CHAPTER

1

Creating Your First Java Program

In this chapter, you will:

- ♦ Learn about programming
- ♦ Understand object-oriented programming concepts
- ♦ Learn about the Java programming language
- ♦ Start a Java program
- ♦ Add comments to a Java program
- ♦ Run a Java program
- ♦ Modify a Java program

As you read your e-mail, a sinking feeling descends on you. There's no denying the message: "Please see me in my office as soon as you are free—Lynn Greenbrier." Lynn Greenbrier is the head of programming for Event Handlers Incorporated, and you have worked for her as an intern for only two weeks. Event Handlers manages the details of private and corporate parties; every client has different needs, and the events are interesting and exciting.

"Did I do something wrong?" you ask as you enter her office. "Are you going to fire me?"

Almost like a mind reader, Lynn stands to greet you and says, "Please wipe that worried look off your face! I want to see if you are interested in a new challenge. Our programming department is going to create several new programs in the next few months. We've decided that the Java programming language is the way to go. It's object-oriented, platform independent, and perfect for applications on the World Wide Web, which is where we want to expand our marketing efforts."

"I'm not sure what 'object-oriented' and 'platform independent' mean," you say, "but I've always been interested in computers, and I'd love to learn more about programming."

"Based on your aptitude tests, you're perfect for programming," Lynn says. "Let's get started now. I'll describe the basics to you."

Learning About Programming

A computer **program** is simply a set of instructions that you write to tell a computer what to do. Computers are constructed from circuitry that consists of small on/off switches, so you could write a computer program by writing something along the following lines:

```
first switch—on
second switch—off
third switch—off
fourth switch—on
```

Your program could go on and on, for several thousand switches. A program written in this style is written in **machine language**, which is the most basic circuitry-level language. The problems with this approach lie in keeping track of the many switches involved in programming any worthwhile task, and in discovering the errant switch or switches if the program does not operate as expected. Additionally, the number and location of switches varies from computer to computer, which means that you would need to customize a machine language program for every type of machine on which you want the program to run.

Fortunately, programming has evolved into an easier task because of the development of high-level programming languages. A **high-level programming language** allows you to use a vocabulary of reasonable terms such as "read," "write," or "add," instead of the sequences of on and off switches that perform these tasks. High-level languages also allow you to assign intuitive names to areas of computer memory, such as "hoursWorked" or "rateOfPay," rather than having to remember the memory locations (switch numbers) of those values.

Each high-level language has its own **syntax**, or rules of the language. For example, depending on the specific high-level language, you might use the verb "print" or "write" to produce output. All languages have a specific, limited vocabulary and a specific set of rules for using that vocabulary. Programmers use a computer program called a **compiler** to translate their high-level language statements into machine code. The compiler issues an error message each time the programmer uses the programming language incorrectly; subsequently, the programmer can correct the error and attempt another translation by compiling the program again. Other languages, including Java, use an **interpreter** to read the compiled code line by line at run time. A Java interpreter can run as a stand-alone program, or it can be part of a Web browser such as Netscape Navigator or Microsoft Internet Explorer where it can be invoked automatically to run applets in a Web page. The ability to run applets in a Web browser is one feature that distinguishes Java from other programming languages. When you are learning a computer programming language, such as the Java programming language, C++, or Visual Basic, you really are learning the vocabulary and syntax rules for that language.

In addition to learning the correct syntax for a particular language, a programmer also must understand computer programming logic. The **logic** behind any program involves executing the various statements and procedures in the correct order to produce the desired results. For example, you would not write statements to tell the computer program to process data until it had been properly read into the program. Similarly, you might be able to use a computer language's syntax correctly, but be unable to execute a logically constructed, workable program. Examples of logical errors include multiplying two values when you meant to divide them, or producing output prior to obtaining the appropriate input. Tools that will help you visualize and understand logic will be presented in Chapter 5.

Understanding Object-Oriented Programming Concepts

There are two popular approaches to writing computer programs: procedural programming and object-oriented programming.

Procedural programming involves using your knowledge of a programming language to create computer memory locations that can hold values—for example, numbers and text, in electronic form. The computer memory locations are called **variables** because they hold values that might vary. For example, a payroll program written for a company might contain a variable named rateOfPay. The memory location referenced by the name rateOfPay might contain different values (a different value for every employee of the company) at different times. During the execution of the payroll program, each value stored under the name rateOfPay might have many **operations** performed on it—the value might be read from an input device, the value might be multiplied by another variable representing hours worked, and the value might be printed on paper. For convenience, the individual operations used in a computer program are often grouped into logical units called **procedures**. For example, a series of four or five comparisons and calculations that together determine an individual's federal withholding tax value might be grouped as a procedure named calculateFederalWithholding. A procedural program defines the variable memory locations, and then **calls** a series of procedures to input, manipulate, and output the values stored in those locations. A single procedural program often contains hundreds of variables and thousands of procedure calls.

Object-oriented programming is an extension of procedural programming in which you take a slightly different approach to writing computer programs. Writing **object-oriented programs** involves both creating objects and creating applications that use those objects. Objects often interrelate with other objects, and once created, objects can be reused over and over again to develop new programs. Thinking in an object-oriented manner involves envisioning program components as objects that are similar to concrete objects in the real world; then you can manipulate the objects to achieve a desired result.

If you've ever used a computer that uses a command-line operating system (such as DOS), and if you've also used a GUI (graphical user interface, such as Windows), then you are familiar with the difference between procedural and object-oriented programs. If you want to move several files from a floppy disk to a hard disk, you can use either a typed command at a prompt or command line, or you can use a mouse in a graphical environment to accomplish the task. The difference lies in whether you issue a series of commands, in sequence, to move the three files, or you drag icons representing the files from one screen location to another, much as you would physically move paper files from one file cabinet to another in your office. You can move the same three files using either operating system, but the GUI system allows you to manipulate the files like their real-world paper counterparts. In other words, the GUI system allows you to treat files as objects.

Objects both in the real world and in object-oriented programming are made up of states and methods. The states of an object are commonly referred to as its attributes. **Attributes** are the characteristics that define an object as part of a class. For example, some of your automobile's attributes are its make, model, year, and purchase price. Other attributes include whether the automobile is currently running, its gear, its speed, and whether it is dirty. All automobiles possess the same attributes, but not, of course, the same values for those attributes. Similarly, your dog has the attributes of its breed, name, age, and whether his or her shots are current.

A **class** is a term that describes a group or collection of objects with common properties. An **instance** of a class is a technical term for an existing object of a class. Therefore, your red Chevrolet automobile with the dent can be considered an instance of the class that is made up of all automobiles, and your Golden Retriever dog named Goldie is an instance of the class that is made up of all dogs. Thinking of items as instances of a class allows you to apply your general knowledge of the class to individual members of the class. A particular instance of an object takes on, or **inherits**, its attributes from a more general category. If a general class Dog has defined attributes and methods, they can be passed on to a specific class Golden Retriever Dog with minimal programming effort. If your friend purchases an Automobile, you know it has a model name, and if your friend gets a Dog, you know the dog has a breed. You might not know the exact contents of your friend's Automobile, its current state or her Automobile's speed or her Dog's shots, but you do know what attributes exist for the Automobile and Dog classes. Similarly, in a GUI operating environment, you expect each component to have specific, consistent attributes, such as a menu bar and a title bar, because each component inherits these attributes as a member of the general class of GUI components.

Tip

By convention, programmers using the Java programming language begin their class names with an uppercase letter. Thus, the class that defines the attributes and methods of an automobile would probably be named Automobile, and the class for dogs would probably be named Dog. However, following this convention is not required to produce a workable program.

Besides attributes, objects can use methods to accomplish tasks. A **method** is a self-contained block of program code. Automobiles, for example, can move forward and backward. They can also be filled with gasoline or be washed, both of which can be programmed as methods to change some of their attributes. Methods exist for ascertaining certain attributes, such as the current speed of an Automobile and the current status of its gas tank. Similarly, a Dog can walk or run, eat food, and get a bath, and there are methods to determine how hungry the Dog is. GUI operating system components can be maximized, minimized, and dragged. Like procedural programs, object-oriented programs have variables (attributes) and procedures (methods), but the attributes and methods are encapsulated into objects that are then used much like real-world objects. **Encapsulation** refers to the hiding of data and methods within an object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a "black box," or a device that you can use without regard to the internal mechanisms. A programmer gets to access and use the methods and data contained in the black box but cannot change them.

If an object's methods are well written, the user is unaware of the low-level details of how the methods are executed, and the user must simply understand the interface or interaction between the method and the object. For example, if you can fill your Automobile with gasoline, it is because you understand the interface between the gas pump nozzle and the vehicle's gas tank opening. You don't need to understand how the pump works mechanically or where the gas tank is located inside your vehicle. If you can read your speedometer, it does not matter how the displayed figure is calculated. As a matter of fact, if someone produces a superior, more accurate speed-determining device and inserts it in your Automobile, you don't have to know or care how it operates, as long as your interface remains the same. The same principles apply to well-constructed objects used in object-oriented programs.

LEARNING ABOUT THE JAVA PROGRAMMING LANGUAGE

The **Java programming language** was developed by Sun Microsystems as an object-oriented language that is used both for general-purpose business programs and for interactive World Wide Web-based Internet programs. Some of the advantages that have made the Java programming language so popular in recent years are its security features, and the fact that it is **architecturally neutral**, which means that you can use the Java programming language to write a program that will run on any platform, or operating system.

Java can be run on a wide variety of computers because Java does not execute instructions on a computer directly. Instead, Java runs on a hypothetical computer known as the **Java virtual machine(JVM)**.

The Java Environment shown in Figure 1-1 is described as follows: the Java programming statements known as **source code** are first constructed using a text editor. Then a special compiler known as the Java compiler converts the source code into a binary program of **byte code**. A program called the **Java Interpreter** checks the byte code and executes the byte code instructions line by line within the Java virtual machine. Since the Java program is isolated from the native operating system, the Java program is insulated from the particular hardware on which it is run. Because of this insulation, the JVM provides security against intruders getting at your computer's hardware through the operating system. In contrast, when using other programming languages, software vendors usually have to produce multiple versions of the same product (a DOS version, Windows version, Macintosh version, Unix version, and so on) so all users can use the program. With the Java programming language, one program version will run on all these platforms.

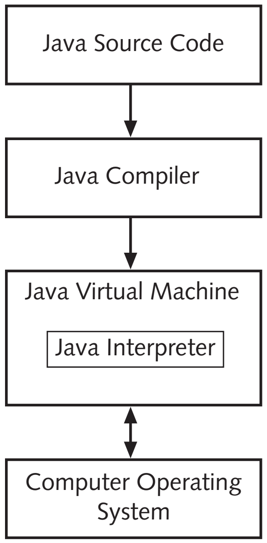


Figure 1-1 Java enviroment

For simplicity, the terms "Java program" and "program for the Java programming language" are used interchangeably throughout this text.

The Java programming language also is simpler to use than many other object-oriented languages. The Java programming language is modeled after the C++ programming language. Although neither language is "simple" to read or understand on first exposure, the Java programming language does eliminate some of the most difficult-to-understand features in C++, such as pointers and multiple inheritance.

Java Program Types

You can write two kinds of programs using Java. Programs that are embedded in a Web page are called Java **applets**. Stand-alone programs are called Java **applications**. Java applications can be further subdivided into **console** applications, which support character output to a computer screen in a DOS window, for example, and **windowed** applications, which create a graphical user interface (GUI) with elements such as menus, toolbars, and dialog boxes. Console applications are the easiest applications to create; we will start using them in the next section. Windowed applications will be introduced in Chapter 11.

STARTING A JAVA PROGRAM

At first glance, even the simplest Java program involves a fair amount of confusing syntax. Consider the following simple program. This program is written on seven lines, and its only task is to print "First Java program" on the screen.

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

The statement that does the actual work in this program is `System.out.println("First Java program");`.

Tip

All Java programming language statements end with a semicolon.

The text "First Java program" is a **literal string** of characters; that is, it is a series of characters that will appear exactly as entered. Any literal string in Java appears between double quotation marks, as opposed to single quotation marks as in 'First Java program'. Even though code is written on multiple lines for ease of reading, the literal string cannot be broken because parts of the literal string will appear on different lines.

The string "First Java program" appears within parentheses because the string is an argument to a method, and arguments to methods always appear within parentheses. **Arguments** consist of information that a method requires to perform its task. For example, you might place a catalog order with a company that sells sporting goods. Processing a catalog order is a method that consists of a set of standard procedures. However, each catalog order requires information such as which item number you are ordering and the quantity of the item desired; this information can be considered the order's argument. If you order two of item 5432 from a catalog, you expect different results than if you order 1,000 of item 9008. Likewise, if you pass the argument "Happy Holidays" to a method, you expect different results than if you pass the argument "First Java program".

Within the statement `System.out.println("First Java program");`, the method to which you are passing "First Java program" is named println(). The println() method prints a line of output on the screen, positions the insertion point on the next line, and stands ready for additional output.

Tip

Method names usually are referenced followed by their parentheses, as in println(), so you can distinguish method names from variable names.

Within the statement `System.out.println("First Java program");`, *out* is an object. The out object represents the screen. Several methods, including println(), are available with the out object. Of course, not all objects have a println() method (for instance, you can't print to a keyboard, to your Automobile, or to your Dog), but the creators of the Java platform assumed you frequently would want to display output on a screen. Therefore, the out object was created and endowed with the method named println(). In this chapter, you will create your own objects and endow them with your own methods.

Tip

The print() method is very similar to the println() method. With println(), after the message prints, the insertion point appears on the following line. With print(), the insertion point does not advance to a new line; it remains on the same line as the output.

Within the statement `System.out.println("First Java program");`, *System* is a class. Therefore, System defines the attributes of a collection of similar "System" objects, just as the Dog class defines the attributes of a collection of similar Dog objects. One of the System objects is *out*. (You can probably guess that another object is *in*, and that it represents an input device.)

Tip

The Java programming language is case sensitive; the class named System is a completely different class from one named system, SYSTEM, or even sYsTeM.

The dots (periods) in the statement `System.out.println("First Java program");` are used to separate the names of the class, object, and method. You will use this same class-dot-object-dot-method format repeatedly in your Java programs.

The statement that prints the string "First Java program" is embedded in the program shown in Figure 1-2.

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

Figure 1-2 Printing a string

Everything that you use within a Java program must be part of a class. When you write `public class First`, you are defining a class named `First`. You can define a Java class using any name or identifier you need, as long as it meets the following requirements:

- A class name must begin with a letter of the alphabet (which includes any non-English letter, such as α or π), an underscore, or a dollar sign.

- A class name can contain only letters, digits, underscores, or dollar signs.
- A class name cannot be a Java programming language reserved keyword, such as `public` or `class` (see Figure 1-3 for a list of reserved keywords).
- A class name cannot be one of the following values: `true`, `false`, or `null`.

Tip: The Java programming language is based on Unicode, which is an international system of character representation. The term *letter* indicates English-language letters, as well as characters from Arabic, Greek, and other alphabets.

abstract	float	private
Boolean	for	protected
break	future	public
byte	generic	rest
byvalue	goto	return
case	if	short
cast	implements	static
catch	import	super
char	inner	switch
class	instanceof	synchronized
const	int	this
continue	interface	throw
default	long	throws
do	native	transient
double	new	try
else	null	var
extends	operator	void
final	outer	volatile
finally	package	while

Figure 1-3 Java programming language reserved keywords

It is a Java programming language standard to begin class names with an uppercase letter and employ other uppercase letters as needed to improve readability. Table 1-1 lists some valid and conventional class names for the Java programming language.

Table 1-1 Some valid class names in the Java programming language

Class Name	Description
Employee	Begins with an uppercase letter
UnderGradStudent	Begins with an uppercase letter, contains no spaces, and emphasizes each new word with an initial uppercase letter
InventoryItem	Begins with an uppercase letter, contains no spaces, and emphasizes the second word with an initial uppercase letter
Budget2004	Begins with an uppercase letter and contains no spaces

Tip

You should follow established conventions for the Java programming language so your programs will be easy for other programmers to interpret and follow. This book uses established Java programming conventions.

Table 1-2 Some unconventional class names in the Java programming language

Class Name	Description
employee	Begins with a lowercase letter
Undergradstudent	New words are not indicated with initial uppercase letters; difficult to read
Inventory_Item	The underscore is not commonly used to indicate new words
BUDGET2004	Appears as all uppercase letters

Table 1-3 Some illegal class names in the Java programming language

Class Name	Description
an employee	Space character is illegal
Inventory Item	Space character is illegal
class	`class` is a reserved word
2001Budget	Class names cannot begin with a digit
phone#	The # symbol is not allowed

In Figure 1-2, the line `public class First` contains the keyword `class`, which identifies First as a class. The reserved word `public` is an access modifier. An **access modifier** defines the circumstances under which a class can be accessed. Public access is the most liberal type of access; you will learn about public and other types of access in Chapter 3.

You enclose the contents of all classes within curly braces ({ and }). A class can contain any number of data items and methods. In Figure 1-2, the class First contains only one method within its curly braces. The name of the method is main(), and the main() method contains its own set of parentheses and only one statement—the println() statement.

Tip

In general, whitespace is optional in the Java programming language. Whitespace is any combination of spaces, tabs, and carriage returns (blank lines). However, you cannot use whitespace within any identifier or keyword. You can insert whitespace between words or lines in your program code by typing spaces, tabs, or blank lines because the compiler will ignore these extra spaces. You use whitespace to organize your program code and make it easier to read.

For every opening curly brace ({) in a Java program, there must be a corresponding closing curly brace (}). The placement of the opening and closing curly braces is not important to the compiler. For example, the following method is executed exactly the same as the one shown in Figure 1-2. The only difference is that the method is organized differently. This location of the curly braces is common in professional code. Usually, code in which you vertically align each pair of opening and closing curly braces is easier to read and is used throughout this textbook. Strive to type your code so it is easy to read.

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

The method header for the main() method is quite complex. The meaning and purpose of each of the terms used in the method header will become clearer as you complete this textbook; a brief explanation will suffice for now.

In the method header `public static void main(String[] args)`, the word `public` is an access modifier, just as it is when you define the First class. In the English language, the word *static* means showing little change, or stationary. In the Java programming language, the reserved keyword **`static`** ensures that main() is accessible even though no objects of the class exist. It also indicates that every member created for the First class will have an identical, unchanging main() method. Within the Java programming language, `static` also implies uniqueness. Only one main() method for the First class will ever be stored in the memory of the computer. Of course, other classes eventually might have their own, different main() methods.

In English, the word *void* means empty. When the keyword **`void`** is used in the main() method header, it does not indicate that the main() method is empty, but rather, that the main() method does not return any value when it is called. This doesn't mean that main() doesn't produce output—in fact, the method does. The main() method does not send any value back to any other method that might use it. You will learn more about return values in Chapter 3.

Not all classes have a main() method; in fact many do not. All Java applications however, must include a method named main(), and most Java applications have additional methods. When you execute a Java application, the compiler always executes the main() method first.

In the method header `public static void main(String[] args)`, you already might recognize that the contents between the parentheses, `(String[] args)`, must represent an argument passed to the main() method, just as the string "First Java program" is an argument passed to the println() method. `String` represents a Java class that can be used to represent character strings. The identifier `args` is used to hold any Strings that might be sent to the main() method. The main() method could do something with those arguments, such as print them, but in Figure 1-2 the main() method does not actually use the `args` identifier. Nevertheless, you must place an identifier within the main() method's parentheses. The identifier does not need to be named `args`—it could be any legal Java identifier—but the name `args` is traditional.

Tip

When you refer to the String class in the main() method header, the square brackets indicate an array of String objects. You will learn more about arrays and the String class in Chapter 6.

The simple program shown in Figure 1-2 has many pieces to remember. However, for now, you can use the program shown in Figure 1-4 as a shell, where you replace the line /******/ with any statements that you want to execute.

```
public class First
{
   public static void main(String[] args)
   {
     /******/
   }
}
```

Figure 1-4 Shell output program

Now that you understand the basic framework of a program written in the Java programming language, you are ready to enter your first Java program into a text editor. It is a tradition among programmers that the first program you write in any language produces "Hello, world!" as its output. You will create such a program now. You can use any text editor, such as Notepad, Textpad, or any other text-processing program.

To write your first Java program:

1. Start any text editor (such as Notepad or Textpad, but be sure the file is saved with the extension .txt), and then open a new document, if necessary. (Textpad is the easiest program to use to write your programs.)
2. Type the class header **`public class Hello`**. In this example, the class name is `Hello`. You can use any valid name you want for the class. If you choose Hello, you must refer to the class as Hello, and not as hello, because the Java programming language is case sensitive.
3. Press **[Enter]** once, type **{**, press **[Enter]** again, and then type **}**. You will add the main() method between the curly braces. Although it is not required, it is a good practice to place each curly brace on its own line. This makes your code easier to read.
4. As shown in Figure 1-5, add the main() method header between the curly braces and then type a set of curly braces for main().

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

Figure 1-5 The main() method shell for the Hello class

Next add the statement within the main() method's brackets that will produce the output, "Hello, world!".

5. Use Figure 1-6 as a guide for adding a println() statement to the main() method.

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

Figure 1-6 main() method for the Hello class

6. Save the program as **Hello.java** in the Chapter.01 folder on your Student Disk. It is important that the file extension is .java. If it is not, the compiler for the Java programming language will not recognize the program.

Help: Many text editors attach their own filename extension (such as .txt or .doc) to a saved file. Double-check your saved file to ensure that it does not have a double extension (such as Hello.java.txt). If the file has a double extension, rename the file. If you explicitly type quotation marks surrounding a filename (such as "Hello.java"), most text editors will save the file as you specify, without adding an extension. Make sure that you save your .java files as text documents. The default for Notepad is to save all documents as text.

ADDING COMMENTS TO A JAVA PROGRAM

As you can see, even the simplest Java program takes several lines of code, and contains somewhat perplexing syntax. Large programs that perform many tasks include much more code, and as you write longer programs, it becomes increasingly difficult to remember why you included steps, or how you intended to use particular variables. **Program comments** are nonexecuting statements that you add to a program for the purpose of documentation. Programmers use comments to leave notes for themselves and for others who might read their programs in the future. At the very least, your programs should include comments indicating the program's author, the date, and the program's name or function. The best practice dictates that a brief comment be made to tell the purpose for each class and its methods.

Tip: It is suggested that as you work through this book you add comments as the first three lines of every program. The comments should contain the program name, your name, and the date. Your instructor might ask you to include additional comments.

Comments also can serve a useful purpose when you are developing a program. If a program is not performing as expected, you can comment out various statements and subsequently run the program to observe the effect. When you **comment out** a statement, you turn it into a comment so the compiler will not execute its command. This helps you pinpoint the location of errant statements in malfunctioning programs.

There are three types of comments in the Java programming language:

- **Line comments** start with two forward slashes (//) and continue to the end of the current line. Line comments can appear on a line by themselves or at the end of a line following executable code.
- **Block comments** start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/). Block comments can appear on a line by themselves, on a line before executable code, or after executable code. Block comments also can extend across as many lines as needed.
- A special case of block comments are **javadoc** comments. They begin with a forward slash and two asterisks (/**) and end with an asterisk and a forward slash (*/). You can use javadoc comments to generate documentation with a program named javadoc.

Tip: The forward slash (/) and the backslash (\) characters often are confused, but they are two distinct characters. You cannot use them interchangeably.

Tip: The Java Development Kit (JDK) includes the javadoc tool, which contains classes that you can use when writing programs in the Java programming language.

Figure 1-7 shows how comments are used in code.

```
//Demonstrating comments
/* This shows
   that these comments
        don't matter */
System.out.println("Hello");//This line executes
     // up to where the comment started
/**Everything but the println() line
 is a comment. */
```

Figure 1-7 Using comments in a program

Next you will add comments to your Hello.java program.

To add comments to your program:

1. Position the insertion point at the top of the file, press **[Enter]** to insert a new line, press the **Up arrow** key to go to that line, and then type the following comments at the top of the file. Press **[Enter]** after typing each line. Insert your name and today's date where indicated.

   ```
   // Filename Hello2.java
   // Written by <your name>
   // Written on <today's date>
   ```

2. Scroll to the end of the line that reads `public class Hello`, change the class name to **`Hello2`**, press **[Enter]**, and then type the following block comment in the program:

   ```
   /*  This program demonstrates the use of the println()
   method to print the message Hello, world!  */
   ```

3. Save the file as **Hello2.java** in the Chapter.01 folder on your Student Disk.

Running a Java Program

After you write and save your program, there are two steps that must occur before you can view the program output.

1. You must compile the program you wrote (called the source code) into bytecode.
2. You must use the Java interpreter to translate the bytecode into executable statements.

To compile your source code from the command line, you type `javac` followed by the filename of the file that contains the source code. For example, to compile a file named First.java, you would type `javac First.java` and then press [Enter]. There will be one of three outcomes:

- You receive a message such as `Bad command or filename`.
- You receive one or more program language error messages.
- You receive no messages, which means that the program compiled successfully.

Tip: When compiling, if the source code file is not in the current path, you can type a full path with the filename, for example, `javac c:\java\myprograms\First.java`.

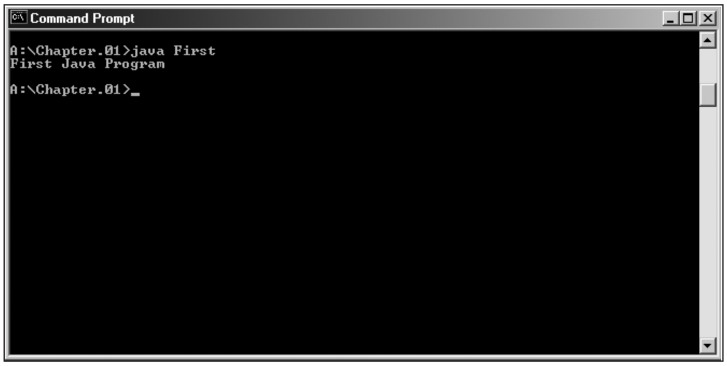
If you receive a message such as `Bad command or filename`, it might mean one of the following:

- You misspelled the command `javac`.
- You misspelled the filename.
- You are not within the correct subfolder or subdirectory on your command line.
- The Java programming language was not installed properly. (See Appendix for information on installation.)

If you receive a programming language error message, then there are one or more syntax errors in the source code. A **syntax error** is a programming error that occurs when you introduce typing errors into your program. For example, if your class name is "first" (with a lowercase f) in the source code, but you saved the file as First.java, you will get an error message, such as `public class first should not be defined in First.java`, after compiling the program because "first" and "First" are not the same in a case-sensitive language. If this error occurs, you must reopen the text file that contains the source code and make the necessary corrections.

If you receive no error messages after compiling the code in a file named First.java, then the program compiled successfully, and a file named First.class was created and saved in the same folder as the program text file. After a successful compile, you can run the class file on any computer that has a Java language interpreter.

To run the program from the command line, you type `java First`. Figure 1-8 shows the program's output. Next you will compile and interpret your Hello2.java program.



```
A:\Chapter.01>java First
First Java Program

A:\Chapter.01>_
```

Figure 1-8 Output of the First program

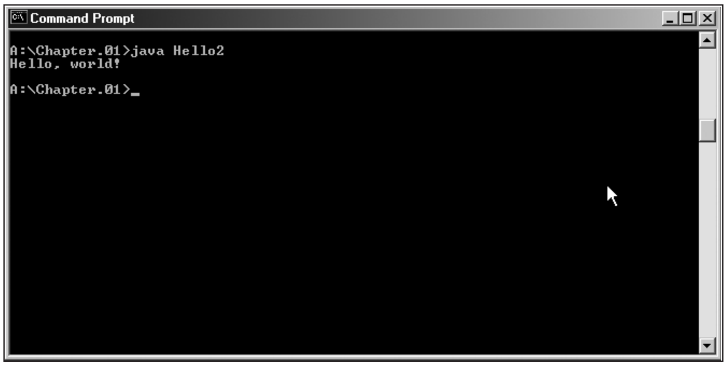
To compile and interpret your Hello2.java program with comments:

1. Go to the command-line prompt for the drive and folder or subdirectory in which you saved Hello2.java.
2. At the command line, type **`javac Hello2.java`**.

Help

If you receive an error message, look in the section "Running a Java Program" to find its cause, and then make the necessary corrections. Save the file again, and then repeat Steps 1 and 2 until your program compiles successfully.

3. When the compile is successful, execute your program by typing **`java Hello2`** at the command line, and then press **[Enter]**. The output should appear on the next line, as shown in Figure 1-9.



```
A:\Chapter.01>java Hello2
Hello, world!

A:\Chapter.01>_
```

Figure 1-9 Output of the Hello2 program

Tip

When you run a Java program using the `java` command, do not add the .class extension to the filename. If you type `java First`, the interpreter will look for a file named First.class. If you type `java First.class`, the interpreter will incorrectly look for a file named First.class.class.

Modifying a Java Program

After viewing the program output, you might decide to modify the program to get a different result. For example, you might decide to change the First program's output from `First Java program` to the following:

```
My new and improved
Java program
```

To produce the new output, first you must modify the text file that contains the existing program. You may also want to change the class name to First2 to indicate that the First program has changed. You will need to change the literal string that currently prints, and then add an additional text string. Figure 1-10 shows the program to change the output.

```
public class First2
{
   public static void main(String[] args)
   {
    System.out.println("My new and improved");
    System.out.println("Java program");
   }
}
```

Figure 1-10 Changing a program's output

The three changes are the class name change of First to First2, the addition of the statement `System.out.println("My new and improved");`, and the removal of the word "First" from the string in the statement `System.out.println("Java' program");`. However, if you type `java First2` at the command line right now, you will not see the new output—you will see the old output. Before the new source code will execute, you must do the following:

1. Save the file with the changes using the same filename (First2.java).
2. Compile the First2 class with the `javac` command.
3. Interpret the First2.class bytecode with the `java` command.

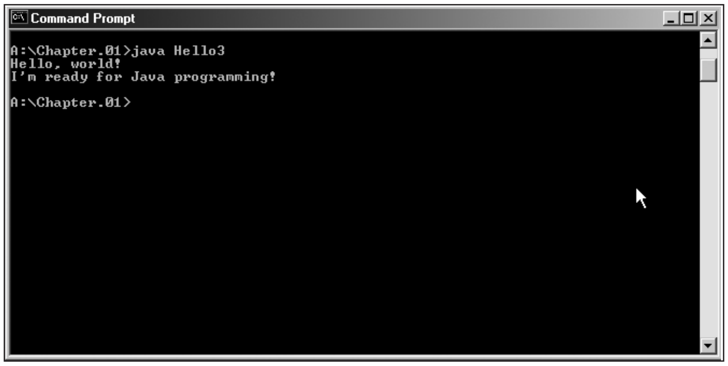
Next you will change your Hello2 class and rerun your program.

To change the Hello2 class and rerun the program:

1. Open the file **Hello2.java** in your text editor. Change both the comment name and the class name to **Hello3**.
2. Add the following statement below the statement that prints "Hello, world!": **`System.out.println("I'm ready for Java programming!");`**. Make sure to type the semicolon at the end of the statement and use the correct case.
3. Save the file as **Hello3.java** in the Chapter.01 folder on your Student Disk. Changing the Hello2 class name to Hello3 and saving the file as Hello3 calls attention to the change being made to the Hello2 class.
4. At the command line, compile the file by typing the command **`javac Hello3.java`**.

If you receive compile errors, return to the Hello3.java file in the text editor, fix the errors, and then repeat Steps 3 and 4 until the program compiles successfully.

5. Interpret and execute the class by typing the command **`java Hello3`**, and then press **[Enter]**. Your output should look like Figure 1-11.



```
A:\Chapter.01>java Hello3
Hello, world!
I'm ready for Java programming!

A:\Chapter.01>
```

Figure 1-11 Output of the revised Hello program

Errors and Debugging

When typing errors are made as code is entered and compiled, the compiler will produce an error message as explained in the examples earlier in the chapter. The exact ror message depends on the compiler. In the First program of Figure 1-2, typing the `System.out.println()` code as `system.out.println("First Java Program");` produces an error message similar to "cannot resolve symbol…". This is a compile-time error commonly referred to as a syntax error. The compiler detects the violation of language rules and refuses to translate the program to machine code. The compiler will try to report as many errors as it can find during compilation so that you can fix as many errors as possible. Sometimes one error in syntax causes subsequent errors that normally would not be errors if the first syntax error did not exist. You should correct the errors that make sense to you and then recompile.

A second kind of error occurs when the syntax of the program is correct and the program is compiled but produces incorrect results. This is a run-time error or logic error. In the First program of Figure 1-1, typing the `System.out.println()` code as `System.out.println("Frst Java Program";` does not produce an error. The compiler does not find the spelling error of "Frst" instead of "First". Errors of this type must be detected by carefully examining the program output. It is the responsibility of the program author to test programs and find any logic errors. Good programming practice stresses programming structure and development that helps minimize errors. Additionally, each chapter in this book has four programs for debugging that will give you practice finding and correcting syntax and logic errors.

FAQs, JDKs, Documentation, and Tutorials

A great wealth of material exists at the Sun Microsystems Web site, *http://java.sun.com*. Of particular value are the FAQs (Frequently Asked Questions) that you can link to from a site search of the Web site. By searching the site for FAQs, you can locate the collections of frequently asked questions that provide brief answers to many common questions about Java software and products. A site search for the Java Development Kit (JDK) will help you locate and download the latest JDK for free. At the time of this writing the latest version is The Java™ 2 Platform, Standard Edition v1.4.0, available in Windows, Linux, and Solaris operating systems. You can search and browse documentation online or you can download the documentation file for the SDK and install it on your computer. Once installed, you can search and browse documentation locally.

A downloadable Java tutorial titled "The Java Tutorial: A practical guide for programmers", with hundreds of complete working examples is available from *http://java.sun.com/docs/books/tutorial/*. The tutorial is organized into trails—groups of lessons on a particular subject. You can start the tutorial at the beginning and navigate sequentially from beginning to end, or jump from one trail to another. As you study each textbook chapter you are encouraged to make good use of these support materials.

Chapter Summary

- Objects are made up of states and methods. The states of an object also are known as its attributes. An individual object is an instance of a class; the object inherits its attributes from the class. The user of an object does not need to understand the details of any method, but must understand the interface with the object.
- A program written in Java is run on a standardized hypothetical computer called the Java virtual machine (JVM) that exists virtually inside your machine by a program. When your program is compiled into bytecode, an interpreter within the JVM subsequently interprets the bytecode and interfaces with the operating system to produce the program results.
- All Java programming language statements end with a semicolon. Periods (called dots) are used to separate classes, objects, and methods in program code. The contents of all classes are contained within opening and closing curly braces. A series of characters that appears between double quotation marks is a literal string. Java programming language methods might require arguments or messages to perform the appropriate task.
- Everything that you use within a Java program must be part of a class. A Java programming language class might take any name or identifier that begins with either an uppercase or lowercase letter of the alphabet, and contains only uppercase and lowercase letters, digits, and underscores. A class name cannot be a reserved keyword of the Java programming language.
- The reserved word `public` is an access modifier that defines the circumstances under which a class can be accessed. The keyword `static` in a method header indicates that every member of a class will have an identical, unchanging method. The keyword `void` in a method header indicates that the method does not return any value when it is called.

- All Java application programs must have a method named main(). Most Java applications have additional methods.
- Program comments are nonexecuting statements that you add to a program for the purpose of documentation. There are three types of comments in the Java programming language: line comments begin with two forward slashes (//); block comments begin with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/); and javadoc comments begin with a forward slash and two asterisks (/**) and end with an asterisk and a forward slash (*/).
- To compile your source code from the command line, type `javac` followed by the name of the file that contains the source code. If the file resides in a different path from the command prompt, use the full path and filename. When you compile your source code, the compiler creates a file with a .class extension. You can run the .class file on any computer that has a Java language interpreter by entering the `java` command followed by the name of the class file. Do not type the .class extension with the filename.
- When you modify a program, before it will execute correctly, you must do the following: save the file with the changes using the same filename, compile the class with the `javac` command, and interpret the class bytecode with the `java` command.
- To avoid and minimize syntax and logic errors you must enter code carefully and closely examine your program's output.

REVIEW QUESTIONS

1. The most basic circuitry-level computer language, which consists of on and off switches, is ________________.
   a. a high-level language
   b. machine language
   c. the Java programming language
   d. C++
2. Languages that let you use a vocabulary of descriptive terms such as "read," "write," or "add" are known as ________________ languages.
   a. high-level
   b. machine
   c. procedural
   d. object-oriented

3. The rules of a programming language constitute its ________________.
    a. objects
    b. logic
    c. format
    d. syntax
4. A ________________ translates high-level language statements into machine code.
    a. programmer
    b. syntax detector
    c. compiler
    d. decipherer
5. Computer memory locations are called ________________.
    a. compilers
    b. variables
    c. addresses
    d. appellations
6. For convenience, the individual operations used in a computer program are often grouped into logical units called ________________.
    a. procedures
    b. variables
    c. constants
    d. logistics
7. Envisioning program components as objects that are similar to concrete objects in the real world is the hallmark of ________________.
    a. command-line operating systems
    b. procedural programming
    c. object-oriented programming
    d. machine languages
8. An object's attributes also are known as its ________________.
    a. states
    b. orientations
    c. methods
    d. procedures
9. An instance of a(n) ________________ inherits its attributes from it.
    a. object
    b. procedure

    c. method
    d. class

10. The Java programming language is architecturally ________.
    a. specific
    b. oriented
    c. neutral
    d. abstract
11. You must compile programs written in the Java programming language into ________.
    a. bytecode
    b. source code
    c. javadoc statements
    d. object code
12. All Java programming language statements must end with a ________.
    a. period
    b. comma
    c. semicolon
    d. closing parenthesis
13. Arguments to methods always appear within ________.
    a. parentheses
    b. double quotation marks
    c. single quotation marks
    d. curly braces
14. In a Java program, you must use ________ to separate classes, objects, and methods.
    a. commas
    b. semicolons
    c. periods
    d. forward slashes
15. All Java programs must have a method named ________.
    a. method()
    b. main()
    c. java()
    d. Hello()

16. Nonexecuting program statements that provide documentation are called ________________.
    a. classes
    b. notes
    c. comments
    d. commands
17. The Java programming language supports three types of comments: ________________, ________________, and javadoc.
    a. line, block
    b. string, literal
    c. constant, variable
    d. single, multiple
18. After you write and save a program file, you ________________ it.
    a. interpret and then compile
    b. interpret and then execute
    c. compile and then resave
    d. compile and then interpret
19. The command to execute a compiled program is ________________.
    a. `run`
    b. `execute`
    c. `javac`
    d. `java`
20. You save text files containing Java language source code using the file extension ________________.
    a. .java
    b. .class
    c. .txt
    d. .src
21. The Java Virtual machine, or JVM, refers to a(n) ________________.
    a. interpreter
    b. operating system
    c. hypothetical computer
    d. compiler

EXERCISES

1. For each of the following Java programming language identifiers, note whether they are legal or illegal:
   a. weeklySales
   b. last character
   c. class
   d. MathClass
   e. myfirstinitial
   f. phone#
   g. abcdefghijklmnop
   h. 23jordan
   i. my_code
   j. 90210
   k. year2000problem
   l. abffraternity
2. Name some attributes that might be appropriate for each of the following classes:
   a. TelevisionSet
   b. EmployeePaycheck
   c. PatientMedicalRecord
3. Write, compile, and test a program that prints your first name on the screen. Save the program as **Name.java** in the Chapter.01 folder on your Student Disk.
4. Write, compile, and test a program that prints your full name, street address, city, state, and zip code on three separate lines on the screen. Save the program as **Address.java** in the Chapter.01 folder on your Student Disk.
5. Write, compile, and test a program that displays the following pattern on the screen:

```
   X
  XXX
 XXXXX
XXXXXXX
   X
```

   Save the program as **Tree.java** in the Chapter.01 folder on your Student Disk.

6. Write, compile, and test a program that prints your initials on the screen. Compose each initial with five lines of initials, as in the following example:

   ```
       J  FFFFFF
       J  F
       J  FFFF
   J   J  F
   JJJJJ  F
   ```

   Save the program as **Initial.java** in the Chapter.01 folder on your Student Disk.
7. Write, compile, and test a program that prints all the objectives listed at the beginning of this chapter. Save the program as **Objectives.java** in the Chapter.01 folder on your Student Disk.
8. Write, compile, and test a program that displays the following pattern on the screen:

   ```
     *
    * *
   * * *
    * *
     *
   ```

   Save the program as **Diamond.java** in the Chapter.01 folder on your Student Disk.
9. Write, compile, and test a program that displays the following statement about comments:

   "Program comments are nonexecuting statements you add to a program for the purpose of documentation."

   Also include the same statement in three different comments in the program; each comment should use one of the three different methods of including comments in a Java program. Save the program as **Comments.java** in the Chapter.01 folder on your Student Disk.
10. Each of the following files in the Chapter.01 folder on your Student Disk has syntax and/or logical errors. In each case, determine the problem and fix the program. After you correct the errors, save each file using the same filename preceded with Fix. For example, DebugOne1.java will become FixDebugOne1.java.
    a. DebugOne1.java
    b. DebugOne2.java
    c. DebugOne3.java
    d. DebugOne4.java

Case Project

Business Cards Limited is a company that designs and prints personal business cards. They have asked you to write a simple program using Java to provide personal business card information for a typical order. The format to be displayed is shown in the table below:

Table 1-4 CardLayout format

Information Layout
First Name Last Name
Address 1
Address 2
City State Zip
Home Phone
Work Phone

Write, compile, and test a Java program to print the table layout using your own personal information. Be sure to use good documentation principles for your program. Save the program as **CardLayout** in the Chapter.01 folder on your Student Disk.