

## Keyboard Input

Our programs thus far generated data internally, operated on the data, and printed results. A more interesting approach is to engage the user to provide input data on the host system's keyboard. In this section we will show how to receive input from the keyboard and process that input within a Java program.

The most common user interface devices are the CRT display for output and the keyboard for input. Just as `System.out` was used to output data on the host system's CRT display, `System.in` is used to receive input from the keyboard. To do this, however, the following initialization is required:

```
BufferedReader stdin =  
    new BufferedReader(new InputStreamReader(System.in), 1);
```

This statement sets-up `System.in` as a buffered character *input stream*.<sup>1</sup> It's a bit messy, but there are two important services provided. The first is to convert raw bytes arriving from the keyboard into characters. This service is performed by the `InputStreamReader` class. Second, the characters are buffered to provide efficient reading from the input stream. This service is performed by the `BufferedReader` class. The statement declares and instantiates an object named `stdin` of the `BufferedReader` class. This is analogous to earlier statements that declared and initialized variables. Now, however, instead of a primitive data type, we have a class (`BufferedReader`) and instead of a variable, we have an object (`stdin`). So an object is "kind of like" a variable, and a class is "kind of like" a data type. Let's leave it at that for the moment and proceed with the job of inputting data from the keyboard.

Following the above statement, a line of text is read from the keyboard as follows:

```
String line = stdin.readLine();
```

Think of `stdin.readLine()` as an expression, which it is. As an expression, it is evaluated and it yields a value. This value is assigned to the variable `line`. More precisely, the `readLine()` method is called on the `stdin` object — the keyboard. A string is returned and assigned to the `String` variable `line`. The string contains the characters of a line of text entered on the keyboard.

A line of input ends when the user presses the Enter key. The Enter key generates an end-of-line code, but this is not included in the string returned by the `readLine()` method.

We'll meet strings more formally in the next section; however, for the moment, let's see what we can do with the string `line` returned by `readLine()`. Of course, the string can be printed:

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

If we want the user to enter a number on the keyboard, then we must once again confront the problem of converting one type to another. For example, if the user types 3, followed by 5, followed by the Enter key, then the `String` variable `line` contains the character '3' and the

---

<sup>1</sup> The argument "1" is used in the `BufferedReader` constructor to set the buffer size to one. According to the Java API documentation, this is not needed. However, a bug in the behaviour of Java implementation on *Windows 95* and *Windows 98* causes inconsistent results with keyboard input. The initialization shown here is suggested by Sun Microsystems to correct the problem.

character '5', as opposed to the integer 35. So before the value entered can be operated on as an integer, the string must be converted to an int.

Let's focus our discussion in the context of a simple demo program. Figure 1 shows the listing for a program that prompts the user to enter a name, age, and the radius of circle.

```
1 import java.io.*;
2
3 public class DemoKeyboardInput
4 {
5     public static void main(String[] args) throws IOException
6     {
7         // open keyboard for input (call it 'stdin')
8         BufferedReader stdin =
9             new BufferedReader(new InputStreamReader(System.in), 1);
10
11         // get input from the keyboard
12         System.out.print("Please enter your name: ");
13         String name = stdin.readLine();
14
15         System.out.print("Please enter your age: ");
16         String s1 = stdin.readLine();
17
18         System.out.print("Please enter the radius of a circle: ");
19         String s2 = stdin.readLine();
20
21         // perform conversions on input strings
22         int age = Integer.parseInt(s1); // string to int
23         double radius = Double.parseDouble(s2); // string to double
24
25         // operate on data
26         age++; // increment age
27         double area = Math.PI * radius * radius; // compute circle area
28
29         // output results
30         System.out.println("Hello " + name);
31         System.out.println("On your next birthday, you will be "
32             + age + " years old");
33         System.out.println("Area of circle is " + area);
34     }
35 }
```

Figure 1. DemoKeyboardInput.java

A sample dialogue with DemoKeyboardInput follows: (User input is underlined.)

```
PROMPT> java DemoKeyboardInput
Please enter your name: Andrew
Please enter your age: 4
Please enter the radius of a circle: 6.7
Hello Andrew
On your next birthday, you will be 5 years old
Area of circle is 141.02609421964584
```

In line 1, an **import** statement is necessary because the **BufferedReader** class and the **InputStreamReader** class are part the **java.io** package. The import statement informs the compiler of the location of these classes. We will discuss the import statement in more detail later.

In line 5, the `main()` method is defined with the `throws IOException` clause. This is required because certain types of errors, known as *exceptions*, are possible with keyboard input. A `throws` clause informs the compiler that we are aware of the possibility of such errors and that we will deal with them in an appropriate manner. Input/output exceptions are discussed in more detail later.

In lines 8-9, a link is made between `System.in` and an object named `stdin`. The effect is to setup the keyboard as an input stream for buffered character input, as discussed earlier.

With these preparations, we are ready to receive input from the keyboard. In lines 11-19, three prompts are output to the standard output, and after each the `readLine()` method is called on the `stdin` object to read a line of input from the keyboard buffer and assign it to a string variable. The first input is the user's name. It is returned as a string, and no further conversion is necessary.

The second input is the user's age. It is also input as a string; however, we wish to treat age as an integer and perform an operation on it. Since it is meaningless to perform arithmetic operations on strings, the `String` variable `s1` (line 16) must be converted to an integer. The conversion takes place in line 22 through the expression `Integer.parseInt(s1)`. This expression calls the `parseInt()` method of the `Integer` wrapper class to convert the string `s1` to an `int`. We'll say more about the `Integer` wrapper class later.

The third input is the radius of a circle. We want to treat this value as a `double` and perform operations with it. Once again, conversion is necessary. This occurs in line 23 using the expression `Double.parseDouble(s2)`. The conversion is very similar to that for an `int` except the `parseDouble()` method of the `Double` wrapper class is applied to convert the string `s2` to a `double`.

Operations are performed on the `int` variable `age` in line 26 and on the `double` variable `radius` in line 27. A reasonably precise constant representing  $\pi$  is available in Java's `Math` class using the expression `Math.PI`. As with all `double` values, it is accurate with about 15 digits of precision. The results are output in lines 30-33.

Note that the `+` operator, as it appears in lines 30-33, does not represent addition. When an argument on either side of the `+` operator is a `String` variable, the operation is *concatenation* — the joining of two strings. This is an example of *operator overloading*, or the ability in an operator to perform different tasks depending on the context. The concatenation operator is discussed in more detail in the next section on strings.

Another demo program using keyboard input is listed in Figure 2. The program `HeightConversion` prompts the user to enter a height in feet and inches. The output is the height in centimeters.

```

1  import java.io.*;
2
3  public class HeightConversion
4  {
5      private static final int INCHES_PER_FOOT = 12; // inches per foot
6      private static final double FACTOR = 2.54; // cm per inch
7
8      public static void main(String[] args) throws IOException
9      {
10         BufferedReader stdin =
11             new BufferedReader(new InputStreamReader(System.in), 1);
12
13         System.out.println("Please enter your height in feet and inches");
14
15         System.out.print("Feet: ");
16         int feet = Integer.parseInt(stdin.readLine());
17
18         System.out.print("Inches: ");
19         double inches = Double.parseDouble(stdin.readLine());
20
21         double cm = (feet * INCHES_PER_FOOT + inches) * FACTOR;
22
23         System.out.print("Height = " + cm + " cm");
24     }
25 }

```

Figure 2. HeightConversion.java

A sample dialogue follows:

```

PROMPT>java HeightConversion
Please enter your height in feet and inches
Feet: 5
Inches: 7.5
Height = 173.07 cm

```

This program includes a few new features. In line 16, the user's height is input and converted to an integer all in the same statement. The call to the `readLine()` method is simply an expression that returns a string.<sup>2</sup> Therefore, it is perfectly reasonable to embed this as an argument to the `parseInt()` method of the `Integer` wrapper class. In line 19, it is anticipated that the user might enter a real number, so `inches` is declared a `double` and the `parseDouble()` method is used for the conversion.

The conversion to centimeters is performed in line 21. Note the use of defined constants (lines 5-6) to make the algorithm more understandable. Integer promotion is taking place, so extra caution is warranted. The expression `feet * INCHES_PER_FOOT` is evaluated first; it involves two `int` variables so the result is an `int`. Then, this result is added to the `double` variable `inches`. The result is a `double`, and this is multiplied by the `double` variable

---

<sup>2</sup> Strictly speaking, it is incorrect to say that the `readLine()` method "returns a string". A string is an object in Java and the `readLine()` method, in fact, "returns a reference to a `String` object". Seasoned Java programmers accept the more terse "returns a string", and it appears frequently in Java textbooks and in the Java API documentation. Readers just learning the distinction between primitive data types and objects can be confused, however. And so, the point is made here to clarify the interpretation.

FACTOR to yield the final result, also a double. This is assigned to the double variable cm, which is printed in line 23.