Java Fundamentals

The “Oh God I haven’t learned anything yet” supplement.

Eoghan Hayes

Objects

- A class is a blueprint of an object. What does this mean?

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

- This is the “blueprint” for hello world.

- Is this an Object Oriented approach?
  No.
- Would it be useful to make an Object Oriented Hello World?
  No.

Objects

But let’s do it anyway.

Objects; and why to not use Static.

- Firstly, let's have a look over the word "Static".

- "Static" is a keyword for variables that must remain constant throughout all instances. In other words: If we created 20 objects that just stored a number in a field called "static int number" , changing that value in any of our objects, would change it for all of them.

- We'll do an example of that later – but for now, the most important thing to think, is "When would I ever want to do that". The answer is "Incredibly rarely".

- What does this have to do with Objects?
  We can finally get rid of appending everything with "Static".

Objects; and why to not use Static.

- Any main method you write, MUST be appended with “Static”. Why? Mostly, convention; and Static methods can be called without creating a new instance of an object. You can edit the Java compiler so that it no longer needs the main method to be static – but that might be a bit beyond us right now.

- As you should know by now however, if you try to access, or change a variable in your main method – the variable must also be static, otherwise the Java compiler will cry at you. The same goes for calling other methods. This leads to a mess of “Static”, dumped all over your code.

- It’s fine not to understand exactly what this means right now – all you need take from this, is that if your code has “Static” all over it, it’s very likely that you’re not using Static correctly. Using Objects is how we can get out of this practice.

- Static = Bad , Objects = Good.

Objects; the Constructor.

- So if we want to start using Objects, we have to understand what a Constructor is.

- You can consider the Constructor to be your “main” method for instantiating (creating) Objects. Whenever you create a new Object, the Constructor is automatically called.

- The Constructor is ALWAYS, the name of the class – and appended with the “public” keyword. So for example, our HelloWorld class, now with a Constructor;

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }

    public HelloWorld() {
        System.out.println("This is the Constructor.");
    }

}
```

Objects; the Constructor.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }

    public HelloWorld() {
        System.out.println("This is the Constructor.");
    }

}
```

- But what happens now when we try to compile and run this program? Does the Constructor get run? Or the main method?

  Just the main method. The Constructor will ONLY be called, if we create a new Object based on this class. So, let's make an Object, based off the HelloWorld "blueprint".

Objects; Actually creating an Object.

- So now that we roughly understand what a Constructor is, and have engrained that for now, “Static = BAD” – we’ll take our Hello World source code (as we have it so far), and create an Object based on it.

- The format for creating an Object based off a class, is as follows;

  ClassName any-name-we-want = new ClassName(any required arguments);

- So, for HelloWorld.class, we would create a new Object of it with;

  HelloWorld myObjectName = new HelloWorld();

  Note: We have no arguments.

- Now, we can interact with this object’s variables and methods.

Objects; Public, and private variables

- In Object Oriented code, the “Public” and “Private” keywords appended to your variables and methods actually mean something now; they basically tell other objects whether or not they have access to them.

- Public variables and methods can be accessed and run by other Objects – wheras private variables and methods can only be accessed within the Object itself.

- So let’s create another new Object – but this time, make it a separate entity from HelloWorld. We’ll just call this our TestObject class, and put a public, and private variable in there.

- As long as the Java files are contained within the same folder, we can create separate objects from ANY class within this folder, by just using our old schema of ClassName any-name = new ClassName();

Objects; Public, and private variables

```
public class TestObject {
    public int publicNumber = 1;
    private int privateNumber = 2;

    public TestObject() {
        System.out.println("I am the TestObject constructor");
    }
}
```

Above: Our new TestObject

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World");
        HelloWorld helloWorld = new HelloWorld();
    }

    public HelloWorld() {
        System.out.println("I am the HelloWorld constructor");
        TestObject testObject = new TestObject();
    }
}
```

Above: Our updated HelloWorld

Note: Constructor now creates a new Object, based on the TestObject class.

Objects; Public, and private variables

- So, let's consider that our HelloWorld class wants to access the publicNumber variable; firstly, we can refer to it through the name we assigned it during our assignment;

```
TestObject testObject = new TestObject();
```

  In this case, we've named our new TestObject class as "testObject" – with a lower case t. This is the standard convention in Java.

- So we can access public methods and variables, by the following schema;

```
testObject.method(arguments); // - Method access
testObject.variable;   // - Variable access
```

  Note: as normal, methods can take in arguments – and if you have not built your method to require any arguments to be passed through, you would use;

```
testObject.method();
```

  The brackets are ALWAYS required for methods – left empty if there are no arguments.

Objects; Public, and private variables

- So we can access publicNumber through the following schema;
  testObject.publicNumber;

  i.e: System.out.println(testObject.publicNumber);
  - Should print out “1” to our console.

- However, what would happen if we attempted this?
  System.out.println(testObject.privateNumber);

- A compile time error would tell you how awful your Java is.

- Private variables can ONLY be accessed by their own object. So if testObject itself wanted to use System.out.println(privateNumber); , it would work fine.
- But as far as other objects in the application go? It doesn’t exist.

Some words on public and private

- You cannot declare “private” or “public” variables within a method. Method variables can ALWAYS be considered private; as they are discarded from memory as soon as the method is finished executing

- If you do not declare a method or variable to be “public”, or “private”, then the Java compiler will assume you meant “private”. As good practice though, you should always type in it’s access level.

- Almost EVERYTHING in Java has an access level – by convention, variables are generally private, while methods are public. We’ll look at this next.

- In Java, private can also be considered “encapsulated” – a common exam question is to explain “encapsulation”. This is different from the PHP encapsulation you may have learned. If asked, explain how a private variable or method works.

Getters and Setters

- So, if variables should normally have private access – how do we access them?

- This is where getters, and setters come in.
  A getter (as the word would imply), is used to get the value of a variable.
  A setter, predictably enough, is used to set the value of a variable.

- So, let's consider the following class;

```
public class TestObject {
    private int privateNumber = 2;

    public TestObject() {

    }
}
```

In order to view and access privateNumber, we'd need to create a getter and setter method; allowing us to view, and change the value within privateNumber from another Object.

Getters and Setters

- So, here's our completed class – and the getters and setters being called from within our HelloWorld class.

```
public class TestObject {
    private int privateNumber = 2;

    public TestObject() {

    }

    public int getPrivateNumber() {
        return privateNumber;
    }

    public void setPrivateNumber(int number) {
        this.privateNumber = number;
    }
}
```

```
public class HelloWorld {

    public static void main(String[] args) {
        HelloWorld helloWorld = new HelloWorld();
    }

    public HelloWorld() {
        TestObject testObject = new TestObject();
        testObject.setPrivateNumber(4);
        int num = testObject.getPrivateNumber();

        System.out.println(num);
    }
}
```

- Getter: public int getPrivateNumber()
  public: this is a public method that another object can call
  int: this method returns an integer
  getPrivateNumber: the name of the method
  (): this method requires no arguments or input.

- Setter: public void setPrivateNumber(int number)
  public: this is a public method that another object can call
  void: this method does not return anything
  setPrivateNumber: the name of the method
  (int number): requires an integer input, which we will refer to as "number" in our code.

Putting this into practice

- Consider your first assignment for CS2500 this year – the beverage dispenser. How would we go about doing this in an Object Oriented manner?
- Consider the structure of the object; each beverage has a name, and a price.
- Consider the methods asked for; getters, computing a price for n pints, and outputting the pint's name and price.
- Try to start new projects with designing your class structure; in this case, we only need one additional object – a Pint class. This can contain a private variable for the price; and then a getter which will return the price to us. The Constructor for the Pint class can take in the price, and set it appropriately.
- Our main class will create Pints – and will contain a static (or instance) method for printing out the pint/price based on the arguments provided.