**OBJECT ORIENTED PROGRAMMING**

**LAB# 10 TASKS**

## **Abstract Class,Upcasting and Downcasting in Java**

1. Write a program for exam department which provide abstract class and method of Exam type which contains general methods related to exams and can be used by different department for conducting exams.
2. You have to implement the UML diagram given below.Also Design and implement a subclass “EquilateralTriangle” having a double variable side denoting the three sides of the equilateral triangle [Note that since all the 3 sides are equal, the constructor will have only one parameter]. The area and perimeter of the equilateral triangle are given as follows:

Area = ¼\*\*(*side*)2

Perimeter = 3\**side*

Provide accessor methods for the sides. Test your class using the TestShapes and DownCastingShapes classes.



**Hint given below.**

**Shape.java**

Note the 2 abstract methods in the following abstract class. Also note the presence of two non-abstract methods name()andtoString()

/\*

 \* Shape.java

\*/

public abstract class Shape{

 public String name(){

 return getClass().getName();

 }

 public abstract double area();

 public abstract double perimeter();

 public String toString() {

return "\n" +name() +"\n Area=" +area() +"\nPerimeter=" +perimeter();

 }

}

**Circle.java**
The first non-abstract child of Shape is Circle.  Note that all abstract methods have been provided with an implementation.

**/\***

 **\* Circle.java**

**\*/**

**public class Circle extends Shape {**

 **private double radius;**

 **public Circle(double r){**

 **radius = r;**

 **}**

 **public double area(){**

 **return Math.PI \* (radius \* radius);**

 **}**

 **public double perimeter(){**

 **return 2.0 \* Math.PI \* radius;**

 **}**

 **public double getRadius(){**

 **return radius;**

 **}**

**}**

**Rectangle.java**

**This class is also derived from the Shape class.**

**/\***

 **\* Rectangle.java**

**\*/**

**public class Rectangle extends Shape {**

 **private double length;**

 **private double width;**

 **public Rectangle(double length, double width){**

 **this.length = length;**

 **this.width = width;**

 **}**

 **public double area(){**

 **return length \* width;**

 **}**

 **public double perimeter(){**

 **return 2\*(length+width);**

 **}**

 **public double getLength(){**

 **return length;**

 **}**

 **public double getWidth(){**

 **return width;**

 **}**

**}**

#### Square.java

**/\***

 **\* Square.java**

**\*/**

**public class Square extends Rectangle{**

 **public Square(double length){**

 **super(length, length);**

 **}**

**}**

**TestShapes.java:**

This class basically creates several **Shape** objects and prints them. Note the up-casting in the assignment of the array randomShapes. Since randomShapes is of type Shape, any sub-class of Shape can be assigned to it. Such a reference is called a polymorphic reference. Also note the polymorphic call to the area() and perimeter() methods when the toString() method of each one of the classes is called depending upon the actual shape.

The assignments here are done randomly to illustrate the point that the methods executed belong to the actual object which is dynamically bound to the object. Consider the line highlighted at the end: System.out.println(randomShapes[i]); Looking at the code, you cannot tell which toString() method will be called as the assignment of objects, the length of the array and even the dimensions of the shapes are randomly assigned only to be decided at runtime.

**// TestShapes.java**

**import java.util.Random;**

**public class TestShapes {**

 **public static Shape[] createShape() {**

 **final int SIZE = 5;**

 **final double DIMENSION = 100;**

 **final int NUMBEROFSHAPES = 3;**

 **Random generator = new Random();**

 **//create an array having b/w 1 and SIZE entries**

 **Shape[] randomShapes = new Shape[generator.nextInt(SIZE) + 1];**

 **for(int i = 0; i < randomShapes.length; i++)**

 **{**

 **//randomly generate values b/w 0 and NUMBEROFSHAPES - 1**

 **int assigner = generator.nextInt(NUMBEROFSHAPES);**

 **switch(assigner) {**

 **case 0: randomShapes[i] =**

 **new Rectangle(generator.nextDouble()\*DIMENSION,generator.nextDouble()\*DIMENSION);**

**break;**

 **case 1: randomShapes[i] = new Circle(generator.nextDouble()\*DIMENSION);**

 **break;**

**case 2: randomShapes[i] = new Square(generator.nextDouble()\*DIMENSION);**

 **break;**

 **}**

 **}**

 **return randomShapes;**

 **}**

 **public static void main(String[] args) {**

 **Shape[] randomShapes = TestShapes.createShape();**

 **for(int i = 0; i < randomShapes.length; i++)**

 **System.out.println(randomShapes[i].toString());**

 **}**

**}**

**DownCasting:**

Consider the following program. Here each shape object is being tested whether it’s a Circle, Rectangle or a Square using the instanceof operator. According to that, it’s radius/length/width is also being printed by downcasting it to the required object..

**DownCastingShapes.java**

**/\***

 **\* DownCastingShapes.java**

**\*/**

**public class DownCastingShapes{**

 **public static void main(String[] args){**

 **Shape[] randomShapes = TestShapes.createShape();**

 **for(int i = 0; i < randomShapes.length; i++){**

 **System.out.println(randomShapes[i]);**

**if(randomShapes[i] instanceof Circle)**

 **System.out.println("Radius= " +**

**((Circle) randomShapes[i]).getRadius());**

**else if(randomShapes[i] instanceof Square)**

 **System.out.println("Length= " +**

**((Square) randomShapes[i]).getLength());**

**else if(randomShapes[i] instanceof Rectangle)**

 **System.out.println("Length= " +**

**((Rectangle) randomShapes[i]).getLength()**

 **+ "\nWidth= " +**

**((Rectangle) randomShapes[i]).getWidth());**

 **}**

 **}**

**}**