**Pre-assessment day pack**

Define the following key object-oriented programming (OOP) principles with examples:

* Encapsulation

Encapsulation is where all instance variables and methods are put together into one unit which is a class. This also means data and methods can be put within an object.

Encapsulation can also be referred to as “black box” which you can use without having to think about the internal mechanisms of the product/device. You can access and use the methods contained within the “black box” however they cannot be altered. An example below shows a class of a laptop with various properties which are set when the object is being created using constructors, these are then accessed by public get…() methods.

package example;

public class Laptop {  
 private String brand;

private String os;

private int price;

//Constructor which will set the properties of the object

Laptop(String b, String o, int p){  
 this.brand = b;

this.os = o;

this.price = p;

}

//Method to get the os property of the object

public String getOS(){

return this.os;

}

}

* Inheritance

This is the capability to create classes which can share the attributes and methods of another existing class. This basically extends a previously written class allowing for more features to be added at the time of writing. This is extremely useful as it allows programmers to use previously written code and further extend it.

Below is an example of the laptop class being extended by hP:

package example;

public class HP extends Laptop{

//constructor to set properties of the object

HP(String b, String o, int p){

super(b, o, p);

}

public String getOS(){

return “This is HP : “ + os;

}

}

* Polymorphism

Polymorphism broken down means many forms, poly meaning many and morphos meaning forms. This is where the same word or symbol can be interpreted in different situations based on the context it is being used for. There are two ways java implements polymorphism. One is static polymorphism and the other is Dynamic polymorphism.

Static polymorphism has the ability to execute different method implementations by changing the argument in the method, this is known as overloading.

class overload{

public void print(String s){

System.out.Println(“method with only one string ” + s);

}

public void print(int i){

System.out.println(“method with only one int ” + i);

}

public void print(String s, int i){

System.out.println(“method with one string and one int ” + s + i);

}

}

public class demo {  
 public static void main(String[] args) {  
 overload obj = new overload();

obj.print(“Hello”);

obj.print(50);

obj.print(“Goodbye”, 100);

}

}

As you can see in the example above, there are three different print methods which have different arguments between them. When overloaded properly you are able to call them and the correct version of the method will execute.

The output of the above example would be:

method with only one string Hello

method with only one int 50

method with only one string and one int Goodbye 100

Dynamic polymorphism is when you create an extended class or subclass. This subclass will contain data from the original class also known as the superclass. However there are times when the superclass is not enough for the subclass objects and when it comes to this you will need to override the superclass members.

package example;

public class demo {

public static void main(String[] args) {

//create an object of the supercass, and call get os method

Laptop o = new Laptop(“HP”, “DELL”, “LENOVO”. 100);

System.out.println(o.getOS());

//create an object of the subclass calling the get os method.

HP h = new HP(“Spectre”, “x360”, 200);

System.out.println(h.getOS());

}

}

* Abstraction

Every programming language has abstractions. This is an important element of OOP (object orientated programming). We manage complexity through abstraction, by using interfaces which allow us to use programs or machinery. For example when driving a car you have a steering wheel, gears, pedals to interact with in order to drive the car you don’t care about the internals unless you’re the mechanic of the vehicle testing it.

The use of hierarchical classifications is an excellent way to manage abstraction, as this allows you to break complex systems down into more manageable sizes. Back to the car example, if you look deeper into how the car operates internally you realise that each part of the car is made of subsystems.