Mingzhi Xu

CSC 22100 Software Design Laboratory Fall 2018

Exercise 1

In this exercise, we are to create a hierarchy of Java classes of MyShape, MyLine, MyPolygon, and MyCircle. The class MyShape is the hierarchy's superclass and inherits Java class Object. An implementation of the class defines a point of x and y and the color of the shape. The class includes appropriate class constructors that takes point x, y and color as parameters and methods that perform the following operations: getX, getY, getColor which returns the point x, y and color of the MyShape object, getX and getY can be simply just return the x and y instance variable of the object but there was some problem when implementing the getColor method, the problem was that javafx.scene.paint.Color and java.awt.Color cannot convert to each other and when I try to import javafx.scene.paint.Color, the class seem to ignore and not use java.awt.Color, so the solution was to declare the method as javafx.scene.paint.Color type and in the body we will convert our java.awt.color to javafx.scene.paint.Color using the rgb function that belongs to the javafx.scene.paint.Color class. Next, we will have the setX, setY, and setColor methods which simply just over writes the instance variable of the object, the same goes for shiftXY which just adds to the instance variable of x and y. Then we have toString method which returns a description of the object as a String and this method will be override in the subclasses, so they will return a different description of the object. Lastly, we have the draw method which will draw the object onto the canvas by using graphic context, it will also be override by subclasses since we have different ways of drawing different shapes, and in this case the draw method in MyShape will simply change the color of the canvas.

Code for MyShape class

```
public class MyShape {
    private double x, y;
    private Color color;
    public MyShape(double x, double y, Color color){
        this.x = x;
        this.y = y;
        this.color = color;
    }
    public double getX(){ return this.x; }
    public double getY(){ return this.y; }
    public javafx.scene.paint.Color getColor(){
        java.awt.Color awtColor = this.color;
        int r = awtColor.getRed();
        int g = awtColor.getGreen();
        int b = awtColor.getBlue();
        int a = awtColor.getAlpha();
        double opacity = a / 255.0 ;
        javafx.scene.paint.Color fxColor = javafx.scene.paint.Color.rgb(r, g, b, opacity);
        return fxColor;
    }
    public void setX(double x){ this.x = x; }
    public void setY(double y){ this.y = y; }
    public void setColor(Color color){ this.color = color; }
    public void shiftXY(double x, double y){
        this.x += x;
        this.y += y;
    }
    public String toString(){
        return "The coordinate of X and Y is (" + getX() + "," + getY() + ") and the color is " +
getColor();
    }
    public void draw(GraphicsContext gc){
        gc.setFill(getColor());
        gc.fillRect(0, 0, this.x, this.y);
    }
}
```

Next, we will implement MyLine class which inherits class MyShape and MyLine object is a straight line defined by its two endpoints (x1, y1) and (x2, y2). The line can be any color which can be obtain by using the superclass methods. The class includes appropriate class constructors that takes point x1, x2, y1, y2, and color as parameter and methods that perform the following operations: getLength returns the length of the MyLine object can simply be implemented using the mathematical formula of finding the distance of two points which is the

square root of (square (x2 – x1) + square (y2 – y1)). Next method would be get_xAngle which returns the angle in degrees of the object with the x-axis and in order to do that we need to find the arctan of (y2 – y1) / (x2 – x1) and convert radian to degrees using the Math.toDegrees. Then we have the toString method which overrides from the superclass with new description of MyLine objects. Lastly, we will need to override the draw method to draw MyLine objects which is done by using graphics context methods strokeLine which takes two end points (x1, y1) and (x2, y2) and draw a line, we can also set the color of the line by returning color using superclass method.

Code for MyLine class

```
public class MyLine extends MyShape {
    private double x2, y2;
    public MyLine(double x1, double x2, double y1, double y2, Color color) {
        super(x1, y1,color);
        this.x2 = x2;
        this.y2 = y2;
    }
    public double getLength() {
        double xsq = Math.pow(this.x2 - getX(), 2);
        double ysq = Math.pow(this.y2 - getY(), 2);
        double length = Math.sqrt(xsq + ysq);
        return length;
    }
    public double get_xAngle() {
        double xAngle = Math.toDegrees(Math.atan((this.y2 - getY())/(this.x2 - getX())));
        return xAngle;
    }
    @Override
    public String toString() {
        return "The length of this line is " + this.getLength() + " with an angle relate to the x-axis
of "
                + this.get_xAngle() + " degrees";
    }
    @Override
    public void draw(GraphicsContext gc){
        gc.setStroke(super.getColor());
        gc.setLineWidth(1);
        gc.strokeLine(super.getX(), super.getY(),this.x2, this.y2);
    }
}
```

Then, we will implement MyCircle class which inherits class MyShape and MyCircle is defined by its radius, and center point x, y and color. The class includes appropriate class constructors that takes center point x and y, radius and color as parameters and methods that perform the following operations: getRadius which simply return the instance variable of radius. getArea, which can be found using the formula radius square times Pi. getParameter which is just finding the circumference of a circle and can be implemented using the formula 2 times radius times Pi. Then we override the toString method that returns a string that shows the description of a MyCircle object. Lastly, we override the draw method and by using the graphics context method strokeOval and fillOval we can draw a circle with declared center point and its radius, we can also set the color by using methods from the superclass.

Code for MyCircle class

```
public class MyCircle extends MyShape {
    private double radius;
    public MyCircle(double x, double y, double radius, Color color) {
        super(x,y,color);
        this.radius = radius;
    }
    public double getRadius() {
        return radius;
    }
    public double getArea() {
        return (Math.PI)* Math.pow(getRadius(),2);
    }
    public double getPerimeter() {
        return (Math.PI)*(2*getRadius());
    }
    @Override
    public String toString() {
        return "The circle has a radius of " + this.getRadius() + " with a perimeter of " +
this.getPerimeter() +
                " and an area of " + this.getArea();
    }
    @Override
    public void draw(GraphicsContext gc) {
        gc.setFill(super.getColor());
        gc.strokeOval(super.getX(), super.getY(),getRadius(), getRadius());
        gc.fillOval(super.getX(), super.getY(),getRadius(), getRadius());
    }
}
```

Lastly, we will implement MyPolygon class which inherits the class MyShape and MyPolygon is defined by the integer parameter N which is the number of the polygon's equal side lengths and equal interior angles. The class includes appropriate class constructors that takes x, y, N, side length, and color as parameters and methods that perform the following operations: getSide, which returns the side length of the MyPolygon object. getPerimeter, returns perimeter which is just number of side lengths, N times side length. getArea which can be found using by squaring side length times N / 4 times tan of 180 / N. getAngle returns the interior angle of the MyPolygon object and can be found by 180 * (N – 2) / N. Then we override the toString method with new descriptions of the MyPolygon object. Lastly, we override the draw method and by using the graphics context method strokePolygon and fillPolygon we can draw and fill the color of a polygon, and to do that we need to provide two arrays of points that the graphics context method takes as parameters. Since we will be drawing polygons with equal side lengths, we can use a loop to determine the x, y vertices based on given N and interior angle.

Code for MyPolygon class

```
public class MyPolygon extends MyShape{
    private int n;
    private double sideLength;
    public MyPolygon(double x, double y, int n, double sideLength, Color color) {
        super(x,y, color);
        this.n = n;
        this.sideLength = sideLength;
    }
    public double getArea() {
        double getRadians = Math.toRadians(180/this.n);
        return Math.pow(getSide(),2) * this.n/4 * (Math.tan(getRadians));
    }
    public double getPerimeter() {
        return this.n * getSide();
    }
    public double getAngle() {
        return 180*(this.n - 2)/ this.n;
    }
    public double getSide() {
        return sideLength;
    }
```

```
    public String toString() {
        return "This polygon has side length of " + getSide() + " , an interior angle of " + getAngle()
+
                " , a perimeter of " + getPerimeter() + " and an area of " + getArea();
    }
    public void draw(GraphicsContext gc){
        gc.setFill(super.getColor());
        double[] x_vertices = new double[this.n];
        double[] y_vertices = new double[this.n];
        double angle = (this.n - 1) * getAngle();
        double angle_increment = (2*Math.PI)/this.n;
        int i;
        for (i = 0; i < this.n; i++) {
            x_vertices[i] =  (int)  ((getSide()*Math.cos(angle)) + super.getX());
            y_vertices[i] = (int) ((getSide()*Math.sin(angle)) + super.getY());
            angle += angle_increment;
        }
        gc.strokePolygon(x_vertices, y_vertices, this.n);
        gc.fillPolygon(x_vertices, y_vertices, this.n);
    }
}
```

Finally, we will implement the Main class to declare objects of those classes we have created and draw the objects onto the canvas. First, we will have the class Main to extend Application then we will have method main which contains launch(args) to pass command lines to javafx.application.Application and open Javafx. The start method is where we will have our stage, scene and objects declared, in this part we will declare a stack pane, scene with appropriate size, canvas with the same size as scene and graphics context which acts as the brush to paint our objects. Now we can declare objects of the class we have created and call on the draw method and then adding canvas to the pane and showing the scene will allow the program to display our result.

Code for Main class

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Scene scene = new Scene(pane, 800, 500);
```

```
        Canvas canvas = new Canvas(800,500);
        GraphicsContext gc = canvas.getGraphicsContext2D();

        MyShape shape1 = new MyShape(800, 500,  Color.black);//actual assignment
        shape1.draw(gc);
        MyCircle circle1 = new MyCircle(150, 0, 500, Color.white);
        circle1.draw(gc);
        MyPolygon polygon1 = new MyPolygon(400,250, 5, 250, Color.black);
        polygon1.draw(gc);
        MyCircle circle2 = new MyCircle(200, 50, 400, Color.red);
        circle2.draw(gc);
        MyPolygon polygon2 = new MyPolygon(400,250, 5, 200, Color.black);
        polygon2.draw(gc);
        MyCircle circle3 = new MyCircle(250, 100, 300, Color.white);
        circle3.draw(gc);
        MyPolygon polygon3 = new MyPolygon(400,250, 5, 150, Color.black);
        polygon3.draw(gc);
        MyLine line1 = new MyLine(0, 800, 0, 500, Color.green);
        line1.draw(gc);
        MyLine line2 = new MyLine(800, 0, 0, 500, Color.green);
        line2.draw(gc);

        pane.getChildren().add(canvas);
        primaryStage.setTitle("MyShape");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Result after running