CHAPTER 2

FIRST JAVA PROGRAMS

OBJECTIVES

Upon completion of this chapter, you should be able to:

- Discuss why Java is an important programming language.
- Explain the Java virtual machine and byte code.
- Choose a user interface style.
- Describe the structure of a simple Java program.
- Write a simple program.
- Edit, compile, and run a program using a Java development environment.
- Format a program to give a pleasing, consistent appearance.
- Understand compile-time errors.
- Write a simple graphics program.

Estimated Time: 3.5 hours

VOCABULARY

- Applet
- Assignment operator
- Byte code
- DOS development environment
- Graphical user interface (GUI)
- Hacking
- Integrated development environment (IDE)
- Java virtual machine (JVM)
- Just-in-time compilation (JIT)
- Parameter
- Source code
- Statement
- Terminal I/O user interface
- Variable

Programs are written in programming languages, and the language used in this book is Java. This chapter gets you up and running with a couple of simple Java programs. We show how to write these first programs, compile them, and run them. In the process, you will become acquainted with a Java programming environment, the structure of a simple Java program, and the basic ideas of variables, input and output (I/O) statements, and sending messages to objects.

2.1 Why Java?

Java is the fastest growing programming language in the world. Companies such as IBM and Sun have adopted Java as their major application development language. There are several reasons for this.

First, Java is a modern object-oriented programming language. The designers of Java spent much time studying the features of classical object-oriented languages such as Smalltalk and C++ and made a successful effort to incorporate the good features of these languages and omit the less desirable ones.

Second, Java is secure, robust, and portable. That is, the Java language

- Enables the construction of virus-free, tamper-free systems (secure)
- Supports the development of programs that do not overwrite memory (robust)
- Yields programs that can be run on different types of computers without change (portable)

These features make Java ideally suited to develop distributed, network-based applications, which is an area of ever-increasing importance.

Third, Java supports the use of advanced programming concepts such as threads. A *thread* is a process that can run concurrently with other processes. For example, a single Java application might consist of two threads. One thread transfers an image from one machine to another across a network, while the other thread simultaneously interacts with the user.

Fourth and finally, Java bears a superficial resemblance to C++, which is currently the world's most popular industrial-strength programming language. Thus, it is easy for a C++ programmer to learn Java and for a Java programmer to learn C++. Compared to C++, however, Java is easier to use and learn, less error prone, more portable, and better suited to the Internet.

On the negative side, Java runs more slowly than most modern programming languages because it is interpreted. To understand this last point we must now turn our attention to the Java virtual machine and byte code.

EXERCISE 2.1

1. What is a portable program?
2. Describe two features of Java that make it a better language than C++.
3. What is a thread? Describe how threads might be used in a program.

2.2 The Java Virtual Machine and Byte Code

Compilers usually translate a higher-level language into the machine language of a particular type of computer. However, the Java compiler translates Java not into machine language, but into a pseudomachine language called Java ***byte code***. Byte code is the machine language for an imaginary Java computer. To run Java byte code on a particular computer, you must install a ***Java virtual machine*** (JVM) on that computer, unless you use a computer such as Apple's Macintosh, in which the JVM comes with the operating system.

A JVM is a program that behaves like a computer. Such a program is called an *interpreter*. An interpreter has several advantages as well as some disadvantages. The main disadvantage of an interpreter is that a program pretending to be a computer runs programs more slowly than an actual computer. JVMs are getting faster every day, however. For instance, some JVMs translate byte code instructions into machine language when they are first encountered—called ***just-in-time compilation*** (JIT)—so that the next time the instruction is encountered it is executed as fast machine code rather than being interpreted as slow byte code. Also, new computer chips are being developed that implement a JVM directly in hardware, thus avoiding the performance penalty.

The main advantage of an interpreter is that any computer can run it. Thus, Java byte code is highly portable. For instance, many of the pages you download on the Web contain small Java programs already translated into byte code. These are called ***applets***, and they are run in a JVM that is incorporated into your Web browser. These applets range from the decorative (displaying a comical animated character on a Web page) to the practical (displaying a continuous stream of stock market quotes).

Because Java programs run inside a virtual machine, it is possible to limit their capabilities. Thus, ideally, you never have to worry about a Java applet infecting your computer with a virus, erasing the files on your hard drive, or stealing sensitive information and sending it across the Internet to a competitor. In practice, however, computer hackers have successfully penetrated Java's security mechanisms in the past and may succeed again in the future. But all things considered, Java applets really are very secure, and security weaknesses are repaired as soon as they become known.

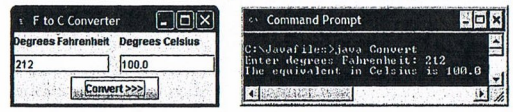
EXERCISE 2.2

1. What does JVM stand for?
2. What is byte code? Describe how the JVM uses byte code.
3. What is an applet? Describe how applets are used.

2.3 Choosing a User Interface Style

Before writing our first program, we must make a difficult decision. What type of user interface do we want to use? There are two choices: the ***graphical user interface*** (GUI), familiar to all PC users, and the less common ***terminal I/O user interface***. Figure 2-1 illustrates both in the context of a program that converts degrees Fahrenheit to degrees Celsius. The graphical user interface on the left is familiar and comfortable. The user enters a number in the first box, clicks the command button, and the program displays the answer in the second box. The terminal-based interface on the right begins by displaying the prompt "Enter degrees Fahrenheit: ". The user then enters a number and presses the Enter key. The program responds by displaying the answer.

FIGURE 2-1
Two user interfaces for a temperature conversion program

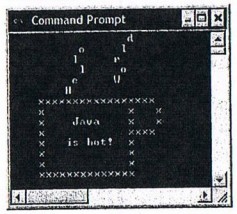


We use terminal I/O user interface in most of the program examples in this book. Beginning with this chapter, an optional end-of-chapter section introduces various aspects of graphics and GUI programming. In the long run, you will discover that this book's core material is independent of interface issues. There are three reasons for using terminal I/O. First, in Java and many other languages, a terminal user interface is easier to implement than a GUI, although in other languages, such as Visual BASIC, the opposite is true. Second, there are programming situations that require terminal I/O rather than a GUI, so familiarity with the techniques of terminal-oriented programming is important. Third, terminal-oriented programs are similar in structure to programs that process files of sequentially organized data, and what we learn here will be transferable to that setting.

2.4 *Hello World*

In conformance with a long and honorable tradition dating back to the early days of the language C, a textbook's first program often does nothing more than display the words "Hello World" in a terminal window. Actually, as you can see in Figure 2-2, we could not resist adding a few embellishments. In case you have not guessed, the imagery is the words "Hello World" rising like steam from the cup of hot Java.

FIGURE 2-2
Hello World



The Source Code

Just as a recipe is a sequence of instructions for a chef, a program is a sequence of instructions for a computer. And just as a recipe does nothing until executed by a chef, so a program does nothing until executed by a computer. With that in mind, the following is the bulk of the instructions, or *source code*, for our HelloWorld program:

```
System.out.println("              d          ");
System.out.println("          o   l          ");
System.out.println("        l     r          ");
System.out.println("         l    o          ");
System.out.println("        e     W          ");
System.out.println("       H                 ");
System.out.println("   xxxxxxxxxxxxxxxxx      ");
System.out.println("   x           x   x      ");
System.out.println("   x   Java    x   x      ");
System.out.println("   x           xxxx       ");
System.out.println("   x  is hot!  x          ");
System.out.println("   x           x          ");
System.out.println("   x           x          ");
System.out.println("   xxxxxxxxxxxxx          ");
```

The Explanation

In this code

- `System.out` is the name of an object that knows how to display or print characters in a terminal window.
- `println` is the name of the message being sent to the `System.out` object.
- The strings enclosed in quotation marks contain the characters to be printed.
- Semicolons (;) mark the end of each *statement* or sentence in the program.

As mentioned at the end of Chapter 1, an object-oriented program accomplishes its tasks by sending messages to objects. In this program, a `System.out` object responds to a `println` message by printing a string of characters in the terminal window. The string of characters that appears between the parentheses following the message is called a *parameter*. Some messages require several parameters, separated from each other by commas, whereas other messages have no parameters. The "ln" in the message `println` stands for "line" and indicates that the `System.out` object should advance to the beginning of the next line after printing a string.

Sending messages to objects always takes the form

```
<name of object>.<name of message>(<parameters>)
```

The period (`.`) between the object's name and the message's name is called a *method selector operator*. The period between the words `System` and `out` is not a method selector operator. For now you can just think of it as part of the object's name.

The Larger Framework

The program as presented so far is not complete. It must be embedded in a larger framework defined by several additional lines of code. No attempt will be made to explain this code until a

later chapter, but, fortunately, it can be reused with little change from one program to the next. Following then is the complete program with the new lines shown in color:

```
// Example 2.1: Our first program

public class HelloWorld{

   public static void main(String [] args) {
      System.out.println("                d          ");
      System.out.println("              o     l      ");
      System.out.println("            l     r        ");
      System.out.println("             l     o       ");
      System.out.println("            e     W        ");
      System.out.println("           H                ");
      System.out.println("     xxxxxxxxxxxxxxxxx      ");
      System.out.println("     x             x   x    ");
      System.out.println("     x    Java     x   x    ");
      System.out.println("     x             xxxx     ");
      System.out.println("     x   is hot!   x        ");
      System.out.println("     x             x        ");
      System.out.println("     x             x        ");
      System.out.println("     xxxxxxxxxxxxxx         ");
   }
}
```

To reuse the framework, replace `HelloWorld` with the name of another program:

```
public class <name of program> {
   public static void main(String [] args) {
      . . . put the source code here . . .
   }
}
```

In this text, we write program comments in green, reserved words in blue, and the rest of the program code in black. Program comments and reserved words will be explained in Chapter 3.

EXERCISE 2.4

1. Give a short definition of "program."
2. What is the effect of the message `println`?
3. Describe how to use the `System.out` object.
4. Write a sequence of statements to display your name, address, and phone number in the terminal window.

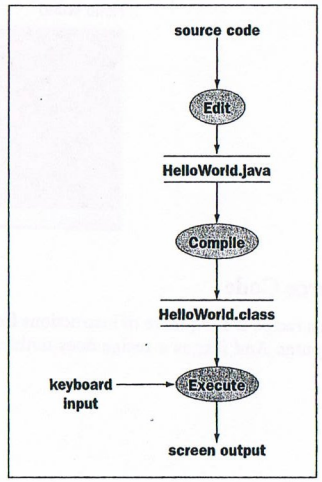
2.5 Edit, Compile, and Execute

In the preceding section, we presented the source code for our first program. Now we discuss how to enter it into a computer and run it. There are three steps:

1. *Edit.* In the first step, the programmer uses a word processor or editor to enter the source code into the computer and save it in a text file. The name of the text file must match the name of the program with the extension `.java` added, as in `HelloWorld.java`.
2. *Compile.* In the second step, the programmer invokes the Java language compiler to translate the source code into Java byte code. In this example, the compiler translates source code in the file `HelloWorld.java` to byte code in the file `HelloWorld.class`. The extension for a byte code file is always `.class`.
3. *Execute.* In the third step, the programmer instructs the JVM to load the byte code into memory and execute it. At this point the user and the program can interact, with the user entering data and the program displaying instructions and results.

Figure 2-3 illustrates the steps. The ovals represent the processes edit, compile, and execute. The names of the files `HelloWorld.java` and `HelloWorld.class` are shown between parallel lines.

FIGURE 2-3
Editing, compiling, and running a program



Development Environments

The details involved in editing, compiling, and running a program vary with the development environment being used. Some common development environments available to Java programmers include the following:

- UNIX or Linux using a standard text editor with command-line activation of the compiler and the JVM. UNIX is available on any Macintosh computer that runs MacOS X.
- Various versions of Microsoft Windows using Notepad for the editor with command-line activation of the compiler and the JVM from inside a command or DOS window. We call this the ***DOS development environment.***
- Windows or MacOS using an ***integrated development environment*** (IDE) such as Metrowerks' Code Warrior, Microsoft's Visual J++, Borland's Jbuilder, or free educational-use-only IDEs such as BlueJ and JGrasp.

The first two options are free and may require you to download and install the Java software development kit (SDK) as described in Appendix A. The third option, an integrated development environment, may cost money, but it has the advantage of combining an editor, a Java compiler, a debugger, and a JVM in a manner intended to increase programmer productivity. IDEs take time to master, however, and they can obscure fundamental details of the edit, compile, and run sequence.

Because we cannot possibly discuss all of these environments simultaneously, we give our instructions in terms of the DOS development environment that has the most widespread use. Macintosh users can use the UNIX command prompt and TextEdit. The installation and use of some of the major alternatives are presented in our supplemental materials on the book's Web site (the URL is in Appendix A).

Preparing Your Development Environment

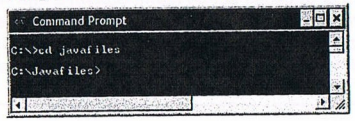
Before writing your first program, you must install a Java development environment on your computer. Guidelines for doing this are presented in Appendix A.

Step-by-Step Instructions

We are now ready to present step-by-step instructions for editing, compiling, and running the `HelloWorld` program. These instructions apply to users of the Windows XP system. After reading what follows, read the supplemental material for an explanation that matches the development environment on your computer.

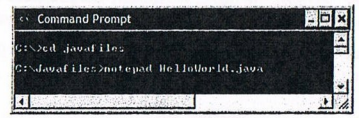
Step 1. Use Windows Explorer to create the directory in which you intend to work (for instance, C:\Javafiles). Open a terminal window by selecting **Command Prompt** (or something similar) on the **Start/All Programs/Accessories** menu. In the terminal window, use the **cd** command to move to the working directory as illustrated in Figure 2-4.

FIGURE 2-4
Using the **cd** command to move to the working directory



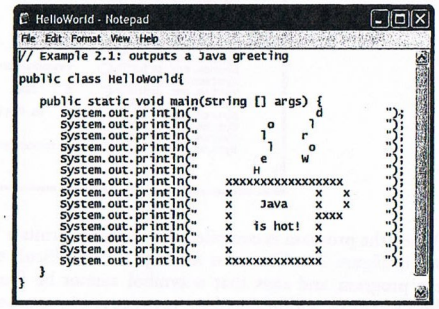
Step 2. Open the Notepad editor and create the file `HelloWorld.java` by typing the text as shown in Figure 2-5.

FIGURE 2-5
Activating Notepad to edit the program



Once Notepad opens, type in the lines of code for the program. Figure 2-6 shows the Notepad window after the program has been entered.

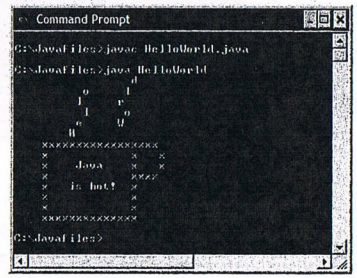
FIGURE 2-6
The program as typed into Notepad



Step 3. Save the file and switch back to the terminal window. Compile the program by typing `javac HelloWorld.java`. The DOS prompt returns when the compilation is complete.

Step 4. Run the program by typing `java HelloWorld`. Figure 2-7 illustrates this step as well as the previous step.

FIGURE 2-7
Compiling and running the program

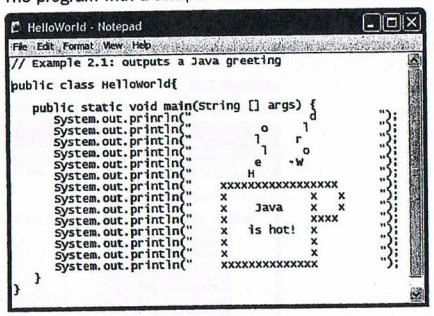


Compile-Time Errors

It is inevitable that we will make typographical errors when we edit programs, and the compiler will nearly always detect them. Mistakes detected by the compiler are called *syntax errors* or *compile-time errors*. To illustrate these, we modify the program so that it includes a syntax error. After reading this subsection, read the supplemental material that matches your development environment.

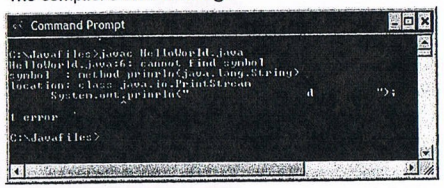
On line 6 of the program we misspell `println` as `prinrln`. Figure 2-8 shows the program with the error as it appears in Notepad.

FIGURE 2-8
The program with a compile-time error on line 6



When the program is compiled, the compiler prints a list of errors in the terminal window as shown in Figure 2-9. The error message is not difficult to understand. It refers to line 6 of the source program and says that a symbol cannot be found. More specifically, the symbol for method `prinrln` within the class `java.io.PrintStream` cannot be found. In fact, the object `System.out` is a member of this class, and the program has attempted to send a message that the class does not recognize. A caret symbol (^) points to the location of the error in the code. Unfortunately, some error messages may be difficult to decipher, but at least they indicate where the compiler encountered text it could not translate into byte code.

FIGURE 2-9
The compiler's error message



Readability

Programs typically have a long life and are usually maintained by many people other than their original authors. For this reason, if for no other, it is extremely important to write programs that are highly readable. The main factor affecting a program's readability is its layout. Indentation, the inclusion of blank lines and spaces, and other typographical considerations make the difference between an intelligible program and an incomprehensible mess. Interestingly, the compiler completely ignores a program's format, provided that there are no line breaks in the middle of words or quoted strings. Throughout the book, we attempt to format our programs in a pleasing and consistent manner, and you should strive to do the same. For your enjoyment the following example is a very unreadable but completely functional rendering of the `HelloWorld` program:

```
public class
     HelloWorld
{public static void main (String [] args) {System.out.println(
"           d         ");System.out.println
       ("          o     l          ");
System.out.println("          l     r          ");
 System.   out
.println("          l     o          ")
; System.out.println("          e     W          ");System.out
.println("          H                ");System.out.println
("    xxxxxxxxxxxxxxxxx      ");System.out.println(
"    x           x   x     ")
;System.out.println("    x    Java    x   x     ");
      System.out.println("    x            xxxx      ");
            System.out.println("    x   is hot!  x  '      ");
System.out.println("    x           x         ");
      System.out.println("    x         x         ");
           System.out.println("    xxxxxxxxxxxxx          "); } }
```


Computer Ethics

INTRUSIVE HACKING

Hacking is a term whose use goes back to the early days of computing. In its original sense, a "hack" is a programmer who exhibits rare problem-solving ability and commands the respect of other programmers. The culture of hackers began in the late 1950s at the MIT computer science labs. These programmers, many of them students and later professionals and teachers in the field, regarded hacking as an accomplishment along the lines of Olympic gymnastics. These programmers even advocated a "hacker ethic," which stated, among other things, that hackers should respect the privacy of others and distribute their software for free. For a narrative of the early tradition of hacking, see Steven Levy, *Hackers: Heroes of the Computer Revolution* (Garden City, New York: Anchor Press/Doubleday, 1984).

Unfortunately, the practice of hacking has changed over the years, and the term has acquired darker connotations. Programmers who break into computer systems in an unauthorized way are called hackers, whether their intent is just to impress their peers or to cause actual harm. Students and professionals who lack a disciplined approach to programming are also called hackers. An excellent account of the most famous case of intrusive hacking can be found in Clifford Stoll, *The Cuckoo's Egg: Tracking Through the Maze of Computer Espionage* (New York: Doubleday, 1989).

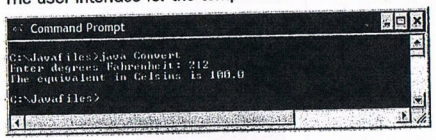
EXERCISE 2.5

1. Name the three steps in writing and running a program.
2. What are compile-time errors?
3. Find the compile-time errors in the following statements:
   a. System.out.println("Here is an error);
   b. System.out.println("Here is another error";
4. Why is readability a desirable characteristic of a program?

2.6 Temperature Conversion

We now present code for the temperature conversion program illustrated earlier in the chapter. To refresh your memory, we show the user interface again in Figure 2-10. This program is fundamentally more interesting than the `HelloWorld` program because it reads user inputs and performs computations. Despite its brevity and simplicity, the program demonstrates several important concepts.

FIGURE 2-10
The user interface for the temperature conversion program



The Source Code

The program's source code is

```
// Example 2.2: inputs degrees Fahrenheit
// from the keyboard and outputs degrees Celsius

import java.util.Scanner;

public class Convert{

   public static void main(String [] args){
      Scanner reader = new Scanner(System.in);
      double fahrenheit;
      double celsius;

      System.out.print("Enter degrees Fahrenheit: ");
      fahrenheit = reader.nextDouble();

      celsius = (fahrenheit - 32.0) * 5.0 / 9.0;

      System.out.print("The equivalent in Celsius is ");
      System.out.println(celsius);
   }
}
```


The Explanation

Following is a line-by-line explanation of the most significant portions of the program.

```
import java.util.Scanner;
```

The program's first line of code is an *import statement*. The program must read inputs entered at the keyboard, and this functionality is provided by something called a scanner object. Such objects are instances of the class `Scanner`. In this line of code, we are telling the compiler where to find complete specifications for the class. The periods that appear in this statement are NOT method selectors.

```
Scanner reader = new Scanner(System.in);
```

In this statement, we instantiate or create a `Scanner` object. We have arbitrarily decided to call the object `reader`. The name suggests what the object does, so it is a good choice. As mentioned in Chapter 1, an object is always an instance of a class and must be created, or instantiated, before being used. In general, instantiation is done like this:

```
SomeClass someObject = new SomeClass(some parameters);
```

The code `System.in` names a variable in the `System` class that refers to the keyboard. This object is passed as a parameter to the code that instantiates the `Scanner` object in order to connect the two objects. Parameters are used to share information between objects.

```
double fahrenheit;
double celsius;
```

In these statements, we declare that the program will use two numeric variables called `fahrenheit` and `celsius`. A numeric *variable* names a location in RAM in which a number can be stored. The number is usually referred to as the variable's *value*. During the course of a program, a variable's value can change, but its name remains constant. The variables in this program are of type `double`, which means they will contain only floating-point numbers. It is customary, though not required, to begin variable names with a lowercase letter, thus `fahrenheit` rather than `Fahrenheit`. We are allowed to declare as many variables as we want in a program, and we can name them pretty much as we please. Restrictions are explained in Chapter 3.

```
System.out.print("Enter degrees Fahrenheit: ");
```

This statement is similar to those we saw in the `HelloWorld` program, but there is a minor difference. The message here is `print` rather than `println`. A `print` message positions the cursor immediately after the last character printed rather than moving it to the beginning of the next line.

```
fahrenheit = reader.nextDouble();
```

In this statement, the `reader` object responds to the message `nextDouble` by waiting for the user to type a number and then press Enter, at which point the `reader` object returns the number to the program. The number is then assigned to the variable `fahrenheit` by means of the *assignment operator* (=). The number entered by the user is now stored in the variable. Note that although the `nextDouble` message has no parameters, the parentheses are still required. As the user types at the keyboard, the characters are automatically echoed in the terminal window, but not until the user presses Enter does this input become available to the program.

```
celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
```

In this statement, the expression to the right of the assignment operator (=) is evaluated, and then the resulting value is stored in memory at location `celsius`. Statements utilizing an assignment operator are called *assignment statements*. When the computer evaluates the expression, it uses the value stored in the variable `fahrenheit`. Notice that all the numbers (32.0, 5.0, and 9.0) contain a decimal point. In Java, some unexpected rules govern what happens when integers and floating-point numbers are mixed in an expression, so until we discuss the rules in Chapter 3, we will not mix integers and floating-point numbers. In the expression, as in algebra, the following symbols are used:

*	indicates the multiplication operator
/	indicates the division operator
-	indicates the subtraction operator

Of course, there is another common operator, namely + for *addition*. Notice the use of parentheses in the previous expression. In Java, as in algebra, multiplication and division are done before addition and subtraction unless parentheses are used to change the order of the computations; in other words, multiplication and division have higher precedence than addition and subtraction.

```
System.out.print("The equivalent in Celsius is ");
```

Here the `System.out` object prints the string "The equivalent in Celsius is ". The cursor is positioned after the last character in preparation for the next line of code.

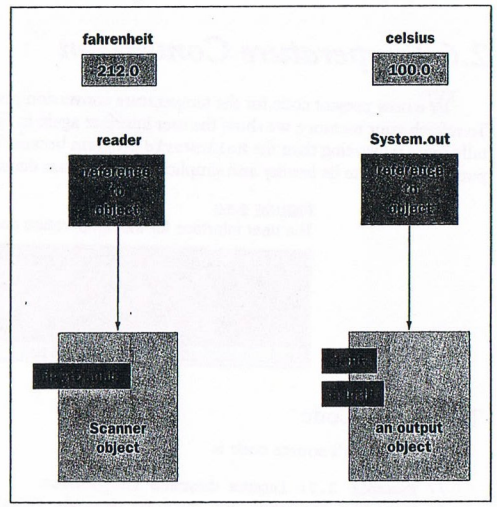
```
System.out.println(celsius);
```

Here the `System.out` object prints the value of the variable `celsius`. The parameter for a `print` or `println` message can be a string in quotation marks, a variable, or even an expression. When a variable is used, the variable's value is printed, not its name. When an expression is used, the expression is evaluated before its value is printed.

Variables and Objects

Figure 2-11 depicts four of the variables and objects used in the program. All of these exist in the computer's memory while the program is running. The variables `fahrenheit` and `celsius` each hold a single floating-point number. At any given instant, the value stored in a variable depends on the effect of the preceding lines of code. The variables `reader` and `System.out` are very different from the variables `fahrenheit` and `celsius`. Instead of holding numbers, they hold references to objects. The arrows in the figure are intended to suggest this fact. During the course of the program, we think of the `reader` variable as being the name of an object. As the figure indicates, we know nothing about what lies inside the `reader` object (information hiding), but we do know that it responds to the message `nextDouble`. `System.out` also names an object, but one that is never declared in our programs. How this can be so is explained in a later chapter. The `System.out` object responds to the messages `print` and `println`. One of the really significant facts about object-oriented programming is that we can use objects without having the least idea of their internal workings. Likewise, we can design objects for others to use without telling them anything about the implementation details.

FIGURE 2-11
Variables and objects used in the conversion program



EXERCISE 2.6

1. What is a variable in a program and how is it used?
2. Describe the role of the assignment (=) operator in a program.
3. What is a `Scanner` object?
4. Explain the difference between a variable of type `double` and a variable of type `Scanner`.
5. Describe the difference between `print` and `println`, and give an appropriate example of the use of each.

2.7 Graphics and GUIs: Windows and Panels

Java comes with a large array of classes that support graphics and GUI programming. In this section, we examine how to set up and manipulate an application window, explore the use of colors, and lay out regions within the window.

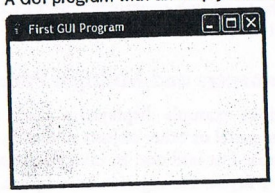
Extra Challenge

This is our first end-of-chapter section on graphics and GUIs. In these sections, we give you an opportunity to explore the concepts and programming techniques required to develop modern graphics applications and GUIs. None of this material is required for the other chapters of the book. But if you elect to go through it, you will learn how to write programs that display colors and geometric shapes, allow the user to interact by manipulating a mouse, animate shapes and images, and employ various "widgets" such as command buttons, text fields, sliders, and drop-down menus to accomplish useful tasks.

A Simple Application Window

Graphics and GUI programs in Java can run either as standalone applications or as applets. We discuss applets, which run in a Web browser, in Chapter 8. A standalone GUI application runs in a window. The window for our first GUI application is shown in Figure 2-12. The visual appearance or "look and feel" of a window might vary from computer to computer, but several features are constant.

FIGURE 2-12
A GUI program with an empty window



The window has a title bar that displays the message "First GUI Program." The user can drag the window to another position on the desktop by moving the mouse cursor to the title bar and then clicking and dragging.

The title bar contains some controls that allow the user to minimize the window (moving it to the desktop tray or dock), zoom it to full screen size, or close it (which usually quits the application).

The window has an initial width and height that the user can modify by selecting its lower-right corner and dragging appropriately.

Other than exhibiting the basic features and common behavior of all GUI applications, our first GUI application displays no other GUI components and does nothing. Here is the code for the application, followed by an explanation.

```
// Example 2.3: an empty frame

import javax.swing.*;      // Access JFrame

public class GUIWindow{

   public static void main(String[] args){
      JFrame theGUI = new JFrame();
      theGUI.setTitle("First GUI Program");
      theGUI.setSize(300, 200);
      theGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      theGUI.setVisible(true);
   }
}
```

The code for application windows in Java is located in the class `JFrame`, which is imported from the package `javax.swing`. The `main` method simply creates an instance of `JFrame` and sends it messages to set up and display the window. Unlike a terminal I/O application, the program does not quit when the end of the `main` method is reached but stays alive until the user selects the window's close box.

A `JFrame` object responds to many messages, among them messages to set its title to a given string, set its initial size to a width and height in pixels, set the operation that will be performed when it closes (exits the application), and set its visibility (true/visible or false/invisible). A set of commonly used `JFrame` methods is listed in Table 2-1.

TABLE 2-1
Some commonly used `JFrame` methods

JFRAME METHOD	WHAT IT DOES
`Container getContentPane()`	Returns the frame's container to which components can be added
`void setResizable(boolean b)`	If `b` is `false`, the user cannot resize the window; if `b` is `true`, the user can resize the window. The default is that the window is resizable.
`void setDefaultCloseIndicator(int i)`	Sets the operation to be performed when the user closes the frame
`void setSize(int width, int height)`	Sets the size of the frame to the width and height in pixels
`void setTitle(String title)`	Displays the title in the frame's title bar
`void setVisible(boolean b)`	Displays the frame if `b` is `true` or hides it if `b` is `false`

Panels and Colors

An application window is really just an empty container that we can fill with other objects. One such object is called a *panel*. A panel is a flat, rectangular area suitable for displaying other objects such as geometric shapes and images. Windows are often organized into multiple panels or *panes*, each of which contains related objects such as images and widgets. Panels themselves have fairly simple features, including a width, height, and background color. The class `JPanel`, also in `javax.swing`, represents panels in Java.

As we mentioned in Chapter 1, colors in most computer systems use the RGB scheme, which encodes 16,777,216 distinct colors. The `Color` class, which appears in the package `java.awt`, can be used to create any of these color values, as follows:

```
Color aColor = new Color(redValue, greenValue, blueValue)
```

where the red, green, and blue values are integers ranging from 0 to 255. Recall that 255 indicates the maximum intensity of a color component, whereas 0 indicates the absence of that component. Thus, the code `new Color(0, 0, 0)` would create an object representing the color black. For convenience, the `Color` class also includes constants for several commonly used colors, which are listed with their RGB values in Table 2-2.

TABLE 2-2
Some `Color` constants

COLOR CONSTANT	RGB VALUE
Color.red	new Color(255, 0, 0)
Color.green	new Color(0, 255, 0)
Color.blue	new Color(0, 0, 255)
Color.yellow	new Color(255, 255, 0)
Color.cyan	new Color(0, 255, 255)
Color.magenta	new Color(255, 0, 255)
Color.orange	new Color(255, 200, 0)
Color.pink	new Color(255, 175, 175)
Color.black	new Color(0, 0, 0)
Color.white	new Color(255, 255, 255)
Color.gray	new Color(128, 128, 128)
Color.lightGray	new Color(192, 192, 192)
Color.darkGray	new Color(64, 64, 64)

Our next example program creates a panel, sets its background color to pink, and adds the panel to the application window.

```
// Example 2.4: a frame with an empty, pink panel

import javax.swing.*;     // For JFrame and JPanel
import java.awt.*;        // For Color and Container

public class GUIWindow{

   public static void main(String[] args){
      JFrame theGUI = new JFrame();
      theGUI.setTitle("Second GUI Program");
      theGUI.setSize(300, 200);
      theGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
      panel.setBackground(Color.pink);
      Container pane = theGUI.getContentPane();
      pane.add(panel);
      theGUI.setVisible(true);
   }
}
```

When this program is run, its window looks just like that of the first program, except that the area below the title bar is pink. Note the procedure for adding the panel to the window. We must first obtain the window's container object by running the method `getContentPane()`. We then add the panel to this container.

Layout Managers and Multiple Panels

The previous example displayed a single panel in an application window. When we have more than one panel or other objects to display in a window, we have to be concerned about how they are organized or laid out. In Java, each container object, such as a frame or a panel, uses an object called a *layout manager* to accomplish this. Thus, when a program adds an object to a container, the container's layout manager actually influences its placement. Each type of container has a default layout manager, which we can reset to a different type of layout manager if the default does not suit our needs.

The default layout manager for frames is an instance of the class `BorderLayout`. A border layout allows us to arrange up to five objects in positions that correspond to the directions north (top), east (right), south (bottom), west (left), and center. If we add fewer than five objects, the layout manager stretches some of them to fill the unoccupied areas. To see what these areas look like when they are all occupied, we modify our second program to add five colored panels to the

window. The north and south panels are red, the east and west panels are blue, and the center panel is white. The result is displayed in Figure 2-13.

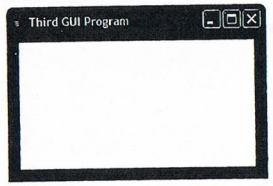
```
// Example 2.5: a frame with 5 colored panels
// that show the border layout

import javax.swing.*;     // For JFrame and JPanel
import java.awt.*;        // For Color and Container

public class GUIWindow{

   public static void main(String[] args){
      JFrame theGUI = new JFrame();
      theGUI.setTitle("Third GUI Program");
      theGUI.setSize(300, 200);
      theGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel northPanel = new JPanel();
      northPanel.setBackground(Color.red);
      JPanel eastPanel = new JPanel();
      eastPanel.setBackground(Color.blue);
      JPanel southPanel = new JPanel();
      southPanel.setBackground(Color.red);
      JPanel westPanel = new JPanel();
      westPanel.setBackground(Color.blue);
      JPanel centerPanel = new JPanel();
      centerPanel.setBackground(Color.white);
      Container pane = theGUI.getContentPane();
      pane.add(northPanel, BorderLayout.NORTH);
      pane.add(eastPanel, BorderLayout.EAST);
      pane.add(southPanel, BorderLayout.SOUTH);
      pane.add(westPanel, BorderLayout.WEST);
      pane.add(centerPanel, BorderLayout.CENTER);
      theGUI.setVisible(true);
   }
}
```

FIGURE 2-13
A border layout with five panels



Note the use of the `BorderLayout` constants to specify the area of the container to which an object is added. When the constant is omitted, as it was in our previous program, a border layout places the object in the center area.

Suppose we want to organize the colored areas in a grid to make a checkerboard. A border layout will not do. Fortunately, the package `java.awt` includes the class `GridLayout` for this purpose. When it's created, a grid layout is given a number of rows and columns. The areas of the cells in the resulting grid are the same size. The objects are placed in cells from left to right, starting with the first row and moving down. Our final program example resets the container's layout to a 2-by-2 grid layout and then places four panels colored white, black, gray, and white in it. The result is shown in Figure 2-14.

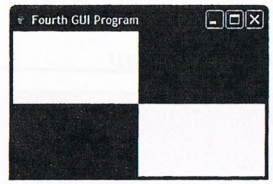
```
// Example 2.6: a frame with a 2 by 2 grid of colored panels

import javax.swing.*;     // For JFrame and JPanel
import java.awt.*;        // For Color, Container, and GridLayout

public class GUIWindow{

   public static void main(String[] args){
      JFrame theGUI = new JFrame();
      theGUI.setTitle("Fourth GUI Program");
      theGUI.setSize(300, 200);
      theGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel1 = new JPanel();
      panel1.setBackground(Color.white);
      JPanel panel2 = new JPanel();
      panel2.setBackground(Color.black);
      JPanel panel3 = new JPanel();
      panel3.setBackground(Color.gray);
      JPanel panel4 = new JPanel();
      panel4.setBackground(Color.white);
      Container pane = theGUI.getContentPane();
      pane.setLayout(new GridLayout(2, 2));
      pane.add(panel1);
      pane.add(panel2);
      pane.add(panel3);
      pane.add(panel4);
      theGUI.setVisible(true);
   }
}
```

FIGURE 2-14
A 2-by-2 grid layout with four panels



EXERCISE 2.7

1. Write the integer values of red, green, and blue for the following RGB colors:
   a. white
   b. black
   c. highest intensity blue
   d. medium gray
2. Describe the roles and responsibilities of a frame, a panel, and a layout manager in a GUI application.
3. Where are panels displayed when a border layout is used to control their placement in a window?
4. Write a code segment that would be used to set the layout for adding panels to a 5-by-5 grid in a window. You may assume that the panel's content pane is named pane.

SUMMARY

In this chapter, you learned:

- Java is the fastest growing programming language in the world. It is secure, robust, and portable. It is also similar to C++, the world's most popular programming language.
- The Java compiler translates Java into a pseudomachine language called Java byte code. Byte code can be run on any computer that has a Java virtual machine installed. The Java virtual machine (JVM) is a program that behaves like a computer—an interpreter.
- Java programs include variables, arithmetic expressions, statements, objects, messages, and methods.
- Three basic steps in the coding process are editing, compiling, and running a program using a Java development environment. Programmers should pay attention to a program's format to ensure readability.
- Java programs accomplish many tasks by sending messages to objects. Examples are sending text to the terminal window for output and receiving input data from the keyboard.
- There are several user interface styles, among them terminal based and graphical based.

VOCABULARY *Review*

Define the following terms:

Applet
Assignment operator
Byte code
DOS development environment
Graphical user interface (GUI)
Hacking
Integrated development environment (IDE)
Java virtual machine (JVM)
Just-in-time compilation (JIT)
Parameter
Source code
Statement
Terminal I/O interface
Variable

REVIEW *Questions*

WRITTEN QUESTIONS

Write a brief answer to each of the following questions.

1. List three reasons why Java is an important programming language.
2. What is byte code?
3. What is the JVM?
4. List two objects that are used for terminal input and output in Java programs.
5. Give examples of two compile-time errors.
6. What steps must be followed to run a Java program?

7. What is the purpose of an import statement in a Java program?

FILL IN THE BLANK

Complete the following sentences by writing the correct word or words in the blanks provided.

1. Two user interface styles are _____ and _____.
2. The message _____ is used to output data to the terminal window.
3. The message _____ is used to input a number from the keyboard.
4. A(n) _____ names a place where data can be stored in a Java program.
5. A(n) _____ stores the value of the expression in the variable.
6. Programs manipulate objects by sending them _____.

PROJECTS

Beginning with this chapter, we conclude each chapter with a set of programming problems and activities. We want to emphasize that programming is not just coding. Thus, a complete solution to each exercise in this section would include not just a set of .java and .class files for the program but also a report that covers the analysis, design, and results of testing the program. Ideally, you would do analysis and design before coding, and perhaps turn in this work for review before coding proceeds. How this is done depends on the size of the class and the time available to the instructor. In any case, when you see the words "write a program that . . .", you should at least pause to reflect on the nature of the problem before coding the solution. For example, your analysis might consist of a description of how the program would be used.

PROJECT 2-1

Write a program that displays your name, address, and telephone number.

PROJECT 2-2

A yield sign encloses the word YIELD within a triangle. Write a program that displays a yield sign. (Use stars to represent the sides of the triangle.)

PROJECT 2-3

Write a program that takes as input a number of kilometers and prints the corresponding number of nautical miles. You may rely on the following items of information:

- A kilometer represents 1/10,000 of the distance between the North Pole and the equator.
- There are 90 degrees, containing 60 minutes of arc each, between the North Pole and the equator.
- A nautical mile is 1 minute of an arc.

PROJECT 2-4

Write a program that calculates and prints the number of minutes in a year.

PROJECT 2-5

An object's momentum is its mass multiplied by its velocity. Write a program that expects an object's mass (in kilograms) and velocity (in meters per second) as inputs and prints its momentum.

PROJECT 2-6

National flags are displayed on various Web sites, such as http://flagspot.net/flags/. The flags of France, Mauritius, and Bulgaria consist of flat, colored areas. Write separate programs that display these flags.

PROJECT 2-7

Write a program that displays a 3-by-3 grid of black and white rectangles. The rectangles should be positioned so that no two rectangles of the same color are adjacent to each other.

CRITICAL *Thinking*

You have an idea for a program that will help the local pizza shop handle takeout orders. Your friend suggests an interview with the shop's owner to discuss her user requirements before you get started on the program. Explain why this is a good suggestion, and list the questions you would ask the owner to help you determine the user requirements.