

Lab 9 - Assignment 5:- CSYE6200 - Harsh Sangani

Problem 1: -

Problem Description -

The problem at hand involves implementing a class named Octagon that extends GeometricObject and incorporates the Comparable and Cloneable interfaces. This Octagon class is designed to represent an octagon with equal-length sides, calculating its area and perimeter. The challenge is not just in the technical implementation but in understanding the broader significance of creating such a class. By tackling this problem, one can grasp the application of interfaces for comparison and cloning, reinforcing object-oriented programming principles. The problem provides an opportunity to think deeply about the structural design of classes, the utilization of interfaces, and the practical implications of geometric calculations. The skills gained from solving this problem can be transferred to other scenarios, fostering a solid foundation in Java programming and object-oriented design that extends beyond this specific context.

Analysis -

To solve the problem, I would design the Octagon class to store the side length, implement the necessary interfaces (Comparable and Cloneable), and define methods for calculating area and perimeter based on the provided formula. The compareTo method would compare the areas of two Octagon objects.

Difficulties might arise in accurately implementing the mathematical formula for area calculation and ensuring proper utilization of interfaces. Additionally, handling the cloning process correctly can pose challenges.

A potential improvement could involve using composition over inheritance, where the Octagon class could encapsulate a GeometricObject instance rather than extending it. This design might enhance flexibility and maintainability by avoiding potential issues related to multiple inheritance.

Furthermore, exploring alternative area calculation methods or considering optimizations could be valuable. However, the provided formula seems straightforward for an octagon with equal-length sides.

Overall, the chosen design should balance simplicity, adherence to best practices, and effective use of Java interfaces while addressing the specific requirements of the problem.

Difficulties Encountered –

1. **Mathematical Accuracy:** Ensuring precise implementation of the area calculation formula ($A = (2 + 4\sqrt{2}) * s^2 * 0.5$) can be challenging, as any error in the formula implementation may lead to incorrect results.
2. **Interface Implementation:** Integrating the Comparable and Cloneable interfaces correctly may pose challenges, especially if there's ambiguity in how these interfaces should be implemented for the specific Octagon class.
3. **Cloning Process:** Implementing the clone() method correctly to create a deep copy of the Octagon object without introducing unintended side effects or breaking encapsulation can be complex.
4. **Testing and Debugging:** Verifying the correctness of the implemented methods, especially compareTo and clone, may require thorough testing and debugging to ensure the Octagon class behaves as expected in various scenarios.
5. **Understanding Geometric Concepts:** A solid understanding of geometric concepts, such as the relationships between sides, angles, and area for an octagon, is crucial. Misinterpretation of these concepts may lead to incorrect implementations.
6. **Decision on Design Choices:** Deciding on the appropriate design choices, such as whether to extend GeometricObject or use composition, involves considering trade-offs and understanding the implications of each design decision.

Conclusion -

Overcoming these difficulties requires careful consideration, testing, and validation of the implemented Octagon class to ensure its correctness, efficiency, and adherence to best practices in Java programming.

Source Code –

Lab9P1.java –

```
package edu.northeastern.csye6200;

public class Lab9P1 {
    public static void main(String[] args) {
        // TODO: write your code here

        Octagon octagon1 = new Octagon(9);

        System.out.println("Area is " + octagon1.getArea());
        System.out.println("Perimeter is " + octagon1.getPerimeter());
    }
}
```

```

try {
    Octagon octagon2 = (Octagon) octagon1.clone();

    System.out.println("Compare the methods: " + octagon1.compareTo(octagon2));
} catch (CloneNotSupportedException e) {
    e.printStackTrace();
}
}
}

```

```

class Octagon extends GeometricObject implements Comparable<Octagon>, Cloneable {
    private double side;

```

```

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

```

```

    public double getSide() {
        return side;
    }

```

```

    @Override
    public double getArea() {
        return (2 + 4 / Math.sqrt(2)) * side * side;
    }

```

```

    @Override
    public double getPerimeter() {
        return 8 * side;
    }

```

```

    @Override
    public int compareTo(Octagon o) {
        if (this.getArea() > o.getArea()) {
            return 1;
        } else if (this.getArea() < o.getArea()) {
            return -1;
        } else {
            return 0;
        }
    }
}

```

```

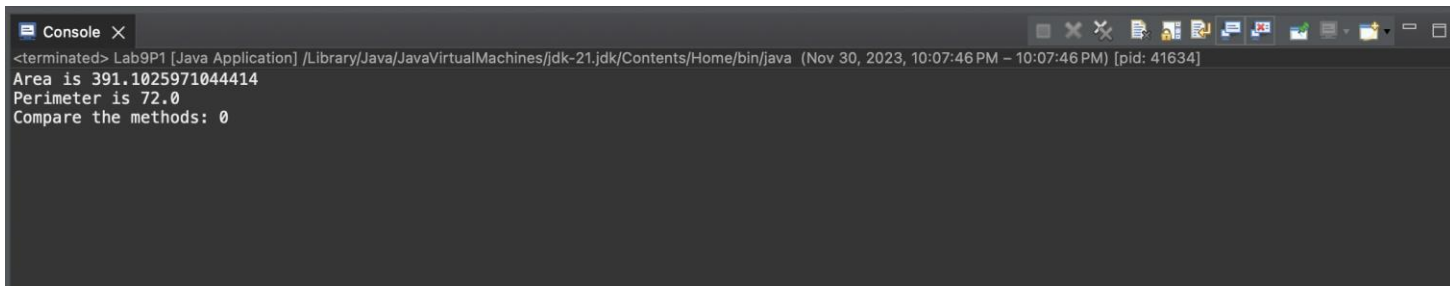
    @Override

```

```
    public Object clone() throws CloneNotSupportedException {  
        return super.clone();  
    }  
}
```

```
abstract class GeometricObject {  
    public abstract double getArea();  
  
    public abstract double getPerimeter();  
}
```

Screenshots of sample runs –

A screenshot of a Java application console window. The title bar reads "Console X". The window content shows the following text: "<terminated> Lab9P1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Nov 30, 2023, 10:07:46 PM – 10:07:46 PM) [pid: 41634]", "Area is 391.1025971044414", "Perimeter is 72.0", and "Compare the methods: 0".

```
<terminated> Lab9P1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Nov 30, 2023, 10:07:46 PM – 10:07:46 PM) [pid: 41634]  
Area is 391.1025971044414  
Perimeter is 72.0  
Compare the methods: 0
```

Problem 2: -

Problem Description –

The problem involves designing an interface named `Colorable` and implementing it in a class named `Square` that extends `GeometricObject`. The interface has a method `howToColor()`, and the `Square` class provides a specific implementation for coloring. A test program then creates an array of various geometric objects, showcasing their areas and invoking the color method if applicable.

This problem is not just about implementing an interface but understanding the significance of abstraction and polymorphism. By using an interface, it enforces a contract that all colorable objects must adhere to, promoting code consistency and making the system more modular. The concept of extending an existing class (`GeometricObject`) and implementing an interface (`Colorable`) illustrates the flexibility of Java's object-oriented design.

The test program exemplifies the power of polymorphism, treating objects uniformly through their shared interface. The problem encourages thinking beyond the immediate task, prompting considerations such as when to use interfaces, how to design for flexibility and future modifications, and the importance of clear and concise contracts between classes.

Solving this problem provides not just a technical solution but a deeper understanding of design principles in object-oriented programming. The lessons learned here can be applied to other scenarios involving interfaces, inheritance, and polymorphism, contributing to a broader skill set in software design.

Analysis –

To solve the problem, I would start by defining the `Colorable` interface with a `void howToColor()` method. The `Square` class would then extend `GeometricObject` and implement the `Colorable` interface, providing a specific implementation for `howToColor()` that prints "Color all four sides." The test program would create an array of `GeometricObjects`, allowing for polymorphic behavior.

Design/Algorithm –

1. Define Interface: Create the `Colorable` interface with the `howToColor()` method.
2. Implement Class: Design the `Square` class that extends `GeometricObject` and implements `Colorable`, providing the required implementation for `howToColor()`.
3. Test Program: Write a test program that creates an array of `GeometricObjects`, including instances of `Square`, `Circle`, and `Rectangle`. Iterate through the array, display the area of each object, and invoke `howToColor()` if the object is colorable.

Difficulties Encountered –

1. Interface Implementation: Ensuring that the `Square` class correctly implements the `Colorable` interface, including providing a meaningful implementation for `howToColor()`.
2. Polymorphism Handling: Ensuring that the test program correctly handles polymorphism, distinguishing colorable objects and invoking the appropriate methods.

Possible Improvements –

An alternative design might involve using an abstract class for `GeometricObject` instead of an interface, allowing for a default implementation of common geometric features. However, this could limit flexibility if there's a need for classes to extend other classes.

Conclusion –

Additionally, considering a more elaborate hierarchy of geometric shapes could lead to a more extensible solution. For instance, having a base class for all shapes and then deriving specific shapes like `Square`, `Circle`, and `Rectangle` could allow for a more organized class structure. However, this would depend on the specific requirements and potential future extensions of the system.

Source Code –

Lab9P2.java –

```
package edu.northeastern.csye6200;

public class Lab9P2 {
    public static void main(String[] args) {
        // TODO: write your code here

        GeometricObject[] objects = {
            new Square(2),
            new Circle(5),
            new Square(5),
            new Rectangle(3, 4),
            new Square(4.5)
        };

        for (GeometricObject object : objects) {
            System.out.println("Area is " + object.getArea());

            if (object instanceof Colorable) {
```

```

        ((Colorable) object).howToColor();
    }

    System.out.println();
}
}

interface Colorable {
    void howToColor();
}

abstract class GeometricObject {
    public abstract double getArea();

    public abstract double getPerimeter();
}

class Square extends GeometricObject implements Colorable {
    private double side;

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

    @Override
    public double getArea() {
        return side * side;
    }

    @Override
    public double getPerimeter() {
        return 4 * side;
    }

    @Override
    public void howToColor() {
        System.out.println("Color all four sides");
    }
}

class Circle extends GeometricObject {
    private double radius;

```

```

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

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}

class Rectangle extends GeometricObject {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        return width * height;
    }

    @Override
    public double getPerimeter() {
        return 2 * (width + height);
    }
}

```

Circle.java –

```

package edu.northeastern.csye6200;

public class Circle extends GeometricObject {
    private double radius;

    /**Default constructor*/
    public Circle() {

```



```

    this(1.0);
}

/**Construct circle with a specified radius*/
public Circle(double radius) {
    this(radius, "white", false);
}

/**Construct a circle with specified radius, filled, and color*/
public Circle(double radius, String color, boolean filled) {
    super();
    this.radius = radius;
}

/**Return radius*/
public double getRadius() {
    return radius;
}

/**Set a new radius*/
public void setRadius(double radius) {
    this.radius = radius;
}

/**Implement the getArea method defined in GeometricObject*/
public double getArea() {
    return radius*radius*Math.PI;
}

/**Implement the getPerimeter method defined in GeometricObject*/
public double getPerimeter() {
    return 2*radius*Math.PI;
}

/**Override the equals() method defined in the Object class*/
public boolean equals(Circle circle) {
    return this.radius == circle.getRadius();
}

@Override
public String toString() {
    return "[Circle] radius = " + radius;
}
}

```

GeometricObject.java –

```
package edu.northeastern.csye6200;
```

```
// GeometricObject.java: The abstract GeometricObject class
```

```
public abstract class GeometricObject {

    private String color = "white";
    private boolean filled;

    /** Default construct */
    protected GeometricObject() {
    }

    /** Construct a geometric object */
    protected GeometricObject(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
    }

    /** Getter method for color */
    public String getColor() {
        return color;
    }

    /** Setter method for color */
    public void setColor(String color) {
        this.color = color;
    }

    /**
     * Getter method for filled. Since filled is boolean, so, the get method
     * name is isFilled
     */
    public boolean isFilled() {
        return filled;
    }

    /** Setter method for filled */
    public void setFilled(boolean filled) {
        this.filled = filled;
    }
}
```

```

    /** Abstract method findArea */
    public abstract double getArea();

    /** Abstract method getPerimeter */
    public abstract double getPerimeter();
}

```

Rectangle.java –

```

package edu.northeastern.csye6200;

public class Rectangle extends GeometricObject {
    private double width;
    private double height;

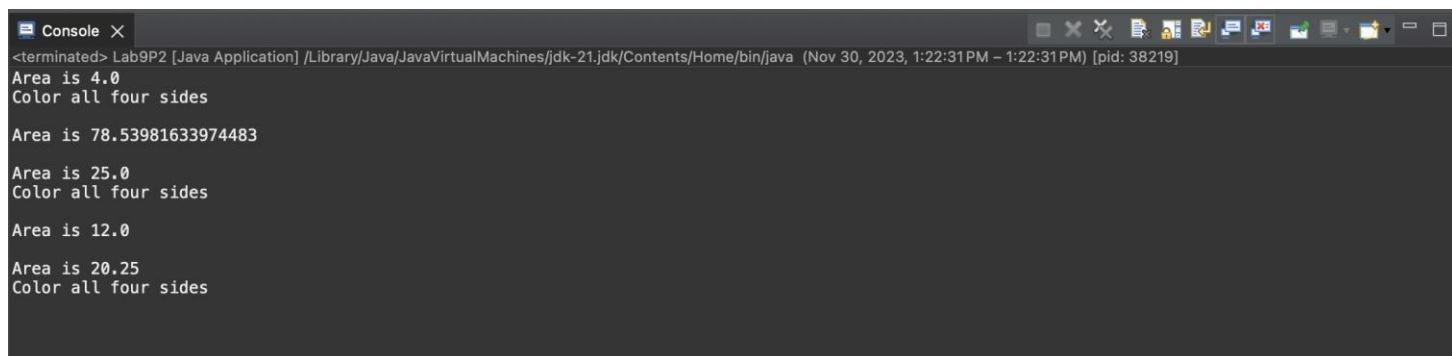
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        return width * height;
    }

    @Override
    public double getPerimeter() {
        return 2 * (width + height);
    }
}

```

Screenshots of sample runs –



The screenshot shows a Java IDE console window with the following output:

```

<terminated> Lab9P2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Nov 30, 2023, 1:22:31 PM – 1:22:31 PM) [pid: 38219]
Area is 4.0
Color all four sides

Area is 78.53981633974483

Area is 25.0
Color all four sides

Area is 12.0

Area is 20.25
Color all four sides

```