CS61B Note 1: Fall 2019

Kevin Moy

Abstract

Welcome to 61B! The course aims to teach you about the many, many wonderful properties about various data structures. In this introductory note, we'll go over a brief overview of the most important parts of object-oriented Java, as it is the language used to teach this course and build our representation of data structures from.

1 Programming

In order to make programs, such as the one you're using to read this note, we need to be able to code. In today's technology-overridden world, the demand for proficient coding is higher than ever.

Let's get straight into it. This note assumes you've taken CS61A thus gained decent familiarity with the Python language, and thus understand core principles of programming such as primitives and basic code structure.

1.1 Baby Program

You might recognize this small snippet of Python from CS61A, (or earlier):

```
print("Hello World")
```

That's really all a Python program needs to print some text. Nice and simple.

Unfortunately, Java isn't as straightforward or abstract. Here's the Java version of Hello World.

```
/** Javadoc CLASS Documentation Comment. [Class Description Here]
* @author Kevin Moy */
public class Main {
   /** Main method is where all the code starts. */
   public static void main(String[] args) {
      System.out.println("Hello world");
   }
}
```

We see that we need **methods/functions** to do actions, and **classes** to place those methods/functions in. However, this is a bad example because classes and functions really serve a different purpose: the point of classes is to serve as a "blueprint" to create **objects**. Methods are usually **called** from those objects. The main method you see above is the exception: It isn't an object method, but a starting point for the code. You'll never *see* the main method called in code.

Try this program! If you don't know how to run a Java program, it's sort of an annoying process. Read this nice tutorial to get it going.

2 Classes

ALL code has to be in a class in Java. Here's a more practical (and common) example of a class.

```
/** Athlete class. **/
* @author Kevin Moy */
public class Athlete {
   /** Athlete object CONSTRUCTOR. */
   public Athlete() {
      _strength = 1000;
      _speed = 1000;
   }
   /** Athlete object METHOD.**/
   public void getInjured(int recoveryTime) {
      _strength = 0;
      _speed = 0;
      _daysOut = recoveryTime;
   }

   /** STRENGTH attribute.**/
   public int _strength;
   /** SPEED attribute. **/
   public int _speed;
   /** SPEED attribute. **/
   private days _daysOut;
}
```

This small code snippet contains the three most important parts of any class: its constructor, its fields or attributes, and its methods.

2.1 Attributes

The **attributes** of a class just represent an object's characteristics or some meaningful information values about it. We see that in the Athlete class, there are three attributes: `_strength`, `_speed`, and `_daysOut`. Each of those attributes

gives some numerical information about an Athlete. However, attributes can be any type: a primitive (int), a Java library type (ArrayList) or even a type we define ourselves (Athlete).

2.2 Constructor

The **constructor** can be thought of as the "object maker". Let's make an `Athlete` object.

```
Athlete lebronJames = new Athlete();
```

What happens when with this line of code? Memory is allocated for an Athlete object, a pointer referred as "lebronJames" points to this object, and Lebron (the object) is given attributes of 1000 strength and 1000 speed (quite fittingly, I must say). Most importantly, we now have an Athlete object we can play around with. What do I mean by that?

2.3 Methods

Methods are really toys. They are, in reality, are simply a special type of function: it's a function that only works when an object [or class] calls it. Let's take a closer look at our tasteful `getInjured` method.

```
public void getInjured(int recoveryTime) {
   _strength = 0;
   _speed = 0;
   _daysOut = recoveryTime;
}
```

The function header, or sometimes called the function *signature*, consists of the return type and the parameter type(s). `void getInjured(int recoveryTime)` makes up our function header. Note that unlike Python, Java forces us to know a little more about what our method/function is trying to do, and what inputs it will take.

Now a little on how to call these methods. There essentially only two ways you'll ever call methods, and both also use **dot notation**. Remember that methods are always specific to a SINGLE class or type. Thus:

1. Create an object and call the method using `object.method([params])`. For example, to injure Lebron, you would call:

   ```
   lebronJames.getInjured();
   ```

   We can see that the Lebron's object attributes- often called **instance variables**, since they belong to the `lebronJames` instance - are changed; indeed, this is the practical use of methods.

2. Use the class name and call the method using `object.method([params])`. An example that should look familiar:

```
System.out.println("RIP XXX");
```

   Here, the `System` class finds its attribute `out` (which is another class), which then calls its method `println`. Note that there was no object creation with the `new` keyword. Methods called in this way have can perform a large variety of functions. However, be careful with calling these: you might find yourself changing something important that screws up your whole program.

Looking at our special `main` method, we see that it takes in a String array parameter type named `args`. Remember what I said before about the main method never being called? Well, it is called- called automatically when you run a Java program with `java JavaProgramName` on the command line. However, you can also give any number of arguments afterwards, separated by spaces– those will become items in the args array.

For example, calling

```
java JavaProgramName a b c
```

will give populate our string array `args` in the `JavaProgramName` class as follows:

```
args = ["a", "b", "c"]
```

This is going to be a key part of the Gitlet project (and probably others)- don't forget it.

3 Access

What do classes, attributes, and methods/functions have in common? They all have an **access modifier**. This is the very first word that comes before declaring those things.

```
public void getInjured(int recoveryTime) {
   _strength = 0;
   _speed = 0;
   _daysOut = recoveryTime;
}
public int _strength;
public int _speed;
private days _daysOut;
```

You can see access modifiers `public` and `private` here (`protected` also exists, but I highly doubt you'll need it for this class).

- `public` means that the class/attribute/function can be used anywhere else in the program- even from other classes (in the same package).
- `private` means that the class/attribute/function can only be used in the class where it was declared- other classes may NOT access the private thing.

If a class is named public, then you can create objects of that class in other classes. However, if any of its entities are named private, you can't access them through that class except through getter or setter methods, which runs code in the class we're trying to access stuff from anyway (and thus the private modifier no longer matters).

4 Static

Having some idea of what access is, we can touch upon **class attributes**. These attributes are **specific to the class** (NOT instances), and are specified by the `static` keyword.

It's fairly straightforward thinking of an example. For our `Athlete` class, we just ask ourselves, "what is one thing that is gonna be shared by all athletes"?

```
public static boolean hasBrain = True;
```

Kind of a cop out, but all athletes do have a brain. A more practical and useful example of a class variable would be a count of something, like the number of times a certain method is called or the number of times a certain attribute is changed for all objects.

What's important to know about static things is that they are ALWAYS second to instance and local variables. What that means is that if I have the code below:

```
public class A {
   public static int B = 3; //Class variable/attribute
   public int B; //Instance variable/attribute
   public void printB() {
      System.out.println(B);
   }
      public static void printBStatic(){
         System.out.println(A.B);
      }
   }
```

Then calling the non static printB method would look for the instance variable B first. If it doesn't find it, then it looks for the class variable version.

However, we can also call the class (static) method printB by using the class-name: `class.staticMethod`, or `A.printBStatic()` in our above example. This would print our class variable B, which is pre-set to 3.

Exactly the same as calling methods, in order to access any attributes, we simply use dot notation: `class.classAttribute`, or `A.B` in the example above. Or we could create an object with type `A` named and call an instance method. Note that if I create 500 `A` objects all of them would print 3 when calling `printBStatic` as an instance method. However, each object also gets their own version of the `B` attribute that can be set via constructor or setter method.

5 Styling

In this class, a significant portion of project points comes from ensuring your code adheres to 61B styling conventions. I've listed some rules below that I've had to fix the most for myself:

- Remove all "//" comments (NOT javadoc comments).
- No lines longer than 80 characters, and no methods with over 60 lines of code.
- Space before each "{".
- Space before and after operators.

```
System.out.println(1 + 2);
int z = 3;
```

- Give an official javadoc comment to each class AND its associated fields and functions. Make sure the first sentence in your comment ends with a period. Specify @param and @return for methods in these comments if they take meaningful parameters or return a non-void type.
- camelCase for variables and function names- no underscores.
- if-else blocks should be formatted like:

```
if (...) {
   doSth();
} else if (...) {
   doSthElse();
} else {
   idfk();
}
```

The full style guide can be found here.

6 Summary

In this "introductory" note, we've covered:

- the most basic intro Java program.
- Classes, their purpose and the most important parts about them (attributes, methods, constructor).
- Styling requirements for project code submission in this class (note: styling isn't required on the HW or labs, but it's a good idea to practice anyway).
- The meaning of access in java.