Lab 11 – Abstract Classes and Interface

File Structure:

```
Lab 11 - Abstract and Interface/
├── Provided/
│   ├── Abstraction/
│   │   ├── Shape.java
│   │   ├── CircleV1.java
│   │   ├── Rectangle.java
│   │   ├── Square.java
│   │   ├── TestShapeV1.java
│   │   └── DownCastingV1.java
│   └── Interface/
│       ├── Eccentric.java
│       ├── Ellipse.java
│       └── CircleV2.java
├── AbstractTask/
│   ├── EquilateralTriangle.java
│   ├── TestShapeV2.java
│   └── DownCastingV2.java
└── InterfaceTask/
    └── TestShapeV3.java
```

Abstraction:

Provided:

class Shape:

```
package Provided.Abstraction;

public abstract class Shape {

    public String name(){
        return getClass().getName();
    }

    public abstract double area();
    public abstract double perimeter();

    @Override
    public String toString() {
        return "\n" + name() +
                "\nArea=" + area() +
                "\nPerimeter=" + perimeter();
    }

}
```

class CircleV1:

```
/*
Version 1: when circle extends Shape
*/


package Provided.Abstraction;

public class CircleV1 extends Shape {
    private double radius;

    public CircleV1(double r){
        radius = r;
    }

    // getter
    public double getRadius() {
        return radius;
    }

    public double area() {
```

```
        return Math.PI * (radius * radius);
    }

    public double perimeter() {
        return 2.0 * Math.PI * radius;
    }


}
```

class Rectangle:

```
package Provided.Abstraction;

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 getLength(){
        return length;
    }
    public double getWidth(){
        return width;
    }

    public double area(){
        return length * width;
    }

    public double perimeter(){
        return 2*(length+width);
    }


}
```

class Square:

```
package Provided.Abstraction;

public class Square extends Rectangle{

    public Square(double length){
        super(length, length); // as length and width both are equal in a square
    }

}
```

Class TestShapeV1:

```
/*
Version 1 of TestShape. (Circle, Rectangle, Square)
 */

package Provided.Abstraction;

import java.util.Random;

public class TestShapeV1 {
    public static Shape[] createShape(){
        final int SIZE = 5;
        final double DIMENSION = 100;
        final int NUMBER_OF_SHAPES = 3;

        Random generator = new Random();

        // declaring array
        Shape[] randomShapes = new Shape[generator.nextInt(SIZE) + 1]; //
generator.nextInt(SIZE) = (0-4), so +1 to make it (1-5)

        // now filling each index with a random shape
        for(int i = 0; i < randomShapes.length; i++){

            int assigner = generator.nextInt(NUMBER_OF_SHAPES); // 0 - 2
            // as we have total 3 shapes,
            // it will give us a number, and we will assign a specific shape
based on that number

            switch (assigner) {
                case 0:
                    randomShapes[i] = new
CircleV1(generator.nextDouble()*DIMENSION);
```

```
                    // multiplying with DIMENSION because generator.nextDouble()
= (0.0 - 1.0)
                    break;
                case 1:
                    randomShapes[i] = new
Rectangle(generator.nextDouble()*DIMENSION, generator.nextDouble()*DIMENSION);
                    break;
                case 2:
                    randomShapes[i] = new
Square(generator.nextDouble()*DIMENSION);
                break;
            }
        }
        return randomShapes;
    }

    public static void main(String[] args) {

        // Shape[] randomShapes = new TestShapeV1().createShape();
        /*
        the new kw is creating obj for TestShapeV1() and then calling
createShape().
        there is no need of creating obj because createShapes() is static

        createShape() will itself return you an address of already created obj in
heap memory.
        */

        // so correct way!
        Shape[] randomShapes = TestShapeV1.createShape();

        for(Shape s : randomShapes){
            System.out.println(s);
        }
    }
}
```

Output:

```
Provided.Abstraction.CircleV1
Area=7987.518963828186
Perimeter=316.8187551087958

Provided.Abstraction.Rectangle
Area=4305.402403517332
Perimeter=264.38054457002886

Provided.Abstraction.CircleV1
Area=366.4860737711257
Perimeter=67.8630962159062

Provided.Abstraction.Rectangle
Area=4680.836003968657
Perimeter=276.50067486169286

Provided.Abstraction.CircleV1
Area=698.0775780754606
Perimeter=93.66056568092299
```

class DownCasting:

```
/*
Version 1 of DownCasting, when equilateral triangle doesn't exists
 */


package Provided.Abstraction;

public class DownCastingV1 {
    public static void main(String[] args) {
        Shape[] randomShapes = TestShapeV1.createShape();

        for (int i = 0; i < randomShapes.length; i++) {
            System.out.println(randomShapes[i]);

            if (randomShapes[i] instanceof CircleV1)
                System.out.println("Radius= " +
((CircleV1)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());
            }
        }
}
```

Output:

```
Provided.Abstraction.Square
Area=6700.492703253637
Perimeter=327.4261493101279
Length= 81.85653732753198

Provided.Abstraction.Rectangle
Area=7675.917534960217
Perimeter=353.3279660441753
Length= 77.07929453001296
Width= 99.58468849207469
```

Task:

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*$\sqrt{3}$ *$(side)^2$

Perimeter = 3**side*

Provide accessor methods for the sides. Test your class using the `TestShapesV2` and `DownCastingV2` classes.

class EquilateralTriangle:

```
package AbstractTask;

import Provided.Abstraction.Shape;

public class EquilateralTriangle extends Shape{
    private double side;

    public EquilateralTriangle(double side) {
        this.side = side;
    }

    public double getSide() {
        return side;
    }

    @Override
    public double area() {
        return (1/4)*(Math.sqrt(3))*(Math.pow(side, 2));
    }

    @Override
    public double perimeter() {
        return (3*side);
    }
}
```

class TestShapesV2:

```
/*
Version 2: (CircleV1, Rectangle, Square, EquilateralTriangle)
*/

package AbstractTask;

import java.util.Random;

import Provided.Abstraction.CircleV1;
import Provided.Abstraction.Rectangle;
import Provided.Abstraction.Shape;
import Provided.Abstraction.Square;
import Provided.Abstraction.TestShapeV1;

public class TestShapeV2 {
    public static Shape[] createShape(){
        final int SIZE = 5;
        final double DIMENSION = 100;
        final int NUMBER_OF_SHAPES = 4; // shapes are now 4

        Random generator = new Random();

        Shape[] randomShapes = new Shape[generator.nextInt(SIZE) + 1]; //
generator.nextInt(SIZE) = (0-4), so +1 to make it (1-5)

        // now filling each index with a random shape
        for(int i = 0; i < randomShapes.length; i++){

            int assigner = generator.nextInt(NUMBER_OF_SHAPES); // 0-3
            // as we have total 4 shapes,
            // it will give us a number, and we will assign a specific shape
based on that number

            switch (assigner) {
                case 0:
                    randomShapes[i] = new
CircleV1(generator.nextDouble()*DIMENSION);
                    // multiplying with DIMENSION because generator.nextDouble()
= (0.0 - 1.0)
                    break;
                case 1:
                    randomShapes[i] = new
Rectangle(generator.nextDouble()*DIMENSION, generator.nextDouble()*DIMENSION);
```

```
                    break;
                case 2:
                    randomShapes[i] = new
Square(generator.nextDouble()*DIMENSION);
                    break;
                case 3:
                    randomShapes[i] = new
EquilateralTriangle(generator.nextDouble()*DIMENSION);
                break;

            }
        }
        return randomShapes;
    }

    public static void main(String[] args) {

        Shape[] randomShapes = TestShapeV1.createShape();

        for(Shape s : randomShapes){
            System.out.println(s);
        }
    }

}
```

Output:

```
Provided.Abstraction.Square
Area=417.11641101799535
Perimeter=81.69371197520606

Provided.Abstraction.Square
Area=663.4012109053249
Perimeter=103.02630428431954

Provided.Abstraction.CircleV1
Area=4708.955627778714
Perimeter=243.2580556225008

Provided.Abstraction.Square
Area=30.612142514054124
Perimeter=22.13129639729372
```

class DownCastingV2:

```
/*Version 2 */
package AbstractTask;
import Provided.Abstraction.CircleV1;
import Provided.Abstraction.Rectangle;
import Provided.Abstraction.Shape;
import Provided.Abstraction.Square;
import Provided.Abstraction.TestShapeV1;
public class DownCastingV2 {
    public static void main(String[] args) {
        Shape[] randomShapes = TestShapeV1.createShape();

        for (int i = 0; i < randomShapes.length; i++) {
            System.out.println(randomShapes[i]);

            if (randomShapes[i] instanceof CircleV1){
                System.out.println("Radius= " +
((CircleV1)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());
            }
            else if (randomShapes[i] instanceof EquilateralTriangle){
                System.out.println("Length of each side = " +
((EquilateralTriangle)randomShapes[i]).getSide());
            }
        }
    }
}
```

Output:

```
Provided.Abstraction.Square
Area=126.09475405156266
Perimeter=44.916768192124
Length= 11.229192048031

Provided.Abstraction.Square
Area=1513.553682610969
Perimeter=155.61766905392042
Length= 38.904417263480106
```

Interface:

Provided:

interface Eccentric:

```
package Provided.Interface;

public interface Eccentric {

    // as there is diff formula for the calculation of eccnetricity
    double eccentricuty();

}
```

class Ellipse:

```
package Provided.Interface;

import Provided.Abstraction.Shape;

public class Ellipse extends Shape implements Eccentric {
    private double a; // major radius
    private double b; // minor radius

    public Ellipse(double s1, double s2) {

        // whichever is greater assign it to a
        if (s1 < s2) {
            this.a = s2;
            this.b = s1;
        } else {
            this.a = s1;
            this.b = s2;
        }
    }

    @Override
    public double perimeter() {
        // Perimeter = P = π*2*underroot((a^2 + b^2) - ((a - b)^2)/2) [Note that
if a =
        // b = r, then P = 2πr]
        // if a = b means it is circle

        if (a == b) {
            return 2 * Math.PI * a;
```

```
        }

        double sqrtPart = Math.sqrt((a * a + b * b) - Math.pow((a - b), 2) / 2);

        return Math.PI * 2 * sqrtPart;
    }

    @Override
    public double area() {
        return Math.PI * a * b;
    }

    @Override
    public double eccentricuty() {
        // Eccentricity = e = 1 - (b^2)/(a^2)

        return 1 - (Math.pow(b, 2) / Math.pow(a, 2));
    }

    @Override
    public String toString() {
        return "\n" + name() +
                "\nArea = " + area() +
                "\nPerimeter = " + perimeter() +
                "\nEccentricity = " + eccentricuty();
    }

}
```

Task 02:
By looking at the formulae for an ellipse, provide the missing code for all of the methods in the class `Ellipse` including the `toString()` method. Test your program using the `TestShapes.java` class. Your output should look as follows (for an ellipse with a = 10 and b = 7) (values are randomly generated).

- *missing code is already completed above*

Task 03:
How about the following class CircleV2. Since a Circle is a special case of an Ellipse, will the output of TestShapes.java be affected if the following class is used instead of the class Circle used previously:

class CircleV2:

```
/*
this time it inherits Ellipse
*/

package InterfaceTask;

import Provided.Interface.Ellipse;

public class CircleV2 extends Ellipse{

    public CircleV2(double radius){
        super(radius, radius);
    }

}
```

class TestShapeV3:

```
/*
Version 3: (Square, Cirlce)
this time i have taken CircleV2.
CircleV2 extends Ellipse
 */


package InterfaceTask;

import java.util.Random;

import Provided.Interface.Ellipse;
import Provided.Abstraction.Shape;
import Provided.Abstraction.Square;


public class TestShapeV3 {
    public static Shape[] createShape() {

        final int SIZE = 5;
        final double DIMENSION = 100;
        final int NUMBER_OF_SHAPES = 2;   // now we have only 2 shapes

        Random generator = new Random();

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

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

            int assigner = generator.nextInt(NUMBER_OF_SHAPES); // gives 0 or 1

            switch (assigner) {
                case 0:
                    randomShapes[i] = new
Square(generator.nextDouble()*DIMENSION);
                    break;
                case 1:
                    randomShapes[i] = new
Ellipse(generator.nextDouble()*DIMENSION, generator.nextDouble()*DIMENSION);
                    break;
            }
        }

        return randomShapes;
```

```
    }

    public static void main(String[] args) {

        // correct way to call a static method
        Shape[] randomShapes = TestShapeV3.createShape();

        for (Shape s : randomShapes) {
            System.out.println(s);
        }



        // Task 2: Testing circle here
        Ellipse circle = new CircleV2(50.0);

        System.out.println(circle);
    }
}
```

Output:

```
Provided.Abstraction.Square
Area=4154.5485612534785
Perimeter=257.82315058980964

Provided.Interface.Ellipse
Area = 191.98889731060314
Perimeter = 70.3079860931954
Eccentricity = 0.463675287833472

Provided.Interface.Ellipse
Area = 2008.7941492947077
Perimeter = 246.20495774580246
Eccentricity = 0.8238997071047216

InterfaceTask.CircleV2
Area = 7853.981633974483
Perimeter = 314.1592653589793
Eccentricity = 0.0
```