VARIABLES AND OBJECTS IN JAVA - II

Variable assignment

To specify the value to which a name refers, we use an assignment statement. An assignment statement looks like the following.

```
<variableName> = <valueToAssignIt>;
```

An assignment statement consists of two parts, separated by an equal sign ('='). The left side of the equal sign should identify the variable in question. And the right side should identify the object to which the variable should refer. Again, a semicolon must terminate the statement to enable the compiler to find the statement's end easily.

Line 6 uses an assignment statement.

```
yertle = new Turtle(50, 100);
```

On its right side, we create a new `Turtle` object on the right side, and on the left side we say that we want `yertle` to be a name for that turtle.

The use of an equal sign ('=') may lead you to believe that we are making a mathematical statement. But mathematics and Java use the equals sign in two very different ways. In mathematics, the equal sign *compares* two values for equality, without modifying anything. In Java, however, the assignment statement actually *changes* the variable to refer to a different value. We'll see how to compare values in Java later.

Note the order of the assignment statement: The variable to be changed goes on the *left* side. Beginners often want to swap the two sides, but this leads to trouble.

```
new Turtle(50, 100) = yertle; // Wrong!!
```

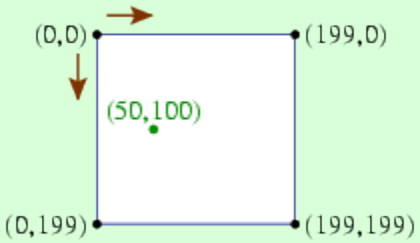
The impulse is natural: The computer determines the object first before assigning the variable to refer to it, and so it would make sense to write this in left-to-right order. Besides, in mathematics, equality is commutative. But assignment is not equality, so order is significant. And Java chose to have the variable name first. (It chooses this because the variable being assigned is the most important part of the statement, and so we start with that.)

The way you create an object in Java is a little peculiar, but it turns out to be convenient.
The `Turtle` class defines a constructor, which allows you as the user to create new objects of that

type. To use a constructor, you use the **new** keyword, followed by the class of object to be created, followed by a set of parentheses containing additional information about how the newly created object should start.

The `Turtle` class's constructor specifies that when creating a turtle, you need to specify the coordinates where the turtle will start. The turtle will automatically start out facing east. The turtle's drawing area by default is 200×200 pixels, with (0, 0) being at the upper left corner and (199, 199) being at the lower right corner, as illustrated in Figure 2.3.

Figure 2.3: The turtles' coordinate system.



In our program, we start the turtle at (50, 100), which is 25% of the way across the drawing area and halfway down. Yes, the turtle's coordinate system is upside-down relative to the traditional mathematical coordinate system, but placing $y = 0$ at the top of the screen is the norm in computing.

Once assigned, you can feel free to reassign the variable. In effect, this tells the computer to change its mind about the object to which that variable refers. Once reassigned, the computer will completely forget that the variable ever referred to the first value. The `DrawEquals` program of Figure 2.4 illustrates such a program. The *first* assignment to a variable is called that variable's initialization. If a program tries to refer to a variable without initializing it, then the compiler will complain.

Figure 2.4: The `DrawEquals` program body.

```
Turtle yertle;
yertle = new Turtle(50, 80);  // draw top line
yertle.forward(100);
yertle.hide();
yertle = new Turtle(50, 120); // draw bottom line
yertle.forward(100);
yertle.hide();
```

Incidentally, Java allows a single statement combining the variable declaration and the initialization of that variable. We can combine lines 5 and 6 thus.

```
Turtle yertle = new Turtle(50, 100);
```

This handy space-saver is certainly convenient, and professional programmers use it all the time. You're welcome to use it, too. But the next few chapters will continue to separate this into two separate statements, to emphasize the important fact that variable declaration and variable initialization are two completely separate actions.

Instance methods

The `Turtle` class as defined in the `turtles` package defines several behaviors that a turtle can perform. Each of these behaviors is called an instance method. *Instance* is a synonym for *object* in Java. The behavior called an *instance method*, or sometimes just a *method*, since it's something that a particular instance (object) knows the the procedure for performing.

Sometimes, when we tell an object to perform a method, we will give additional information about exactly what it should do. As an analogy from real life, Jeeves might know how to fetch things, but I would never say simply, Jeeves, fetch. Instead I would tell Jeeves exactly what to grasp: Jeeves, fetch the mail. We would say that Jeeves' *fetch*method requires some additional information. Each such piece of additional information is called a parameter.

Lines 7 and 8 of our program tell our turtle to do things for us.

```
yertle.forward(100);
yertle.hide();
```

To tell an object to do something, you give the name of the object first, followed by a period, followed by the name of the instance method you want it to perform. (You can think of the period in Java corresponding to the comma in an English imperative sentence such as Jeeves, fetch the mail.) After the instance method name is a set of parentheses containing any parameters.

In line 7, we tell the turtle named by `yertle` to perform its `forward` method; the 100 in parentheses is a parameter telling the turtle how far it should go forward. In line 8, we tell the turtle to hide. Notice that even though the `hide` method requires no parameters, we still have to include parentheses.

Each class defines a set of instance methods, specifying exactly what behaviors are defined for objects. There's nothing particularly sacred about the word `forward` or `hide` — they're just identifiers chosen by the person who wrote the `Turtle` class. Other classes will have other instance methods, with different names.

Source : http://www.toves.org/books/java/ch02-obj/index.html