QA Consulting Pre-Assessment

**Exercise 1**

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data/variables and coding acting on the data together as one single unit. The variable class in Encapsulation would be hidden from other classes, they can be accessed only through methods of their current class. The Term encapsulation can also be known has data hiding.

To achieve encapsulation within java, the user would start off with declaring the variables of a class as private. Also provide public setter and getter methods to modify and view the variables values.

The following example below would show how encapsulation would work in java.

public class EncapTest {

private String name;

private String idNum;

private int age;

public int getAge() {

return age;

}

public String getName() {

return name;

}

public String getIdNum() {

return idNum;

}

public void setAge( int newAge) {

age = newAge;

}

public void setName(String newName) {

name = newName;

}

public void setIdNum( String newId) {

idNum = newId;

}

}

The example above would show you how encapsulation would work. Within the example you will be able to see public setXXX() and getXXX() methods would be the access points of the instance variables of the EncapTest class. The reason behind using this method would be because they are referred as getters and setters. Meaning any class which would like access to the variables, should access them through these getter and setters.

Inheritance

Inheritance can be defined as the process, where one class would acquire the properties of another. When using inheritance the information is made in a manageable hierarchical order. When the class would inherit the properties of another would be known as a subclass, also a derived class and a child class. The class which is inherited is known as a superclass, also known has base class and parent class.

The keyword which would be used is EXTENDS, where the properties of a class can be inherited. Below you will be able to see the syntax for the following EXTENDS.

class Super {

.....

.....

}

class Sub extends Super {

.....

.....

}

Below you will be able to see sample code involving Java Inheritance. The code will show two classes names Calculation and My\_Calculation.

class Calculation {

int z;

public void addition(int x, int y) {

z = x + y;

System.out.println("The sum of the given numbers:"+z);

}

public void Subtraction(int x, int y) {

z = x - y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class My\_Calculation extends Calculation {

public void multiplication(int x, int y) {

z = x \* y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]) {

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

}

Within the given program, when the object of My\_Calculation class is created a copy of the contents of the superclass would be made within it. By using the objects of the subclass you are able to access the members of a superclass.

Polymorphism

Polymorphism is the ability of an object to take on many forms. The use of polymorphism in OOP is a parent class reference is used to refer to a child class object.

In Java any object that can be able to pass more than on IS-A test is considered to be polymorphic. All java objects would be polymorphic, since the objects will pass the IS-A test for their own type and for the class object.

It would also be important to know that the only possible way to access an object is through a reference variable, the variable can be only of one type. Once it has been declared, the type of reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

The reference variable can also refer to any object of its declared type, or any subtype of its declared type. The reference variable can also be declared as a class or even a interface type.

Below you will be able to see the example

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

The example above would show Deer class is considered to be polymorphic since this has multiple inheritance. Since it would have the following

* Deer IS-A Animal
* Deer IS-A Vegetarian
* Deer IS-A Deer
* Deer IS-A Object

Abstraction

Abstraction for object-oriented programming is the process of hiding the implementation details from the user, the functionality will be provided to the user. Meaning the user would have information on what the object does instead of how it does it. Within Java abstraction is achieved using abstract classes and interfaces.

Abstract classes may or may not contain abstract methods, for example methods without body. If the class has one abstract method then the class must be declared abstract. Once the class has declared to be an Abstract, it cannot be instantiated. If the user would like to use an abstract class, you would have to inherit it from another class, provide implementations to the abstracts methods In it. The inherited abstract class would have to provide implementations to all the abstract methods in it.

For the user to make an abstract class, they would just have to use the key word abstract before the class key word, within the class declaration.

Below you would be able to see an example of how to create an abstract within java.

public abstract class Employee {

private String name;

private String address;

private int number;