# Classes, Types, and Typing

## 1 Classes and Objects

Java is an *Object-Orientated Language* in which every file must a *class* or *interface*. Functions within these files is called *methods*. Classes in Java are powerful because we can create a skeleton for many different objects instead of repeating ourselves. Let us take an example of 2 wildcats Jeremy and Josephine.These two wildcats have all the exact same characteristics except their name is different. The naive approach to representing Jeremy and Josephine would be to make a separate class for each one. However, Java lets us do something much better, we can create a new *instance* of the class instead of just rewriting it. Here is an example of how to instantiate 2 objects.

```
1 Wildcat jeremy = new Wildcat("Jeremy");
2 Wildcat josephine = new Wildcat("Josephine");
```

We now know that we can instantiate classes; however, we can have various types of data and methods inside of our classes. Below is a list of what we can store inside of classes

- **Instance Variables:** These are variables that are different for each instance of a class.
- **Static Variables:** Static Variables are variables that are common to all instances of a class. This means that if the static variables changes value in one instance of the class, it will change in all in all. instances.
- **Instance Method:** In order for an instance method to be called, you must call it from an instance of the object.
- **Static Method:** Static methods can be called directly through the class
- **Constructor:** The constructor is a "method" that is used in order to instantiate a class. Whenever you instantiate a class, the arguments that are passed in are what get passed into the constructor.
  - ***Important note: Static Methods cannot have any references to instance variables or methods.***

  ***However, instance methods can reference static variables and methods. More on this in the next lesson.***

Below is an example class, see if you can identify all its components.

```
1  public class Human{
2        public int weight;
3        public static int joy = 100;
4        public Human(int mass){
5            weight = mass;
6        }
7        public void eat(){
8              weight +=5;
9              System.out.println("I love eating");
10       }
11       public static void play(){
12             System.out.println("I have" + joy + "happiness ");
13       }
14 }
```

*Weight- Instance Variable, Joy- Static Variable, public Human- constructor, eat- instance method, play- static method.*

## 2 Static Typing vs Dynamic Typing

As we foreshadowed earlier, there is a difference between when static methods can be used as opposed to Instance (dynamic) methods. Let us take the example of the human class from 1.1. This is a possible series of calls:

```
1  Human.play();
2  Human kartik = new Human(5);
3  kartik.eat();
4  kartik.play();
```

Note that before we called kartik.eat(), we had to instantiate kartik to be a human. Another important note to make is that we can still call play on kartik even if play is static. If we attempted to call *Human.eat()* we would end up with a compilation error.
So far, we have not had any difficult problems. Rest assured, in a later lesson, this concept will become much trickier. Make sure that you understand the difference between Static and Dynamic typing for now.

## 3 Primitives and Reference Types

In Java, a variable can be either a *Primitive* or a *Reference type*. There is a concept of bits, which is not discussed too much in 61B, but the main concept is that bits are a unit of measurement to see the amount of space that something takes up. Whenever you declare a variable, of either type, Java sets aside some memory for it.

**The Golden Rule of Equals** is given two variables x and y, if we set y equal to x, all the bits from x are copied to y.

```
1  y = x;
```

In Java there are 8 primitive types, below I have listed the types, a brief description, and how many bits they use.

- **byte** - a unit of data that is 8 binary digits long. 8 bits.
- **short** - An integer from -32,768 to 32,767. 16 bits
- **int** - An integer from -2,147,483,647 to 2,147,483,648. 32 bits.
- **long** - An integer from -9,223,372,036,854,775,807 to 9,223,372,036,854,775,808. 64 bits.
- **float** - From 3.402,823,5 E+38 to 1.4 E-45. 32 bits.
- **double** - A decimal number. 64 bits.
- **boolean** - True or False value. 1 bit.
- **char** - A sole character. 16 bits.

If something in Java is not a primitive then it must be a reference type, or the instance of an Object. Whenever you instantiate an object, Java stores it's **address** in 64 bits. This has an incredibly important consequence. If we take our prior example where we set y equal to x, copying the bits of a reference type would not result in creating a duplicate object like it would for primitives, rather we would just copy over the address to the object. ***This is a very important distinction to make.***