1.6 Simple Input and Output

Java provides a rich set of classes and methods for performing input and output within a program. There are classes in Java for doing graphical user interface design, complete with pop-up windows and pull-down menus, as well as methods for the display and input of text and numbers. Java also provides methods for dealing with graphical objects, images, sounds, Web pages, and mouse events (such as clicks, mouse overs, and dragging). Moreover, many of these input and output methods can be used in either stand-alone programs or in applets.

Unfortunately, going into the details on how all of the methods work for constructing sophisticated graphical user interfaces is beyond the scope of this book. Still, for the sake of completeness, we describe how simple input and output can be done in Java in this section.

Simple input and output in Java occurs within the Java console window. Depending on the Java environment we are using, this window is either a special pop-up window that can be used for displaying and inputting text, or a window used to issue commands to the operating system (such windows are referred to as shell windows, command windows, or terminal windows).

Simple Output Methods

Java provides a built-in static object, called System.out, that performs output to the "standard output" device. Most operating system shells allow users to redirect standard output to files or even as input to other programs, but the default output is to the Java console window. The System.out object is an instance of the java.io.PrintStream class. This class defines methods for a buffered output stream, meaning that characters are put in a temporary location, called a ***buffer***, which is then emptied when the console window is ready to print characters.

Specifically, the java.io.PrintStream class provides the following methods for performing simple output (we use *baseType* here to refer to any of the possible base types):

print(String *s*): Print the string *s*.

print(Object *o*): Print the object *o* using its toString method.

print(*baseType b*): Print the base type value *b*.

println(String *s*): Print the string *s*, followed by the newline character.

println(Object *o*): Similar to print(*o*), followed by the newline character.

println(*baseType b*): Similar to print(*b*), followed by the newline character.

An Output Example

Consider, for example, the following code fragment:

```
System.out.print("Java values: ");
System.out.print(3.1416);
System.out.print(',');
System.out.print(15);
System.out.println(" (double,char,int).");
```

When executed, this fragment will output the following in the Java console window:

```
Java values: 3.1416,15 (double,char,int).
```

Simple Input Using the java.util.Scanner Class

Just as there is a special object for performing output to the Java console window, there is also a special object, called System.in, for performing input from the Java console window. Technically, the input is actually coming from the "standard input" device, which by default is the computer keyboard echoing its characters in the Java console. The System.in object is an object associated with the standard input device. A simple way of reading input with this object is to use it to create a Scanner object, using the expression

new Scanner(System.in)

The Scanner class has a number of convenient methods that read from the given input stream, one of which is demonstrated in the following program:

```
import java.util.Scanner;                    // loads Scanner definition for our use

public class InputExample {
  public static void main(String[ ] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your age in years: ");
    double age = input.nextDouble( );
    System.out.print("Enter your maximum heart rate: ");
    double rate = input.nextDouble( );
    double fb = (rate − age) ∗ 0.65;
    System.out.println("Your ideal fat-burning heart rate is " + fb);
  }
}
```

When executed, this program could produce the following on the Java console:

```
Enter your age in years: 21
Enter your maximum heart rate: 220
Your ideal fat-burning heart rate is 129.35
```

java.util.Scanner Methods

The Scanner class reads the input stream and divides it into ***tokens***, which are strings of characters separated by ***delimiters***. A delimiter is a special separating string, and the default delimiter is whitespace. That is, tokens are separated by strings of spaces, tabs, and newlines, by default. Tokens can either be read immediately as strings or a Scanner object can convert a token to a base type, if the token has the right form. Specifically, the Scanner class includes the following methods for dealing with tokens:

hasNext( ): Return **true** if there is another token in the input stream.

next( ): Return the next token string in the input stream; generate an error if there are no more tokens left.

hasNext*Type*( ): Return **true** if there is another token in the input stream and it can be interpreted as the corresponding base type, *Type*, where *Type* can be Boolean, Byte, Double, Float, Int, Long, or Short.

next*Type*( ): Return the next token in the input stream, returned as the base type corresponding to *Type*; generate an error if there are no more tokens left or if the next token cannot be interpreted as a base type corresponding to *Type*.

Additionally, Scanner objects can process input line by line, ignoring delimiters, and even look for patterns within lines while doing so. The methods for processing input in this way include the following:

hasNextLine( ): Returns **true** if the input stream has another line of text.

nextLine( ): Advances the input past the current line ending and returns the input that was skipped.

findInLine(String *s*): Attempts to find a string matching the (regular expression) pattern *s* in the current line. If the pattern is found, it is returned and the scanner advances to the first character after this match. If the pattern is not found, the scanner returns **null** and doesn't advance.

These methods can be used with those above, as in the following:

```
Scanner input = new Scanner(System.in);
System.out.print("Please enter an integer: ");
while (!input.hasNextInt( )) {
  input.nextLine( );
  System.out.print("Invalid integer; please enter an integer: ");
}
int i = input.nextInt( );
```