Guided LAB - 303.10.3 - Abstraction

Lab Objective:

In this lab, you will demonstrate the Java Abstraction and we will utilize the previous lab (GLAB - 303.10.2),
By the end of this lab, learners will be able to use Java Abstraction in Java applications.

Introduction:

In the previous lab (GLAB - 303.10.2), we used examples of Shapes. We created Circle, Rectangle, and Triangle objects. The Shape class can only be used as a superclass for Inheritance and Polymorphism purposes; it cannot be used for objects. The class that is not used for creating objects is known as abstract.

Using an abstract class, you can improve the ***Shape*** class that was shown in the previous lab (GLAB - 303.10.2). Since there is no meaningful concept of area for an undefined two-dimensional shape, the following version of the program declares ***getArea( )*** as an abstract method inside the ***Shape* class**. This means that all classes derived from the ***Shape* class** must override ***getArea( )***.

Remember that we cannot instantiate the **Abstract class**; so there is no need to create a Constructor in the **Abstract class**. We will remove the constructor from the Shape class and make a few changes in all subclasses accordingly.

Create a class named **Shape**. This will be an Abstract class and superclass. Write the code below:

```
public abstract class Shape {
   protected String color;
   protected double height;
   protected double width;
   protected double base;

   public void setColor(String color) {
       this.color = color;
   }

   public void setWidth(double width) {
       this.width = width;
   }

   public void setHeight(double height) {
       this.height = height;
   }

   public void setBase(double base) {
       this.base = base;
   }

// The getArea method is abstract.
   // It must be overridden in a subclass.
   /** All shapes must provide a method called getArea() */.
   public abstract double getArea();
   /** Returns a self-descriptive string */

   public String toString() {
       return "Shape[color=" + color + "]";
   }

   public void displayshapName()
   {
       System.out.println("I am a Shape.");
   }
}
```

Create a class named **Circle**. This will be a Child class. Write a code below:

```
public class Circle extends Shape {
   protected double radius;
   private final double PI = Math.PI;

   public Circle(double radius) {
       this.radius = radius;
   }

   public Circle(double radius, double height) {
       this.radius = radius;
       super.height = height;
   }
   public double getArea() {
       //double area = PI * this.radius * this.radius;
       double area = PI * Math.pow(this.radius, 2); // initializing value in
parent class variable
       return area; //reference to  parent class variable
   }
   @Override
   public void displayshapName() {
       System.out.println("Drawing a Circle of radius " + this.radius);
   }
   /** Returns a self-descriptive string */
   @Override
   public String toString() {
       return "Circle[ radius = " + radius + super.toString() + "]";
   }
}
```

Create a class named **Rectangle**. This will be a Child class. Write the code below:

```
public  class Rectangle extends Shape {

  public Rectangle(String color) {
      super.color = color;
  }

  public Rectangle() {
  }
  public Rectangle(String color, double width, double height) {
```

```
    super.height = height;
    super.width = width;
    super.color = color;
   }
   @Override
   public double getArea() {
       return super.width *  super.height;
   }
   //Overriding method of base class with different implementation
   @Override
   public void displayshapName() {
       System.out.println("I am a Rectangle"  );
   }
   @Override
   public String toString() {
       return "Rectangle[height=" + height + ",width=" + width + "," +
super.toString() + "]";
   }
}
```

Create a class named **Triangle**. This will be a Child class. Write the code below:

```
public class Triangle extends Shape {

   public Triangle(){}


   public Triangle(String color) {
      super.color = color;
   }
   public void setBase(int base) {
       this.base = base;
   }
   @Override
   public double getArea() {
          return 0.5*super.base * super.height;
   }
   //Overriding method of base class with different implementation
   @Override
   public void displayshapName() {
       System.out.println("I am a TriAngle"  );
   }
   /** Returns a self-descriptive string */
   @Override
```

```
   public String toString() {
       return "Triangle[base=" + super.base + ",height=" + super.height + "," +
super.toString() + "]";
   }
}
```

Create a class named **myRunner**. This will be the Main class or **entry point** for the application. Write the code below:

```
public class myRunner {
   public static void main(String[] args) {

       Circle c = new Circle(100);
       System.out.println("Area of Circle " + c.getArea());

        // Shape sObj = new Shape(); // This will give Error, we can not
instantiate Abstract class

       // object creation of the Circle class
       System.out.println("++++++++++++++++++");
       // it's fine because a Circle is a Shape by inheritance
       Shape shapeCircleObj = new Circle(100);  // UpCasting
       shapeCircleObj.displayshapName();
       System.out.println("Area of Circle " + shapeCircleObj.getArea());
       System.out.println(shapeCircleObj);  // Run circle's toString()
       // Use instanceof operator for Validation
       System.out.println(shapeCircleObj instanceof Circle); // true

       System.out.println("++++++++++++++++++");
       Shape shapeRectangleObj = new Rectangle("Red"); //UpCasting
       shapeRectangleObj.displayshapName();
       shapeRectangleObj.setHeight(2);
       shapeRectangleObj.setWidth(4);
       System.out.println("Area of Rectangle is " +
shapeRectangleObj.getArea());
       System.out.println(shapeRectangleObj);  // Run Rectangle's toString()
       // Use instanceof operator for Validation
       System.out.println(shapeRectangleObj instanceof Rectangle); // true
```

```
        System.out.println("--------------------");
        Shape shapeTriangleObj = new Triangle("Blue"); //UpCasting
        shapeTriangleObj.displayshapName();
        shapeTriangleObj.setHeight(10);
        shapeTriangleObj.setBase(15);
        System.out.println("Area of Triangle is " + shapeTriangleObj.getArea());
        System.out.println(shapeTriangleObj);  // Run Triangle's toString()
    }
}
```

Output:

```
Area of Circle 31415.926535897932
++++++++++++++++++
Drawing a Circle of radius 100.0
Area of Circle 31415.926535897932
Circle[ radius = 100.0Shape[color=null]]
true
I am a Rectangle
Area of Rectangle is 8.0
Rectangle[height=2.0,width=4.0,Shape[color=Red]]
true
--------------------
I am a TriAngle
Area of Triangle is 75.0
Triangle[base=15.0,height=10.0,Shape[color=Blue]]
true
```

Submission Instructions:

Include the following deliverables in your submission:

- Submit your source code using the Start Assignment button in the top right corner of the assignment page in Canvas.

CANVAS STAFF USE ONLY: Canvas Submission Guideline:

Instructions for Canvas Assignment Creation
Assignment Name: GLAB - 303.14 - Abstraction **Points:** 100 **Assignment Group:** Module 303: Java SE Review (Not Graded) **Display Grade As:** Complete/Incomplete **Do not count this assignment towards the final grade:** Checked **Submission Types:** File uploads **Everything else is the default.**