Vikas Meneni **Lab 9 – Assignment 5** **NUID:002309537**

1.Problem 1

Problem Description:

The problem requires creating a class named Octagon to model a regular octagon with equal side lengths. This class extends an abstract class GeometricObject, which defines methods for calculating area and perimeter, and implements the Comparable and Cloneable interfaces. The area is calculated using the formula Area=(2+4/2)×side2Area=(2+4/2)×side2, and the perimeter as 8×side8×side. A test program must create an Octagon object with a side length of 9, display its area and perimeter, clone the object using the clone() method, and compare the original and cloned objects using the compareTo() method. The implementation demonstrates inheritance, polymorphism, and interface usage in Java. It highlights how to compute geometric properties, compare objects, and duplicate them through cloning. The task emphasizes designing modular and reusable code, applying these skills to broader object-oriented design problems.

Analysis

1.Class Structure:

The Octagon class extends GeometricObject to inherit geometric properties.

Implements the Comparable interface, allowing for comparison of octagons based on their side lengths.

Implements the Cloneable interface to support creating copies of an Octagon object.

2.Key Methods:

getArea: Calculates the area of the octagon using the formula area=(2+42)×side2area=(2+24)×side2.

getPerimeter: Computes the perimeter by multiplying the side length by 8.

clone: Implements the clone method from the Cloneable interface to generate a duplicate of the Octagon.

compareTo: Compares two Octagon objects based on their side lengths.

3.Test Program:

An Octagon object is created with a side length of 9.

The area and perimeter are displayed.

The object is cloned, and the original is compared with the clone using the compareTo method.

Challenges Faced

1.Inheritance and Interfaces Interaction: Gaining a clear understanding of how inheritance and interfaces interact within Java was a challenge. Specifically, ensuring that both the Comparable and Cloneable interfaces were correctly implemented.

2.Implementing the clone Method: Implementing the clone method proved difficult due to the need to properly handle the CloneNotSupportedException.

Possible Enhancements

1.Additional Methods: The Octagon class could be extended to include methods for calculating diagonal lengths or performing transformations (such as rotation or scaling).

2.Improved Precision: If more precision is needed in the area calculations, Java's BigDecimal could be utilized to avoid potential inaccuracies from floating-point arithmetic.

Through this exercise, the concepts of code reusability and extensibility are demonstrated. Similar geometric computations for other polygons, such as pentagons or hexagons, can be developed using a similar approach with minimal changes.

<u>Source code:</u>

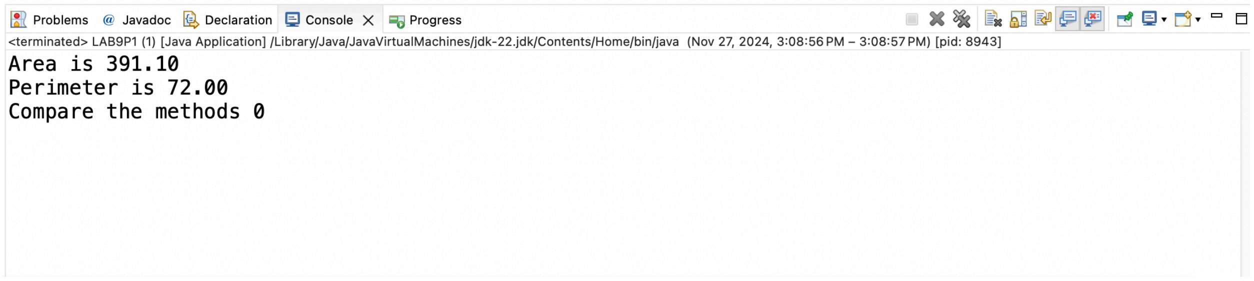
```
package edu.northeastern.csye6200;

class Octagon extends GeometricObject implements Comparable<Octagon>,
Cloneable {

  private double side;



  public Octagon(double side) {

    this.side = side;

  }
```

```
public double getSide() {
   return side;
}



public void setSide(double side) {
   this.side = side;
}

@Override
public int compareTo(Octagon o) {
   return Double.compare(this.side, o.side);
}

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

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

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

public class LAB9P1 {
  public static void main(String[] args) {
    Octagon octagon1 = new Octagon(9);

    System.out.printf("Area is %.2f\n", octagon1.getArea());
    System.out.printf("Perimeter is %.2f\n", octagon1.getPerimeter());

    Octagon octagon2 = null;
    try {
      octagon2 = (Octagon) octagon1.clone();
    } catch (CloneNotSupportedException e) {
      e.printStackTrace();
    }



    System.out.println("Compare the methods " + octagon1.compareTo(octagon2));
  }
}
```

Screenshots of sample runs:

Screenshot :-



```
<terminated> LAB9P1 (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java (Nov 27, 2024, 3:08:56 PM – 3:08:57 PM) [pid: 8943]
Area is 391.10
Perimeter is 72.00
Compare the methods 0
```

2.Problem 2

Problem Description:

The task involves designing an interface named Colorable with a method howToColor(), which specifies instructions for coloring a geometric object. A class named Square must be created that extends GeometricObject and implements the Colorable interface, with the howToColor() method displaying the message: *"Color all four sides."* The test program requires creating an array of five GeometricObject instances, including Square, Circle, and Rectangle objects. For each object in the array, the program must display its area and invoke the howToColor() method if the object is colorable. This problem highlights the use of interfaces to enforce consistent behavior across classes, while demonstrating polymorphism by handling diverse geometric shapes in a unified manner. It underscores the importance of designing reusable and extendable code, a fundamental principle in software development.

Analysis:

1.Interface Definition:

The problem requires designing a Colorable interface with a method void howToColor().

The interface is implemented by classes representing objects that can be colored.

2.Class Design:

GeometricObject: A base class that provides shared functionality (e.g., calculating area) for all geometric shapes.

Square:

Inherits from GeometricObject.

Implements the Colorable interface.

Overrides the howToColor() method to display the specific instruction, "Color all four sides."

Other shapes (e.g., Circle, Rectangle):

Inherit from GeometricObject.

Do not implement the Colorable interface.

3.Test Program:

A test program creates an array of five GeometricObject instances:

new Square(2), new Circle(5), new Square(5), new Rectangle(3, 4), and new Square(4.5).

The program iterates through the array and:

Displays the area of each object using the getArea() method.

Checks if the object is colorable using instanceof. If true, it invokes the howToColor() method.

Difficulties Encountered:

1.Dynamic Type Checking:

Using instanceof to determine if an object implements the Colorable interface can be error-prone in cases where the array contains unexpected types. This can lead to runtime errors or overly complex type-checking logic.

2.Interface Implementation:

Ensuring that all colorable objects provide meaningful and consistent behavior in the howToColor() method can become challenging, especially when dealing with more complex or varied geometric shapes.

Potential Improvements

1.Default Implementation in Interfaces:

Introduce a default implementation of the howToColor() method in the Colorable interface (available in Java 8+). This eliminates code duplication for objects with similar coloring behavior and simplifies implementation for future classes.

2.Scalability with Collections:

Replace the array with a List<GeometricObject> to handle a dynamic number of objects. Using collections allows for easier modifications, additions, or deletions of objects, making the solution more flexible and scalable.

Source Code:

1)LAB9P2.java

```
package edu.northeastern.csye6200;


public class LAB9P2 {

  public static void main(String[] args) {


    GeometricObject[] objects = {

      new Square(2),

      new Circle(5),

      new Square(5),

      new Rectangle(3, 4),

      new Square(4.5)

    };
```

```
        for (GeometricObject obj : objects) {
            System.out.printf("Area is %.2f%n", obj.getArea());



            if (obj instanceof Colorable) {
                ((Colorable) obj).howToColor();
            }
        }
    }
}



interface Colorable {
    void howToColor();
}

class Square extends GeometricObject implements Colorable {

    private double side;



    public Square(double side) {
        this.side = side;
    }
    public double getSide() {
        return 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");
    }
}
```

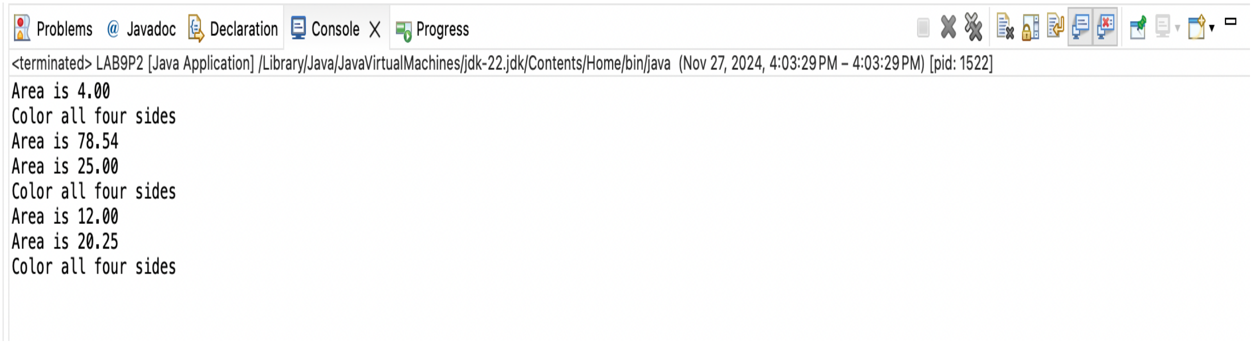
2)Rectangle.java

```
package edu.northeastern.csye6200;


public class Rectangle extends GeometricObject {
```

```
private double width;

private double height;



public Rectangle(double width, double height) {

   super();

   this.width = width;

   this.height = height;

}



public double getWidth() {

   return width;

}



public double getHeight() {

   return height;

}



@Override

public double getArea() {

   return width * height;

}
```

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

    @Override
    public String toString() {
        return String.format("Rectangle with width %.2f and height %.2f", width, height);
    }
}
```

<u>Screenshots of sample runs:</u>



Problems @ Javadoc Declaration Console Progress

```
<terminated> LAB9P2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java (Nov 27, 2024, 4:03:29 PM – 4:03:29 PM) [pid: 1522]
Area is 4.00
Color all four sides
Area is 78.54
Area is 25.00
Color all four sides
Area is 12.00
Area is 20.25
Color all four sides
```

<u>3.Problem 3:</u>

<u>Source Code:</u>

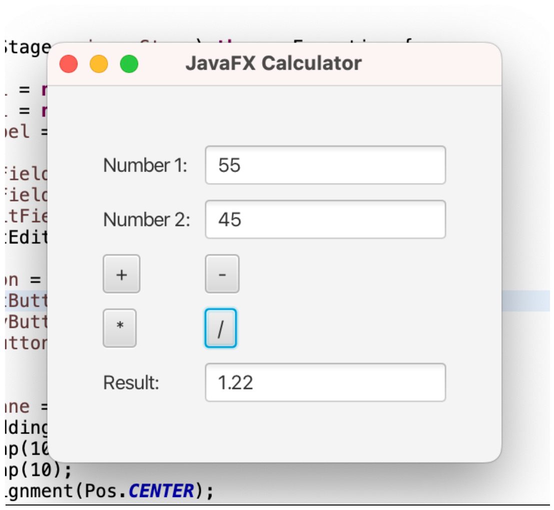
```
package edu.northeastern.csye6200;
```

```
import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.*;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;


public class LAB9P3 extends Application {


  @Override

  public void start(Stage primaryStage) throws Exception {


    Label num1Label = new Label("Number 1:");

    Label num2Label = new Label("Number 2:");

    Label resultLabel = new Label("Result:");


    TextField num1Field = new TextField();

    TextField num2Field = new TextField();

    TextField resultField = new TextField();

    resultField.setEditable(false);


    Button addButton = new Button("+");

    Button subtractButton = new Button("-");

    Button multiplyButton = new Button("*");

    Button divideButton = new Button("/");
```

```
GridPane gridPane = new GridPane();

gridPane.setPadding(new Insets(10));

gridPane.setHgap(10);

gridPane.setVgap(10);

gridPane.setAlignment(Pos.CENTER);



gridPane.add(num1Label, 0, 0);

gridPane.add(num1Field, 1, 0);

gridPane.add(num2Label, 0, 1);

gridPane.add(num2Field, 1, 1);

gridPane.add(addButton, 0, 2);

gridPane.add(subtractButton, 1, 2);

gridPane.add(multiplyButton, 0, 3);

gridPane.add(divideButton, 1, 3);

gridPane.add(resultLabel, 0, 4);

gridPane.add(resultField, 1, 4);



addButton.setOnAction(e -> performCalculation(num1Field, num2Field, resultField, "+"));

subtractButton.setOnAction(e -> performCalculation(num1Field, num2Field, resultField, "-"));

multiplyButton.setOnAction(e -> performCalculation(num1Field, num2Field, resultField, "*"));
```

```
        divideButton.setOnAction(e -> performCalculation(num1Field, num2Field,
resultField, "/"));



        Scene scene = new Scene(gridPane, 300, 250);

        primaryStage.setTitle("JavaFX Calculator");

        primaryStage.setScene(scene);

        primaryStage.show();

    }


    private void performCalculation(TextField num1Field, TextField num2Field, TextField
resultField, String operator) {


        if (num1Field.getText().isEmpty() || num2Field.getText().isEmpty()) {

            resultField.setText("Enter both numbers");

            return;

        }



        Double num1 = parseInput(num1Field.getText(), resultField);

        Double num2 = parseInput(num2Field.getText(), resultField);


        if (num1 == null || num2 == null) {

            return;

        }


        double result = 0;
    switch (operator) {
      case "+":
        result = num1 + num2;
        break;
      case "-":
        result = num1 - num2;
        break;
      case "*":
        result = num1 * num2;
        break;
      case "/":
        if (num2 == 0) {
          resultField.setText("Cannot divide by zero");
          return;
        }
        result = num1 / num2;
        break;
    }


    resultField.setText(String.format("%.2f", result));
  }

  private Double parseInput(String input, TextField resultField) {
        try {
            return Double.parseDouble(input);
        } catch (NumberFormatException e) {
            resultField.setText("Invalid input");
            return null;
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

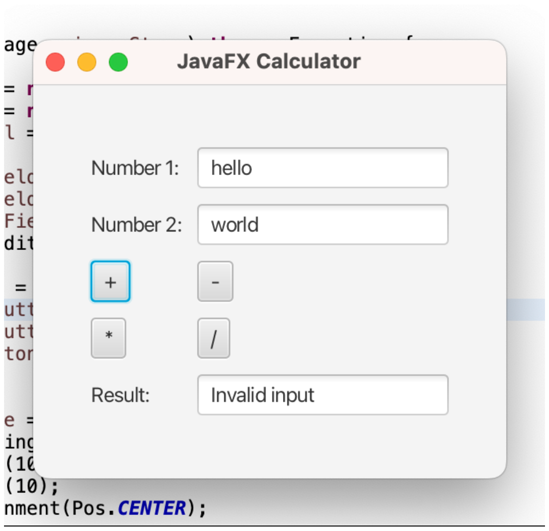
<u>**Screen shots of Sample Runs:-**</u>



Number 1: 55

Number 2: 45

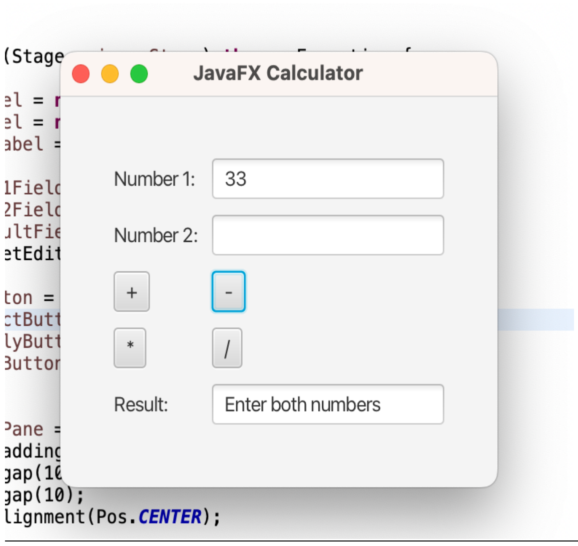
Result: 1.22



Number 1: hello

Number 2: world

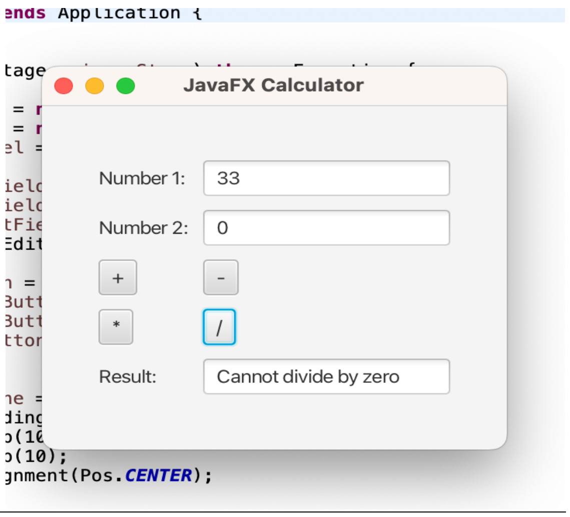
Result: Invalid input



Number 1: 33

Number 2:

Result: Enter both numbers



Number 1: 33

Number 2: 0

Result: Cannot divide by zero