LAB SUBMISSION 3 - 221047012

1. **Illustrate the importance of Constructor Overloading with appropriate example.**

```
package _221047012;
 class Box
{
    double width, height, depth;

    // constructor used when all dimensions
    // specified
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }

    // constructor used when no dimensions
    // specified
    Box()
    {
        width = height = depth = 0;
    }

    // constructor used when cube is created
    Box(double len)
    {
        width = height = depth = len;
    }

    // compute and return volume
    double volume()
    {
        return width * height * depth;
    }
}

class Constructor
{
    public static void main(String args[])
    {
        // create boxes using the various
        // constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

        // get volume of first box
        vol = mybox1.volume();
        System.out.println(" Volume of mybox1 is " + vol);

        // get volume of second box
        vol = mybox2.volume();
        System.out.println(" Volume of mybox2 is " + vol);

        // get volume of cube
        vol = mycube.volume();
        System.out.println(" Volume of mycube is " + vol);
    }
}
```

2. **<u>With respect to inheritance demonstrate following</u>**
    a. **<u>Java’s support to multi-level inheritance</u>**

```
package Lab3_221047012;

class Shape {
    public void display() {
        System.out.println("Inside display");
    }
}
class Rectangle extends Shape {                  //class rectangle inherits properties of shape
    public void area() {
        System.out.println("Inside area");
    }
}
class Cube extends Rectangle {                   //class cube inherits properties of both
                                                   shape and rectangle
    public void volume() {
        System.out.println("Inside volume");
    }
}
public class Test{
    public static void main(String[] arguments) {
        Cube cube = new Cube();
        cube.display();
        cube.area();
        cube.volume();
    }
}
```

b. **Usage of Super from at method level and constructor level**

```
package Lab3_221047012;

class Animal {                                  // Superclass (parent)
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Dog extends Animal {                      // Subclass (child)
    public void animalSound() {
        super.animalSound();                    // Call the superclass method
        System.out.println("The dog says: bow wow");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();               // Create a Dog object
        myDog.animalSound();                    // Call the method on the Dog object
    }
}
```

c. **Working of Protected access.**

```
package Lab3_221047012;

public class A2_3{
    protected void msg()
    {System.out.println("Hello");
}
}

package L3_221047012;

import Lab3_221047012.*;

class B2_3 extends A2_3{
    public static void main(String args[]){
        B2_3 obj = new B2_3();
        obj.msg();
    }
}
```

3. Differentiate between method overloading and overriding with appropriate example

Overloading:

```
package _221047012;

class Adder
{

                static int add(int a, int b)          //class add with int datatype
                {
                return a+b;
                }
                static double add(double a, double b) //Same class add with double datatype
                {
                return a+b;
                }
                }
                class Overloading1
                {
                public static void main(String[] args)
                {
                System.out.println(Adder.add(11,11));
                System.out.println(Adder.add(2.3,2.6));
                }
                }
```

Overriding:

```
package _221047012;
class Bank{
int getRateOfInterest()             //Method
{
return 0;
}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest()             //Same method name
{
return 8;
}
}

class ICICI extends Bank
{
int getRateOfInterest()              //Same method name
{
return 7;
}
}
class AXIS extends Bank
{
int getRateOfInterest()                          //Same method name
{
return 9;
}
}

class Test{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());      //Same method name of
                                                                           class SBI
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());    // Same method name of
                                                                           class ICICI
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());   // Same method name of
                                                                           class AXIX
}
}
```

4. **Demonstrate the usefulness of finalize() method**

```
package Lab3_221047012;

class Test4
{
   public static void main(String[ ] args)          // A "main thread" gets introduced
   {
      String s = new String("Gate Vidyalay");  // A String object gets created
      s = null;                                        // String Object becomes eligible for garbage
                                                          collection
      System.gc( );                                   // A request is made to JVM for running garbage
                                                          collector ; A "gc thread" gets introduced
      System.out.println("End of main method");
   }
   public void finalize( )                            // Test class finalize( ) method
   {
      System.out.println("Finalize method of Test class");
   }
}
```

5. **Illustrate the concepts of Abstract class and Interface with appropriate example**

```
package _221047012;
interface A{
void a();//by default, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the
methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Abstract_Interface{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}
}
```

6. **Illustrate the significance of Encapsulation – namely the control the concept provides in your application through appropriate examples.**

```
package Lab3_221047012;

class Student {
   private int Student_Id;
   private String name;

   //getters, setters for Student_Id and name fields.
   public int getId() {
       return Student_Id;
   }
   public void setId(int s_id) {
       this.Student_Id = s_id;
   }
   public String getname() {
       return name;
   }
   public void setname(String s_name) {
       this.name = s_name;
   }
}
class Main6{
   public static void main(String[] args) {
       //create an object of Student class
       Student s=new Student();
       //set fields values using setter methods
       s.setId (27);
       s.setname("Abc");
       //print values using getter methods
       System.out.println("Student Data:" + "\nStudent ID:" + s.getId()+ " Student Name:"+
                               s.getname());
   }
}
```