# VARIABLES AND OBJECTS IN JAVA

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.

## The `DrawLine` program

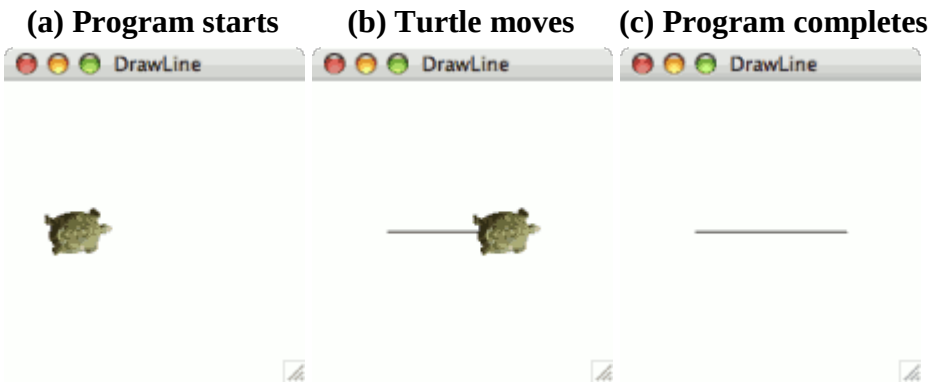
We'll begin with a slight extension of `BaseProgram` called `DrawLine`, found in Figure 2.1. As you might guess by the name, this program manages to draw a line, as Figure 2.2illustrates. As the program executes, it demonstrates an animation of a turtle tracing out a line.

**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`.

**(a) Program starts** **(b) Turtle moves** **(c) Program completes**




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.

# 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.

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