**OBJECT-ORIENTED PROGRAMMING (OOP)**

* Object Oriented is a programming **paradigm**
* Paradigm is **style of programming**



**Benefits of Object-Oriented Programming**

* Object Oriented programming is about objects
* These objects interact with one another to perform various tasks
* As an analogy let’s think of a car
* Cars consist of many collaborating objects
* Almost all of the objects are re-usable or replaceable 

**Course structure:**



**Classes and Objects**

* Class = a blueprint/template for creating objects 
* Object = an instance of class
* From the above example CAR is a class and car1,car2 and car3 are objects

**Creating Classes**

* As before we create a brand new project like HelloWorld
* From here we go to the SRC folder and right click on this to create a separate class



**Remember that naming of a class we use Pascal naming convention with each word starting with a capital**

* We add 3 members within the curly braces namely
  + Field = Public access modifier, then specify the type of this field, and then give it a name (using camelCasing notation) NOTE: As we become more experienced we will know that we do not define fields as public as is done below
  + Let’s declare a couple of methods  
    NOTE: While declaring the method we note that the name of the parameter is exactly the same as the name of the field So we use the this keyword to point to the field  
      
      
      
    

**Creating Objects**

* We will declare this in our main method
* We declare a TextBox variable so call it textBox1 and initialise it as follows 
* We use the new operator to create a new instance of the TextBox class
* NOTE: We are instantiating the TextBox class which means we are creating a new instance of this class
* NOTE: We can make our code even cleaner by not repeating TextBox. We can simplify our code by using the var keyword   
    
    
  NOTE: The Java compiler will detect the type of this variable based on what we have on the right side of the assignment operator(=)
* Next we use the dot operator to check the members of this class   
    
  

**Encapsulation**

* This is the first principal of OOP.
* **Encapsulation** = Bundle the data and methods that operate on the data in a single unit/object
* We can use the encapsulation principal by creating a class  

**Getters and Setters (REVISIT).**

* In Java we use setters to set the value of a field and getter to get the value of a field
* We need to create a method whereby we do some data validation
* Here is an example of the method for a setter and note that the access modifier once we do this will be private in our new Employee class  
    
  
* Here is an example of a setter  
* If we wanted to get the value of the baseSalary field we can use a getter. See below  
   
* NOTE: Very important trick is hover on a field that you want to create a getter/setter and press [alt] + [ENTER] and this will appear. This short cut will have Intellij suggesting to automatically create a getter and setter as we have done before.

**Abstraction**

* This is the second principle of object-oriented programming
* Abstraction = We should reduce complexity by hiding unnecessary details in our classes
* With abstraction we want to hide the implementation detail of a class and treat it like a black box .i.e buttons of a remote control
* Because the fields below are like the transistors in a remote control we made them private. We do not want to work to work directly with these fields.

**Coupling**

* Coupling = The level of dependency between classes
* We need to try to reduce coupling between classes
* The benefit of this is so that once a class is changed then the dependant class need to be modified. The more classes there are dependent then we will have to modify more.
* Buttons on a remote are like different methods

**Reducing Coupling**

* Refer to the Browser class example

**Constructors**

* We can use default constructor to initialise our fields to default values i.e numbers will be zero, Booleans will be false and reference types are going to be null