Exercise 1

Encapsulation:

Encapsulation is defined as the wrapping of variables, together with the code acting on these variables, into a single unit. It almost acts like a protective shield that prevents the data from being accessed by the code outside this shield. Because of this, it is often described as data hiding; the variables within a class are hidden from other classes.

This can be achieved simply by declaring certain variables as private within a class. Public "setter" and "getter" methods are used to access these hidden variables and to set the values of these variables, respectively.

The ideas behind use of Encapsulation include;

1. can construct variables of a class to be read only or write only by omitting setter and getter methods, depending on what is required.
2. makes for codes which are easy to change to reuse for new requirements.
3. the class has total control over what is stored in its "fields" (variables inside a class).

But it is mainly to hide the implementation details from users.

As an example piece of code-

```
public class Encap {

    private String Name;

      public String getName()
        {
          return Name;
        }
```

Here, a private variable Name has been declared which can only be accessed by public methods of a class and so a getter method was implemented. This following example of a class could be used to access this variable, set it and display it-

```
public class EncapTest {

    public static void main(String[] args) {

        Encap obj = new Encap();

        obj.setName("Jimmy James");

        System.out.println("QA Successful Applicant's name: " + obj.getName());

    }

}
```

Inheritance:

This is simply where a subclass gains the properties (methods and fields) of a superclass.

Here, the keyword in terms of coding is “extends” and using the classes from the Encapsulation example-

```
public class EncapTest extends Encap {
```

Where EncapTest is the subclass and Encap is the superclass, meaning the class EncapTest gains the methods and fields from the class Encap.

Similarly with setter and getter methods in Encapsulation, Inheritance too brings with it more subconcepts, e.g. @Override, allowing a subclass to override methods of its superclass, so it may execute its own behaviour. In addition, various types of Inheritance could also be achieved by extending the subclasses further.

Inheritance is very useful as it allows for information to be more mangeable when using similar methods for different classes- it avoids repetition of code.

Polymorphism:

When an object is created from a class, it can sometimes take many forms. This is described as polymorphism.

This in a sense links to Inheritance as an object is polymorphic if it has multiple inheritance. For example-

```
public interface Successful{}
public class Employee{}
public class James extends Employee implements Successful{}
```

The object James is said to be polymorphic as it is an Employee, Successful, an object and also a James. It takes all of these forms.

This is in reference to objects, but another way of describing polymorphism

is “performing a single action by different ways”.

There are two types of polymorphism in java- compile time polymorphism and runtime polymorphism, depending on when a call to a method (overloading and overriding) is resolved.

Abstraction:

Abstraction is a concept used to hide certain details and only show the essential features of the object, hence it deals with the outside view of an object (interface). It hides the implementation details. It is essentially only providing the functionality to the user- what the object does rather than how it does it.

Not to be confused with Encapsulation, Abstraction is induced when deciding what to implement, as opposed to Encapsulation which is hiding something that has already been implemented.

Abstraction is mainly used for identifying common areas and reducing features that will be worked with at different levels of the code. For example, if James and John had been working at a company called BigBusiness for a number of years (i.e. both fall under the category of Employees), a question could be how many years Employees had worked for BigBusiness. The fact that they are both Employees at BigBusiness means there is an obvious similarity and the information has been abstracted and indentified as being common.

If another Employee is added into the code later (e.g. Josephine), the majority of the code would remain oblivious to this fact, and the implementation of Josephine alone deals with Josephines’ particularities i.e. how many years she has worked at BigBusiness. So Abstraction easily allows for the addition of data.

A class or method can be made abstract by simply declaring the term “abstract” after public/private is set, e.g.-

```
public abstract class EncapTest {
```