STRINGS IN JAVA

Programs must often manipulate text, like words or sentences. Such text is composed of characters; a character might be are a letter, digit, punctuation mark, or space. Any particular sequence of characters is called a string.

Java includes the `String` class, each instance of which represents a string of characters. This class is one of the most heavily used in Java, and we study it in this chapter.

The String class

All Java classes require the use of **`new`** to create new instances of that class — with one exception. That exception is the `String` class, which is so useful that Java designers built special support for it into the language. To create a `String` object representing a sequence of characters, you need only to enclose that sequence inside a pair of double quotes.

```
String sentence = "Hello, world.";
```

In this code fragment, we create a variable named `sentence`, and we assign it to refer to the string `Hello, world.`

Sometimes you may want to include quotation marks inside a string. To do this, you can precede the quotation marks with a backslash.

```
String quote = "Frobozz said, \"Hello, sailor.\"";
```

If for some reason you wanted a backslash in a string, then you would precede it with a backslash.

There is another special exception with `String` objects, too: For other classes, you cannot apply operators like `*` or `>` to their objects; such operators can be applied only to primitive types like **`int`** and **`double`**. But you can apply the `+` operator using the `String` class; in this case, it combines values into a larger string. In the below example, we suppose that `name` is a `String` variable that has already been assigned to reference the user's name.

```
String greeting = "Hello, " + name + ".";
```

This fragment creates a new variable `greeting`. If `name` references the string `Dave`, `greeting` will be assigned to reference the string `Hello, Dave.` If `name` were `Dolly`, `greeting`would reference `Hello, Dolly.`

Strictly speaking, it isn't quite true that `String` is the only class that permits any operators to be applied to its objects: There are three operators that can be applied to objects of all classes: `==`, `!=`, and `=`. And later we'll see a special object type called an *array* that permits the bracket operator. But the `String` class is the only normal class that allows any *other* operators to be applied to its instances.

A program can add any type onto a `String` object, resulting in another `String` object. Suppose for example that `k` is an **`int`** variable holding the value 42.

```
String s = "k is " + k + ".";
```

The above fragment would assign `s` to be the string `k is 42.` That is, it places the two characters '4' and '2' into the string, since these two digits constitute the decimal expansion of the value referenced by `k`.

Sometimes this usage of `+` can lead to unusual behavior. What string do you suppose the following would compute?

```
String t = "k+1 is " + k+1 + ".";
```

We would hope that `t` would reference `k+1 is 43.` But in fact this fragment creates the string `k+1 is 421.` The Java compiler simply sees the usage of the `+` operator, and computes it left-to-right. So the computer first adds `k` to `k+1 is `, then it adds `1` onto that, and finally it adds a period. (The compiler doesn't pay attention to the spaces.)

String input and output

The `GraphicsProgram` class includes several methods that use the `String` class. Most accept a `String` as a parameter that is then displayed for the user to see.

`void print(String message)`

> Displays `message` to the user, leaving the cursor at the character following the last character of `message` (without advancing to the next line).

`void println(String message)`

> Displays `message` to the user and advances the cursor to the next line's beginning.

`double readDouble(String message)`

Displays `message` to the user, waits for the user to type a number, and then returns the number the user typed.

`int readInt(String message)`

Displays `message` to the user, waits for the user to type a integer, and then returns the integer the user typed.

`String readLine(String message)`

Displays `message` to the user, waits for the user to type a line of text, and then returns a `String` holding the characters typed by the user (omitting the final end-of-line character).

Note that `GraphicsWindow` also has `readInt` and `readDouble` methods that take no parameters, in addition to the above. Java allows multiple methods with the same name: When a program invokes a method for which multiple methods are named identically, the compiler will count the parameters and identify the parameter types, and use that to determine the appropriate method. (The compiler will not allow a class to define two methods with the same number of parameters, and with the corresponding parameter types all being identical.)

Figure 8.1 contains an example of a program using these methods. It is an enhanced version of the `MovingBall` program of Figure 6.3. In this version, the user first enters the ball's initial velocity. This version also informs the user once the ball has exited the screen.

Figure 8.1: The `CustomMovingBall` program.

```
import acm.program.*;
import acm.graphics.*;
import java.awt.*;

public class CustomMovingBall extends GraphicsProgram {
    public void run() {
        double xVelocity = readDouble("Horizontal velocity? ");
        double yVelocity = readDouble("Vertical velocity? ");

        GOval ball = new GOval(25, 25, 50, 50);
        ball.setFilled(true);
        ball.setFillColor(new Color(255, 0, 0));
        add(ball);

        println("Ball starts rolling.");
16          int frames = 0;
17          while(ball.getX() < getWidth() && ball.getX() > -ball.getWidth()
18                  && ball.getY() < getHeight() && ball.getY() > -
ball.getHeight()) {
19              pause(40);
20              frames++;
21              ball.move(xVelocity, yVelocity);
22          }
23          println("Ball exited window after " + frames + " frames.");
24      }
25  }
```

Notice how line 23 builds the string to be printed by adding an `int` variable's value onto the end of a string.

When executed, the program displays the following; the user's input is displayed in boldface. This dialogue will take place in a completely separate place from where the graphics are displayed. Exactly where the dialogue takes place depends on your computer's configuration, but the best bet is the window from where you started the program.

```
Horizontal velocity? 8
Vertical velocity? 6
Ball starts rolling.     output pauses while ball rolls across screen
Ball exited screen after 78 frames.
```

Source : http://www.toves.org/books/java/ch08-string/index.html