Guided LAB - 303.10.2 - Polymorphism, Inheritance, Overriding, Object Type Casting

Introduction:

In this lab, we will demonstrate how to achieve Polymorphism using Inheritance, Overriding, Object Type Casting, Encapsulation, and **instanceOF** objects.

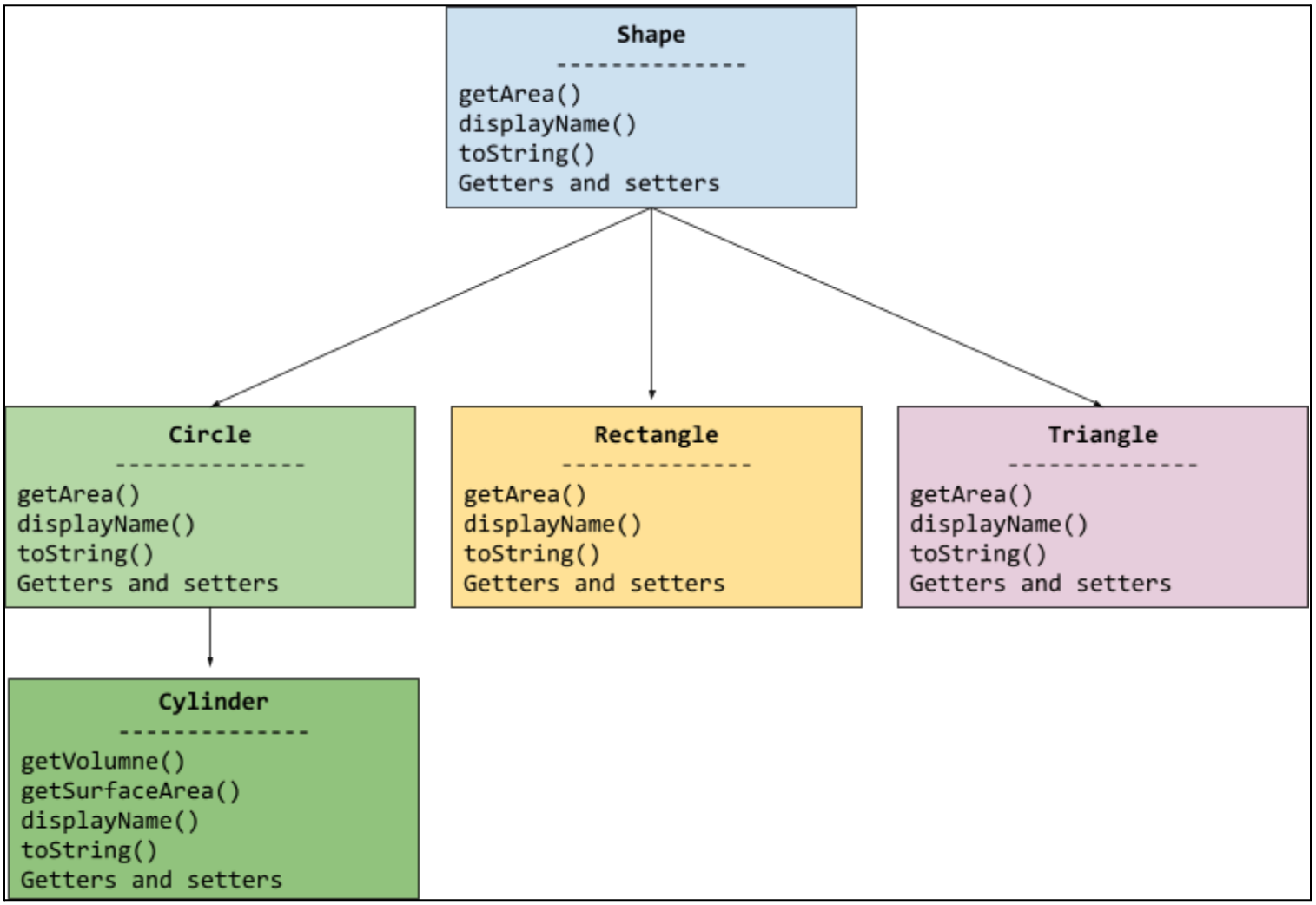
Objective:

By the end of this lab, learners will be able to use Polymorphism concepts using Inheritance, Encapsulation, and **instanceOF** objects.

One of the key features of Inheritance is that a ***reference variable (Object)*** of a superclass type can point to an object of its subclass. Polymorphism is the art of taking advantage of this simple but powerful and versatile feature.

Consider the following illustration:

> Suppose that our program uses many kinds of ***Shapes***, such as triangles, rectangles, and so on. We should design a superclass called Shape, which defines the public (common) behaviors of all the shapes.
>
> For example, we would like all shapes to have a method called **getArea**(), which returns the area of that particular shape.



Instructions:

For the demonstration, we will use both ***constructors*** and the ***setter*** methods for value initialization.

Step 1:

Create a class named **Shape**. This will be a Super or Parent class. Write the code below:

```
public class Shape {
   private String color;
   protected double area = 1.0;
   protected double base = 1.0;
```

```
    protected double width = 1.0;
    protected double height = 1.0;

    /** Constructs a Shape instance with only the given color */
    public Shape (String color) {
        this.color = color;
    }
    public Shape()
    {
    }
    /** Constructs a Shape instance with the given values */
    public Shape(String color, double area, double base, double width,
double height) {
        this.color = color;
        this.area = area;
        this.base = base;
        this.width = width;
        this.height = height;
    }

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

    public void setArea(double area) {
        this.area = area;
    }

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

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

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

    /** Returns a self-descriptive string */
    @Override
    public String toString() {
        return "Shape[color=" + color + "]";
    }

    /** All shapes must provide a method called getArea() */
    public double getArea() {
        // We have a problem here!
```

```
        // We need to return some value to compile the program.
        System.out.println("Shape unknown! Cannot compute area!");
        return 0;
    }
    public void displayshapName()
    {
        System.out.println("I am a Shape.");
    }
}
```

Step 2:

Create a class named **Circle**. This will be a Child class. Write the 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;
        super.area = PI * Math.pow(this.radius, 2); // initializing value in
parent class variable
        return super.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() + "]";
    }
}
```

Step 3:

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);
   }

   public Rectangle() {
   }

public Rectangle(String color, double area, double base, double width, double
height)
   {
       super(color, area, base, width, height);
   }

   @Override
   public void setBase(double base) {
       super.base = base;    }
   @Override
   public void setWidth(double width) {
       super.width = width;    }
   @Override
   public double getArea() {
       return width * height;    }

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

Step 4:

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, double area, double base, double width, double
height) {
        super(color, area, base, width, height);
   }

   public Triangle(String color) {
        super(color);
   }

   @Override
   public void setBase(double base) {
        super.base = base;
   }
   @Override
   public void setWidth(double width) {
        super.width = width;
   }
   @Override
   public double getArea() {
          return 0.5*base*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=" + base + ",height=" + height + "," +
super.toString() + "]";
   }
}
```

Step 5:

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

```
public class Cylinder extends Circle {
   private final double PI = Math.PI  ;

   public Cylinder(double radius, double height) {
       super(radius, height);
       // TODO Auto-generated constructor stub
   }

   public Cylinder(double radius) {
       super(radius);
   }

   /** Returns the volume of this cylinder */
   public double getVolumne() {
       return  (PI*Math.pow(super.radius, 2)) * super.height;
   }

   public double getSurfaceArea() {
       return 2.0 * Math.PI*super.radius*super.height;
   }
   @Override
   public void displayshapName() {
       System.out.println("Drawing a Cylinder for radius " +
super.radius);
   }
   public String toString()
   {
       return "radius is: " + super.radius +"height is : " +
super.height;

   }
}
```

Step 6:

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());

       // Example of Object type casting
       // declaration of object variable obj of the Shape class

      // Shape sObj ; // object creation of the Shape class
       Shape sObj = new Shape();
       sObj.displayshapName();
       System.out.println(sObj instanceof Shape); // true


      // 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(sObj instanceof Circle); // false because Shape is not a Circle

       System.out.println("--------------------");
       Shape shapeRectangleObj = new Rectangle("Red"); //UpCasting
       shapeRectangleObj.displayshapName();
       shapeRectangleObj.setHeight(2);
       shapeRectangleObj.setWidth(2);
       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(sObj instanceof Rectangle); // false because Shape is not
a Rectangle
```

```
        System.out.println("--------------------");
        Shape shapeTriangleObj = new Triangle("Blue"); //UpCasting
        shapeTriangleObj.displayshapName();
        shapeTriangleObj.setHeight(2);
        shapeTriangleObj.setBase(3);
        System.out.println("Area of Triangle is " + shapeTriangleObj.getArea());
        System.out.println(shapeTriangleObj);  // Run Triangle's toString()

        // Use instanceof operator for Validation
        System.out.println(shapeTriangleObj instanceof Triangle); // true
        System.out.println(sObj instanceof Triangle); // false because Shape is not a
Triangle
        System.out.println("--------------------");

        Cylinder cylinderShape = new Cylinder(3); //UpCasting
        cylinderShape.displayshapName();
        cylinderShape.setHeight(3);
        System.out.println("Area of Cylinder is " + cylinderShape.getVolumne());
        System.out.println(cylinderShape);  // Run cylinderShape's toString()
    }
}
```

- In the above example, we have created objects of the Shape class: ***sObj, shapeCircleObj, shapeRectangleObj,*** and ***shapeTriangleObj***. These objects are **polymorphic variables**.
- We can summarize this by stating that Superclass reference variables are polymorphic reference variables. They can refer to objects of their own class or objects of the subclasses inherited from their class.
- The ***instanceof*** operator is a Boolean operator that tests whether an object belongs to a given class.

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 - 303.10.2 - Polymorphism, Inheritance, Overriding, Object type Casting **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 Upload **Everything else is the default.**