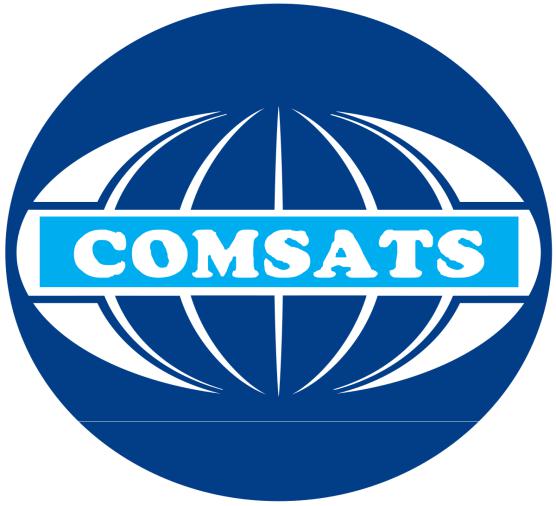
**Object Oriented Programming**

****

**Assignment 2nd**

**Submitted to: Muhammad Ali-Khan**

**Submitted by: Nasratullah (FA18-BCS-401)**

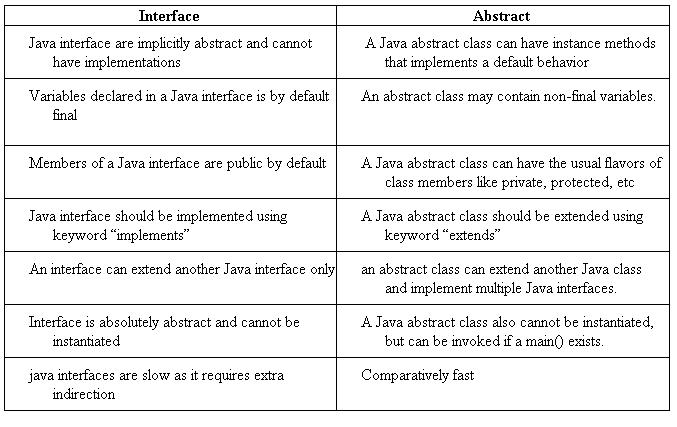
**Submission Date: 26 April,2021.**

**Section: B**

**Q1.** Explain the differences between a class, an abstract class and an interface in Java.

**Answer:** Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body). Interfaces specify what a class must do and not how.

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body). Method that are declared without any body within an abstract class are called abstract method. The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.



**Q2:** Define compile time and run-time polymorphism and provide a code fragment that implements these concepts.

**Answer:** There are two types of polymorphism in java:

1. **Static Polymorphism** also known as compile time polymorphism.
2. **Dynamic polymorphism** also known as runtime Polymorphism.
3. These Type of polymorphism is resolved during compile time is known as Static polymorphism. Method overloading (more than one method having same name but the parameter is different) is an example of compile time polymorphism.

**Ex:**

|  |
| --- |
| class SimpleCalculator{  int add(int a, int b){  return a+b;  }  int add(int a, int b, int c) {  return a+b+c;  }  }  public class Demo  {  public static void main(String args[])  {  SimpleCalculator obj = new SimpleCalculator();  System.out.println(obj.add(10, 20));  System.out.println(obj.add(10, 20, 30));  }  } |

**Output:**

30

60

* **Runtime Polymorphism** is also known as Dynamic Dispatch. Dynamic Polymorphism is a process in which a call to an overridden method is resolved at runtime, that’s why is called runtime polymorphism.

**Ex:**

|  |
| --- |
| class ABC{  public void myMethod(){  System.out.println("Overridden Method");  }  }  public class XYZ extends ABC{  public void myMethod(){  System.out.println("Overriding Method");  }  public static void main(String args[]){  ABC obj = new XYZ();  obj.myMethod();  }  } |

**Output :**

Overriding Method

**Q3**. If you do not specify an access modifier when you declare a member field of a class, what does Java assign it? Demonstrate your answer by providing minimal Java examples that will and will not compile, as appropriate. [6]

**Answer:** If we do not explicitly specify any access modifier for classes, methods, variables, etc, then by default the default access modifier is considered. For example,

|  |
| --- |
| package defaultPackage;  class Logger {  void message(){  System.out.println("This is a message");  }  } |

Here, the Logger class has the default access modifier. And the class is visible to all the classes that belong to the default Package package. However, if we try to use the Logger class in another class outside of default Package, we will get a compilation error.

Q4. Consider the Java class below: [8]

package questions;

public class X {

MODIFIER int value = 3;

};

Another class Y attempts to access the field value in an object of type X. Describe what happens at

compilation and/or runtime for the range of MODIFIER possibilities (i.e., public, protected, private and

unspecified) under the following circumstances:

(a) Y subclasses X and is in the same package;

(b) Y subclasses X and is in a different package;

(c) Y does not subclass X and is in the same package;

(d) Y does not subclass X and is in a different package.

**Answer:** We know about the Visibility and Scope of Access modifier.



(a) Y subclasses X and is in the same package;

**Answer:**

if wesearch out above table we know that public, protected and default can access it in the same package but private is not accessible there is compilation error.

(b) Y subclasses X and is in a different package;

Public and protected can access it in different package but private and default is not accessible there is compilation error.

(c) Y does not subclass X and is in the same package;

Public: within class, package and outside the class is accessible.

Protected: within class, package is accessible but outside the class is not accessible.

default: within class, package is accessible but outside the class is not accessible.

Private: within class is accessible but in package and outside the class is not accessible.

(d) Y does not subclass X and is in a different package.

Public: within class, package and outside the class is accessible.

Protected: within class, package is accessible but outside the class is not accessible.

default: within class, package is accessible but outside the class is not accessible.

Private: within class is accessible but in package and outside the class is not accessible.

**Q5.** A programming language designer proposes adding `selective inheritance' whereby a programmer manually specifies which methods or fields are inherited by any subclasses. Comment on this idea.

**Answer:**  selective inheritance of parameters or fields from parent entities and into child entities responsive to persistent indications of the inheritability of such parameters or fields stored in a non-volatile memory. Further, in many implementations, basing a determination of inheritability on such persistent information permits inheritance to be selectively restricted for a parameter or field without requiring any changes to the database schema for a database that controls access to the parent and child entities.

By using Selective Inheritance, we can inherit those methods which have same behaviour of the child class like a parent. We only write fields and methods in parent class then we override this method.