Classes

Java is what you call an object-oriented programming language. This means that almost everything in Java is grouped together into groupings called objects. You can think of many objects representing an object in the real world. For the code on a robot, you may have an object that represents the motor, a sensor, or a camera. Each one would have a state (variables) and behaviors that it could perform (functions). So a motor might have variables like whether it is on or off, the current speed, and it may have functions like turning it on and off.

CLASS VS OBJECT

So then what is a class? A class is a blueprint for an object. It lays out a description of how each object is going to behave. Then each object is an instantiation of that blueprint. So our description that a motor has a certain set of variables and functions would be in our class and then each motor that would have its own state would be an object.

CREATING A CLASS

Let's imagine we are creating a drawing program of some sort and we want to represent the shapes we can draw in classes. Let's start with a rectangle. We'll want some representation of the location, size, color. So let's look at some basic code:

```
public class Rectangle {
	public int length;
	public int width;
	public int x;
	public int y;
	String color;

	public int getArea() {
		return length * width;
	}
}
```

Wait a second! This is starting to look just like all the other code you've written before…right? That's because it is just like every other file. Since Java is object-oriented, that means that every file you've written *is* a class.

So, here's the main method? We don't have one. We've been using one before because we've had all of our code in one file. Since this will just act as a supporting file, we don't need a main method here.

CREATING A VARIABLE

So what you can do with your class is create a new variable with that type.

In a different file (Main.java), inside the main() function, we put this line

```
Rectangle r = new Rectangle();
```

This looks a little bit like creating a new variable of other kinds. The type is Rectangle, the name of the variable is r. Then the value we're assigning to that variable is what is different. We have a special

keyword new and then we are calling a function called a constructor. It will construct the object. It will create all of the default values for all the variables that we need. This is a special type of function. In our original class, we didn't write one, so let's add it in and see what it might look like to add one.

```
public class Rectangle {
     public int length;
     public int width;
     public int x;
     public int y;
     String color;
     public Rectangle() {
          length = 0;
          width = 0;
          x = 0;
          y = 0;
          color = "black";
     }

     public int getArea() {
          return length * width;
     }
}
```

Let's look at a few things. Notice that this is written almost like any other function, except for a few things. First, it doesn't have a return type. Secondly, the name will always match the name of your class. It will only ever get called when the object is created, so it will consist of code that sets up the object with initial values. Here with no extra information, we just set the length, width, x and y to 0 and the color to black.

We can even add a second constructor that has parameters. This second constructor imagines a situation where someone clicks on a point with a particular color to start creating a rectangle. So now we know x, y and the color, but the length and width are still 0.

```
public class Rectangle {
     public int length;
     public int width;
     public int x;
     public int y;
     String color;
     public Rectangle() {
          length = 0;
          width = 0;
          x = 0;
          y = 0;
          color = “black”;
     }
     public Rectangle(int x, int y, String color) {
          length = 0;
          width = 0;
          this.x = x;
          this.y = y;
          this.color = color;
     }

     public int getArea() {
          return length * width;
     }
```

```
}
```

When we call it, it will look something like this

```
Rectangle r = new Rectangle(3, 10, “red”);
```

Some things to look at here. We did a lot of what looks like duplicate naming here, but it is all still allowed.

First of all, we have two constructors. They have the same name. They have to. They are both constructors, but since they have a different set of parameters (one has none and one has int, int, String) everything is fine. But if I wanted to add one that just did length, width, and color that wouldn’t be allowed because that is also int, int, String and if I called it like above, you wouldn’t know if I mean it was length 3 and with 10 or x=3 and y=10. So as long as the parameter sets are different, we’re fine.

Next is the parameter names match the global variable names. That is perfectly fine because one is local to the function, the other is global to the file. The thing is that if I just use x, y or color, I will always be referring to the local variable. So in order to refer to the global variable I put “this.” in front of it. So those last three lines of the constructor just make the global variables match the parameters you sent in.

USING YOUR VARIABLE

So how do you use this new variable that you have? In order to access any of the things you’ve just created inside your new class, you’ll have to use the . operator.

So if you wanted to create a rectangle and print out its x, y and area, here’s how you would do it.

```
Rectangle r = new Rectangle(3, 10, “red”);
System.out.println(r.x);
System.out.println(r.y);
System.out.println(r.getArea());
```

We can also change values the same way. So these lines also work

```
r.length = 4;
r.width = 7;
```

```
int x = r.length + r.width;
```

You can use the variables/functions within this class the same way you would any other variable. The reason we're able to do that here is because we made all of our functions and variables public. Public here means that they are available outside of the file. We can also mark them private, which means that we cannot access them directly outside the file.

At first it may seem strange to mark variables or functions private but the use of things marked private can be useful for things that have very specific rules around how they can be changed. If one variable always has to change along with another variable, you may want to mark them both private and have them change only in a function. If a variable value needs validation of some kind, it may also make sense to have it be private. Many times, functions should be private if they were just created for internal use and have no meaning outside that file.

Let's think about a motor class. It may have a constructor that has a parameter for where the motor is plugged in. That number should probably never change over the lifetime of the motor, and so it would probably be in a private variable, so I can't go in and change it later. The motor may have some track of what speed it is going, but when I set the speed I want it to go to, there is a lot more that needs to happen than just changing a variable, so any internal tracking of speed is probably a private variable and there is probably a public setSpeed() function that does the actual electrical changes that help change the speed.

COMMON FUNCTIONS IN CLASSES

While they aren't required, there are several function types that are common when building a class.

Setters

Setters are functions that set a particular variable. These are used when a variable is private. In its simplest form a setter looks like this:

```
public void setLength(int length) {
     this.length = length;
}
```

It takes in a value, and sets a global variable to that value. Since this is just a function, it can do anything else you want, but you commonly see things like validation. In this case you might want to check that the length is not negative or maybe even not too large.

Getters

These are the counterpart to setters and are a way to give access to the value of a variable without giving edit access. In their simplest form a getter looks like this:

```
public int getLength() {
     return length;
}
```

Because of its name, the getArea() function in our original Rectangle class could be considered a getter, even though there isn't really an area variable to return, particularly since it is quite simple code that it is running.

equals

You may want to see if two of your object are the same object. In this case you would need an equals() function.

```
public boolean equals(Rectangle other) {
     return this.x == other.x && this.y == other.y && this.width ==
other.width && this.length == length &&
this.color.equals(other.color);
}
```

We use "this." to make it clearer which is this and which is other in this case. You may also see here that we're using .equals() from String, and the syntax may look pretty similar. That's because unlike the other basic variable types we used before, Strings are actually class. So you've been using objects all along.

toString

The final common function type that we are going to discuss is toString(). Often if you want to know more about a bug, you print out information to a console or the screen, so it can be useful to have a debug string that prints the useful information from your class. This may include key values while leaving out ones that are irrelevant and may include some simple formatting.

```
public String toString() {

     return “Rectangle @ (“ + x + “, ” + y + “) ” + length + “x” +
width;

}
```

This would give something in the format “Rectangle @ (3, 10) 5x5”, leaving out color.

PACKAGES

Classes can be sorted into packages. Packages are sort of like folders. At the top of a file, you put the name of the package your class is in, generally a dot-separated organization name of some sort. And of course packages can have sub-packages.

So the top line of your file should be something like:

```
package packagename.subpackagename;
```

Then when you want to use that class, you need to tell the compiler where in the package structure to find any class you are using by importing it. An import statement would be something like:

```
import packagename.subpackagename.ClassName;
```

ENUMS

Sometimes you want to build a value that is just one of a set of states. This could be a set of positions for an arm of a robot, or a set of valid states for a controller or a finite state machine. Now they could be represented as the numbers 1, 2, 3, 4 or something like that, but that could be difficult to read the code. And what happens if only the values 1-4 have meaning and someone sets that value to 5, 0 or 104?

This is where an enum is useful. Enum is short for enumeration. It lets you fully list and name every possible value you have.

Let’s go back to the Rectangle example. We may only have a certain number of valid colors in our drawing program so if we want to specify those we may use an enum. This is how you create one.

```
public enum Color {

     RED, GREEN, BLUE, BLACK, ORANGE
```

```
}
```

Now if we want to use it. We can say something like

```
Color c = Color.RED;
```

WHY MAKE A CLASS

Classes are often made for the same reason that functions are. It allows for code reuse. It gives code a name that makes things clearer. It also allows for code to be created by one person and used by another. In some cases, it is done via a library. Think about a case where one person creates a physical device and writes some code for it, like a motor. Someone can create a Motor class and then the person who is using the motor doesn't have to worry about the exact electrical stuff and just publish and API (application programming interface), basically a list of public variables and functions and what they do.

TRY IT OUT!

1. Create a class that represents an individual card from a deck of cards. You can represent the internal parts of it however you want, but your class should have a constructor, an equals method and a toString method.
2. Based on your card class, create a deck class. It should have at least a no-argument constructor which creates all cards that are part of a normal deck of cards.
3. Build a simple class that create a deck of cards, and deals 5 cards to each of 5 players.
4. Try something more complicated with your deck of cards. Shuffle. Build a function to compare two cards. Build the game war (first try without ties, then build ties).