Programming via Java-Variables and objects

Writing programs in Java requires creating and using objects, which we will think of as entities that do things for us. Of course, the computer doesn't actually have inside it miniature factories creating physical objects. But thinking in these terms turns out to be a useful conception of programming, so we'll talk as if that is indeed what is happening. Exactly how the computer provides this factory illusion is a topic for another book.

Each object has a particular type, called its class. You can think of the *class* as being the factory, and the *object* as being a product produced by the factory. The object's class determines what the object can do. In this chapter, we'll be working with turtles, so our objects will be members of a class named `Turtle`.

Since most programs will manipulate a variety of objects, a program must have some way to distinguish them. To permit this, the program will associate a name with each object. In programming, we refer to these names as variables.

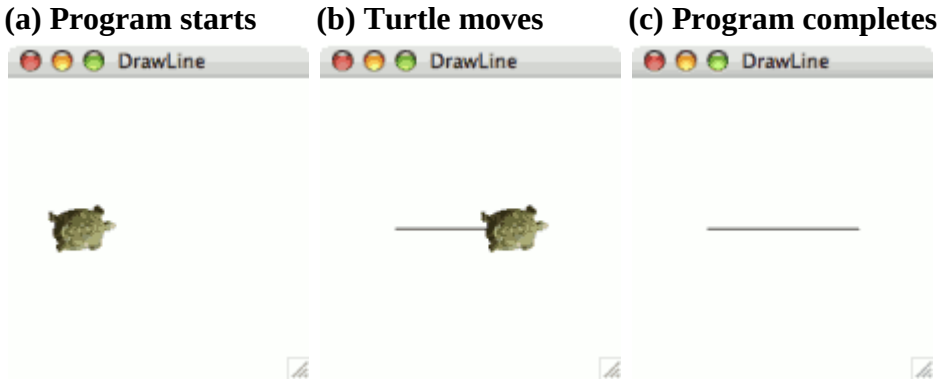
2.1. The `DrawLine` program

We'll begin with a slight extension of `BaseProgram` called `DrawLine`, found in Figure 2.1. As you might guess by the name, this program manages to draw a line, as Figure 2.2 illustrates. As the program executes, it demonstrates an animation of a turtle tracing out a line.

Figure 2.1: The `DrawLine` program.

```
1  import turtles.*;
2
3  public class DrawLine extends TurtleProgram {
4      public void run() {
5          Turtle yertle;
6          yertle = new Turtle(50, 100);
7          yertle.forward(100);
8          yertle.hide();
9      }
10 }
```

Figure 2.2: Running `DrawLine`.



As promised, the outside bits of the program (lines 1 to 4, and 9 to the end) are precisely as in `BaseProgram`, except for changing line 3 to say `DrawLine` instead of`BaseProgram`. You still don't need to understand those lines very well.

By the end of this chapter, though, you should understand the middle part well (lines 5 to 8). This portion tells the computer exactly what it is supposed to do when running the program. It is separated into statements, each being a single piece of the program telling the computer to do one particular thing. Most statements in Java end with a semicolon, as you see here, just as most statements in English end with a period. You'll see that we follow the convention of placing each statement on its own line. The computer doesn't itself pay attention to this, but using separate lines helps to make the program easy to read.

2.2. Variable declaration

We'll start at the first statement and proceed down, just as the computer does when it executes the program. The first statement is on line 5, which tells the computer that our program uses a variable.

```
Turtle yertle;
```

As already mentioned, a variable is an object's name. In Java, each variable has an associated type, which indicates what sort of object the variable will be naming. Before saying what object the variable names, Java requires that the program create the variable using a variable declaration. The variable declaration warns the computer that the variable exists and notifies the computer the type for that variable. The Java syntax for declaring a variable is to give the type first, followed by the variable's name, followed by a semicolon.

```
<typeName> <variableName>;
```

In our program, we want to warn the computer that our program uses a variable named *yertle*, which will refer to a turtle. Line 5 accomplishes this.

As far as the computer is concerned, what name you choose for the variable isn't important, as long as you're consistent. For example, we might have chosen to name our turtle *mack*. In that case, the core of our program would look like this.

```
Turtle mack;
mack = new Turtle(50, 100);
mack.forward(100);
mack.hide();
```

Name your variables with care. It is much easier to write and understand a program when its variables' names indicate their purpose. The name *yertle* wasn't a good example for this.

In Java, you can choose virtually anything you want as a name. Java requires that the name contain only letters, digits, and underscore characters ('_'). And it can't begin with a digit. Java reserves a few dozen words for special uses (like **`import`** and **`class`**). But other than that, any name is good. Letter case is significant, so for example `yertle` and `Yertle` are different variables.

Java programmers generally follow the convention that variable names (like `yertle`) start with lower-case letters, while class names (like `Turtle` or, for that matter, `DrawLine`) start with capital letters. This helps alleviate confusion. When the name incorporates multiple words, the convention is to use no spaces and capitalize the second and following words, such as `teenageMutantNinja`, for example. (We shall speak of ninjas no more.)

So line 5 creates the name `yertle` for a `Turtle` object. This does *not* mean that `yertle` is a name for a `Turtle` yet. It's just a name right now that can potentially be a name for a particular `Turtle` object, just as you understand *Ben* is a man's name, but you won't understand who I mean by it until I point him out to you. Internally, the computer allocates some memory for remembering to what object `yertle` refers. But it hasn't been told to what it refers, so for the moment the computer will remember the variable `yertle` as referring to nothing.

2.3. 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!!
```

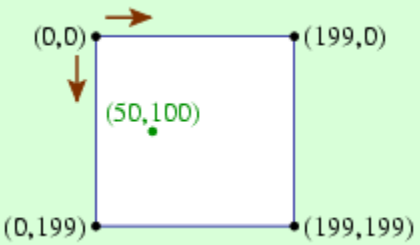
The impulse is natural: The computer determines the object first before assigning the variable to refer to it, and so it would make sense to write this in left-to-right order. Besides, in mathematics, equality is commutative. But assignment is not equality, so order is significant. And Java chose to 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.

2.4. 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