to create; it can be a different name from the name of our Java file(s).

- inputfile(s) is a space-separated list of files that we want to include in the JAR file. It typically includes the .class files and may include folders or files containing resources used in our application, such as images or sounds.

We can create a JAR file named *Greeting.jar* from our *FirstProgram* application by typing the following at the command line:

```
jar cf Greeting.jar FirstProgram.class
```

Note that the input file is the *.class* file, not the *.java* file. Figure 1.5 shows the contents of the folder after executing the command above.
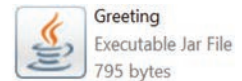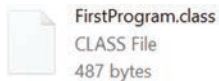
The file *Greeting.jar* is created in the current folder, as shown in Figure 1.5.

If our application includes several Java files, we can use the wildcard character * to include all the corresponding .class files. The following creates a JAR file for all the .class files located in the current folder:

```
jar cf Greetings.jar *.class
```

Figure 1.5

The Folder After the JAR File Is Created

FirstProgram.class
CLASS File
487 bytes

FirstProgram
jGRASP Java file
247 bytes

Greeting
Executable Jar File
795 bytes

CHAPTER REVIEW

- Basic components of a computer include the CPU, memory, a hard disk, keyboard, and monitor.

- Each type of CPU has its own set of instructions for performing arithmetic and logical operations, moving data, and changing the order of execution of instructions.

- An operating system controls peripheral devices, supports multitasking, allocates memory to programs, and prevents the user from damaging the system.

- Computer networks link two or more computers so that they can share resources, such as files or printers.

- The Internet connects millions of computers around the world. Web servers deliver webpages to clients running Internet browsers.

- Binary numbers are composed of bits. Each bit has the value 0 or 1. A byte holds eight binary digits.

- To convert a binary number to a decimal number, multiply each digit in the binary number by $2^{position-1}$, counting the rightmost position as position 1 and moving left through the number. Then add the products together.

- To convert a decimal number into a binary number, first find the largest power of 2 that is smaller than or equal to the decimal number, subtract that number from the decimal number, then do the same with the remainder, and so on, until the decimal number reaches 0.

- Hexadecimal digits can be used to represent groups of four bits.

- The Unicode character set, which Java uses, can encode up to 65,536 characters using 16 bits per character.

- Machine language and assembly language are early forms of programming languages that require the programmer to write to the CPU's instruction set. Because this low-level programming is time consuming and difficult, and the programs are not portable to other CPU architectures, machine language and assembly language are rarely used.

- High-level languages are highly symbolic and somewhat portable. They can be compiled, interpreted, or as in the case of Java, converted to byte codes, which are interpreted at run time by the Java Virtual Machine.

- A good program is readable, easily maintainable, and reusable.

- Object-oriented programming uses classes to encapsulate data and the functions needed to manipulate that data. Objects are instantiated according to the class design. An advantage to object-oriented programming is the ability to reuse classes.

- Programs use a combination of sequential processing, method calls, selection, and iteration to control the order of execution of instructions. Performing operations in order, one after another, is called sequential processing. Temporarily executing other code, then returning, is called a method call. Selecting which code to execute based on the value of data is called selection. Repeating the same code on each item in a group of values is called iteration, or looping.

- Pseudocode allows a programmer to design a program without worrying about the syntax of the language.

- In programming, we often perform tasks for which there are standard methods of processing, called algorithms. For example, accumulating is a common programming operation that finds the sum of a group of numbers.

- Programming, in large part, is reducing a complex task to a set of subtasks that can be implemented by combining standard algorithms that use sequential processing, selection, and looping.

- Java source code is stored in a text file with an extension of *.java*. Compiling the code produces one or more *.class* files.

- An Integrated Development Environment (IDE) consists of a program editor, a compiler, and a run-time environment, integrated via a Graphical User Interface.

- Compiler errors are detected by the compiler and are usually caused by incorrect Java syntax or misspellings. Run-time errors are detected by the Java Virtual Machine and are usually caused by exceptions or incorrect use of classes. Logic errors occur during program execution and are caused by incorrect program design.

1.7 Exercises, Problems, and Projects

1.7.1 Multiple Choice Exercises

1. Which one of these is not an operating system?
   - ❑ Linux
   - ❑ Java
   - ❑ Windows
   - ❑ Android

2. Which one of these is not an application?
   - ❑ Word
   - ❑ Firefox
   - ❑ Linux
   - ❑ Excel

3. How many bits are in three bytes?
   - ❑ 3
   - ❑ 8
   - ❑ 24
   - ❑ 0

4. In a network, the computers providing services to the other computers are called
   - ❑ clients.
   - ❑ servers.
   - ❑ laptops.

5. A binary number ending with a 0
   - ❑ is even.
   - ❑ is odd.
   - ❑ cannot tell.

6. A binary number ending with a 1

     ❑ is even.

     ❑ is odd.

     ❑ cannot tell.

7. A binary number ending with two 0s

     ❑ is a multiple of 4.

     ❑ is not a multiple of 4.

     ❑ cannot tell.

8. Using four bits, the largest positive binary number we can represent is 1111.

     ❑ true

     ❑ false

9. Which one of these is not a programming language?

     ❑ C++

     ❑ Java

     ❑ Windows

     ❑ Fortran

10. Which one of these is not an object-oriented programming language?

     ❑ C

     ❑ Java

     ❑ C++

     ❑ Smalltalk

11. What is the file extension for a Java source code file?

     ❑ .java

     ❑ .exe

     ❑ .class

12. What is the file extension of a compiled Java program?

     ❑ .java

     ❑ .exe

     ❑ .class

13. In order to compile a program named *Hello.java*, what do you type at the command line?

    ❑ java Hello

    ❑ java Hello.java

    ❑ javac Hello

    ❑ javac Hello.java

14. You have successfully compiled *Hello.java* into *Hello.class*. What do you type at the command line in order to run the application?

    ❑ java Hello.class

    ❑ java Hello

    ❑ javac Hello

    ❑ javac Hello.class

1.7.2   Converting Numbers

15. Convert the decimal number 67 into binary.

16. Convert the decimal number 1,564 into binary.

17. Convert the binary number 0001 0101 into decimal.

18. Convert the binary number 1101 0101 0101 into decimal.

19. Convert the binary number 0001 0101 into hexadecimal.

20. Convert the hexadecimal number D8F into binary.

1.7.3   General Questions

21. A RAM chip is organized as $\times 8$ memory, i.e., each unit contains 8 bits, or a byte. There are 7 address pins on the chip. How many bytes does that memory chip contain?

22. If a CPU is rated at 2.5 GHz, how many clock cycles per second are performed?

23. If a CPU's clock cycles 2.6 billion times per second, what is the rating of the CPU in MHz?

24. Suppose we are using binary encoding to represent colors. For example, a black-and-white color system has only two colors and therefore needs only 1 bit to encode the color system as follows:

Bit   Color

0   black

1   white

With 2 bits, we can encode four colors as follows:

Bit pattern   Color

00   black

01   red

10   blue

11   white

With 5 bits, how many colors can we encode?

With n bits (n being a positive integer), how many colors can we encode? (Express your answer as a function of n.)

25. In HTML, a color can be coded in the following hexadecimal notation: *#rrggbb*, where

rr represents the amount of red in the color

gg represents the amount of green in the color

bb represents the amount of blue in the color

rr, *gg*, and *bb* vary between 00 and FF in hexadecimal notation, i.e., 0 and 255 in decimal equivalent notation. Give the decimal values of the red, green, and blue values in the color #33AB12.

26. RGB is a color system representing colors: R stands for red, G for green, and B for blue. A color can be coded as *rgb* where *r* is a number between 0 and 255 representing how much red there is in the color, *g* is a number between 0 and 255 representing how much green there is in the color, and *b* is a number between 0 and 255 representing how much blue there is in the color. The color gray is created by using the same value for *r*, *g*, and *b*. How many shades of gray are there?

27. List three benefits of the Java programming language.

28. What is the name of the Java compiler?

29. Write the pseudocode for a program that finds the product of two numbers.

30. Write the pseudocode for a program that finds the sums of the numbers input that are greater than or equal to 10 and the numbers input that are less than 10.

1.7.4   Technical Writing

31. List the benefits of having a Local Area Network versus standalone computer systems.

32. For one day, keep a diary of the computer applications that you use. Also note any features of the applications that you think should be improved or any features you'd like to see added.

33. You are looking at two computers with the following specifications, everything else being equal:

    PC # 1   PC # 2

    3.4-GHz CPU   3.3-GHz CPU

    16 MB L2 cache   16 MB L2 cache

    1 GB RAM   4 GB RAM

    1 TB hard drive   1 TB hard drive

    $399 $399

    Which PC would you buy? Explain the reasoning behind your selection.

34. Go to Oracle's Java site (*www.oracle.com/technetwork/java*). Explain what resources are available there for someone who wants to learn Java.

1.7.5   Group Project (for a group of 1, 2, or 3 students)

35. In the octal system (base 8), numbers are represented using digits from 0 to 7; a 0 is placed in front of the octal number to indicate that the octal system is being used. For instance, here are some examples of the equivalent of some octal numbers in the decimal system:

Octal	Decimal
000	0
001	1
007	7
010	8
011	9

In the hexadecimal system, numbers are represented using digits from 0 to 9 and letters A to F; 0x is placed in front of the hexadecimal number to indicate that the hexadecimal system is being used. For instance, here are some examples of the decimal equivalents of some hexadecimal numbers:

Hexadecimal	Decimal
0x0	0
0x1	1
0x9	9
0xA	10
0xB	11
0xF	15
0x10	16
0x11	17
0x1C	28

1. Convert 0xC3E (in hexadecimal notation) into an octal number.
2. Convert 0377 (in octal notation) into a hexadecimal number.
3. Discuss how, in general, you would convert a hexadecimal number into an octal number and an octal number into a hexadecimal number.