Java Classes

Summary: In Java, classes can be considered blueprints for objects. These blueprints contain all of the methods and variables that the objects have access to. When a program creates an object, the object is given a type that corresponds to a class. To denote this programmers will often say: `Object` is of type `Class` or `Object` is an instance of `Class`. Both indicate that the object `Object` has access to all the data and methods defined in the `Class` file.
It is important to note that each instance of a class has unique variables, i.e. changing a field in one instance of an object will not change that field in any other objects of that type. Classes can also have methods and variables that are not unique to each instance, these are called static methods and static variables. More on what static means can be found here.

Why we should use it: Putting code relevant to a given class in a single file causes any problems with the class to be localized to that file, making it significantly easier to find where errors are in larger programs that use classes from multiple files. Using classes as a way to design code allows the same class to be used in multiple different programs. For example, the same point class can be used in both a program that graphs lines, in a program that displays scatterplot data, or in a game as a way to store and represent obstacles.

Key definitions:
Class - All objects have a Class that determines what methods and variables they have access to
Method - a section of code in a class that will execute when called, more information can be found here
Foo/Bar - Generic method/class names that don't have any real meaning
Variables - data held by objects of a class, or data available to all members of any given class

Use Case 1: Creating a method in a class

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

Explanation: The code above creates the main method for the Foo class. The main method is the method that will be run when running a java program.

Use Case 2: Creating a class constructor

```
public class Foo {
    int bar;
    public Foo() { //Constructor with no parameters
        this.bar = 1;
    }
}
```

Explanation: Shown is an example of a class with a stored variable. The constructor runs whenever a new instance of the class `Foo` is created in a program. In this example, the value for the internal field `bar` is set to `1`. More about constructors can be found here.