Monday, September 26, 2022

SimplePaintObject - Writeup

---

this assignment was challenging at first but as I progressed through it, it became clearer and clearer. Honestly I personally think the first assignment was more complicated for me because I hadn't coded in Java in a while, which made it a little difficult to understand what exactly was going on in the code. *The additional code requirements hindered my solution to this problem* at first by having to learn how the layout is going to work. Along with the layout, the class diagram was a little confusing at first but once I learned what each arrow meant, it made the code a lot easier. *Some advantages to these code requirements going forward* is one, having a clear idea of what needs to be done, and two, it taught me how to work smarter by having working code as often as possible. *I personally do not think the required class hierarchy gets in the way of code reuse.* I found that the class hierarchy made it easier to implement code.

APPENDIX

```
import java.util.ArrayList;
import javafx.application.Application;
import javafx.geometry.Point2D;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.shape.Ellipse;
import javafx.scene.shape.Line;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;
import javafx.scene.paint.Color;

/*-----------------------ABSTRACT-TOOLS CLASS---------------------------------- */

/* This makes the rectangles to hold the tools and and Colors*/
abstract class AbstractTool extends StackPane {
   Rectangle rectangle;
```

```
    public AbstractTool(Color color) {
        rectangle = new Rectangle();
        rectangle.setWidth(60);
        rectangle.setHeight(60);
        rectangle.setOpacity(0.4);
        rectangle.setFill(color);
        rectangle.setStroke(Color.WHITE);
        this.getChildren().add(rectangle);
    }

    public void activate() {
        rectangle.setOpacity(1.0);
    }

    public void deactivate() {
        rectangle.setOpacity(0.4);
    }

}

/*
 * ^----------END OFABSTRACT-TOOLSCLASS----------^
 */

/*
 * v----------COLOR-TOOLSCLASS----------v
 */

/* This sets the color for the color tools */
class ColorTool extends AbstractTool {
```

```
    private Color color;

    // Constructor
    public ColorTool(Color color) {
        super(color);
        this.color = color;
        this.setOnMousePressed(e -> activate());
    }

    public Color getColor() {
        return this.color;
    }
}

/*----------------------END OF COLOR-TOOLS CLASS---------------------------- */

/*------------------------ACTION-TOOLS CLASS---------------------------- */

/* ACTION-TOOL CLASS */
class ActionTool extends AbstractTool {

    // Constructor
    public ActionTool(Color color) {
        super(color);

    }

    // Constructor with Runnable function to clear the canvas
    public ActionTool(String text, Runnable action) {
```

```
        super(Color.LIGHTCORAL);
        Label cmdName = new Label(text);
        cmdName.setTextFill(Color.LEMONCHIFFON);
        cmdName.setFont(Font.font("Verdana", FontWeight.BOLD, 14));
        this.getChildren().add(cmdName);
        setOnMousePressed(e -> {
            this.activate();
        });
        setOnMouseReleased(e -> {
            this.deactivate();
            action.run();
        });

    }
}
/* END OF ACTION-TOOL CLASS */

/*----------------------END OF ACTION-TOOLS CLASS---------------------------- */

/*---------------------------SHAPE-TOOL CLASS--------------------------- */

abstract class ShapeTool extends AbstractTool {
    public ShapeTool(Color color) {
        super(color);
        this.setOnMouseClicked(e -> activate());
    }

    // Returns the shape object for the tools
    abstract public ShapeObject getPaintShape();
```

```
    abstract public void draw(GraphicsContext g, Color color, Point2D start, Point2D end);
}
/* END OF SHAPE-TOOL CLASS */

/*--------------------------POINT-TOOL CLASS------------------------------ */

class PointTool extends ShapeTool {
    private int penWidth;
    lineSegmentShape LineShape; // Default instance

    Color color;
    Point2D start;
    Point2D end;

    public PointTool(int penWidth) {
        super(Color.LIGHTCORAL);
        this.penWidth = penWidth;
        LineShape = new lineSegmentShape(penWidth, color, new Point2D(0, 0), new
Point2D(0, 0)); // default
        Ellipse toolImage = new Ellipse(penWidth, penWidth);
        toolImage.setStroke(Color.LEMONCHIFFON);
        toolImage.setFill(Color.LEMONCHIFFON);

        this.getChildren().add(toolImage);
    }

    public int getPenWidth() {
        return penWidth;
    }
```

```
    @Override
    public void draw(GraphicsContext g, Color color, Point2D start, Point2D end) {
        LineShape = new lineSegmentShape(penWidth, color, start, end);
    }

    @Override
    public ShapeObject getPaintShape() {
        return LineShape;
    }

}

/*-------------------------END OF POINT-TOOL CLASS---------------------------- */

/*-----------------------------LINE-TOOL CLASS---------------------------- */

class LineTool extends ShapeTool {

    LineShape lineShape;
    Point2D start, end;
    Color color;

    public LineTool(double initX, double initY, double x, double y) {
        super(Color.LIGHTCORAL);
        Line lineImage = new Line(initX, initY, x, y);
        lineShape = new LineShape(new Point2D(0, 0), new Point2D(0, 0), color);
        lineImage.setStroke(Color.LEMONCHIFFON);
        lineImage.setStrokeWidth(2);

        this.getChildren().add(lineImage);
```

```
    }

    @Override
    public void draw(GraphicsContext g, Color color, Point2D start, Point2D end) {
        lineShape = new LineShape(start, end, color);
        // lineShape.draw(g);
    }

    @Override
    public ShapeObject getPaintShape() {
        return lineShape;
    }

}

/*----------------------------END OF LINE-TOOL CLASS---------------------------- */

/*-----------------------------RECTANGLE-TOOL CLASS---------------------------- */

class Rectangletool extends ShapeTool {
    RectangleShape rectangleShape = new RectangleShape(new Point2D(0, 0), new
Point2D(0, 0), Color.BEIGE);

    public Rectangletool(int width, int height) {
        super(Color.LIGHTCORAL);
        Rectangle rect = new Rectangle(width, height);
        rect.setFill(Color.LEMONCHIFFON);

        this.getChildren().add(rect);
    }
```

```
    @Override
    public void draw(GraphicsContext g, Color color, Point2D start, Point2D end) {
        rectangleShape = new RectangleShape(start, end, color);
    }

    @Override
    public ShapeObject getPaintShape() {
        // TODO Auto-generated method stub
        return rectangleShape;
    }
}

/*------------------------------END OF RECTANGLE-TOOL CLASS---------------------------- */

/*--------------------------------OVAL-TOOL CLASS---------------------------------- */

class OvalTool extends ShapeTool {
    OvalShape goatCircle = new OvalShape(new Point2D(0, 0), new Point2D(0, 0),
Color.BEIGE);

    public OvalTool(int width, int height) {
        super(Color.LIGHTCORAL);
        Ellipse circle = new Ellipse(width, height);
        circle.setFill(Color.LEMONCHIFFON);

        this.getChildren().add(circle);
    }

    @Override
```

```
    public void draw(GraphicsContext g, Color color, Point2D start, Point2D end) {
        goatCircle = new OvalShape(start, end, color);
    }

    @Override
    public ShapeObject getPaintShape() {
        return goatCircle;
    }

}

/*-----------------------------END OF OVAL-TOOL CLASS----------------------------*/

/*-----------------------------ROUNDED-RECT-TOOL-CLASS------------------------------ */

class RoundedRectangleTool extends ShapeTool {
    RoundedRectangleShape roundRect = new RoundedRectangleShape(new Point2D(0,
0), new Point2D(0, 0), Color.BEIGE);

    public RoundedRectangleTool(int width, int height) {
        super(Color.LIGHTCORAL);
        Rectangle roundedRect = new Rectangle(width, height);
        roundedRect.setArcWidth((width / 2) / 2); // Makes the corner rounded
        roundedRect.setArcHeight((width / 2) / 2); // Makes the corner rounded
        roundedRect.setFill(Color.LEMONCHIFFON);

        this.getChildren().add(roundedRect);
    }

    @Override
```

```
    public void draw(GraphicsContext g, Color color, Point2D start, Point2D end) {
        roundRect = new RoundedRectangleShape(start, end, color);

    }

    @Override
    public ShapeObject getPaintShape() {
        return roundRect;
    }
}

/*-----------------------------END OF ROUNDED-RECT-TOOL CLASS---------------------------- */

/*-----------------------------RECTANGLE-SHAPE CLASS---------------------------------- */

class RectangleShape extends FilledPolyShape {

    public RectangleShape(Point2D start, Point2D end, Color color) {
        super(start, end, color);

    }

    @Override
    public void draw(GraphicsContext g) {
        g.setFill(this.color);
        g.fillRect(offsetX, offsetY, growX, growY);

    }

}
```

```
/*-----------------------------END OF RECTANGLE-SHAPE CLASS---------------------------- */

/*---------------------------------OVAL-SHAPE CLASS-------------------------------- */

class OvalShape extends FilledPolyShape {

    public OvalShape(Point2D start, Point2D end, Color color) {
        super(start, end, color);

    }

    @Override
    public void draw(GraphicsContext g) {
        g.setFill(this.color);
        g.fillOval(offsetX, offsetY, growX, growY);

    }

}

/*-----------------------------END OF OVAL-SHAPE CLASS---------------------------- */

/*--------------------------ROUNDED-RECT-SHAPE CLASS---------------------------- */

class RoundedRectangleShape extends FilledPolyShape {

    public RoundedRectangleShape(Point2D start, Point2D end, Color color) {
        super(start, end, color);
```

```
    }

    @Override
    public void draw(GraphicsContext g) {
        g.setFill(this.color);
        g.fillRoundRect(offsetX, offsetY, growX, growY, 25, 25);

    }

}

/*------------------------------END OF ROUNDED-RECT-SHAPE CLASS---------------------------- */

/*--------------------------------SHAPE OBJECT INTERFACE--------------------------------- */

interface ShapeObject {
    void draw(GraphicsContext g);

    Boolean dragUpdate();
}

/*--------------------------------END OF SHAPE OBJECT INTERFACE--------------------------- */

/*----------------------------------FILLED-POLY-SHAPE CLASS------------------------------ */

abstract class FilledPolyShape implements ShapeObject {

    public double initX, initY, x, y;
    public double offsetX, offsetY, growX, growY;
    public Color color;
```

```
    public FilledPolyShape(Point2D start, Point2D end, Color color) {
        this.color = color;
        initX = start.getX();
        initY = start.getY();
        x = end.getX();
        y = end.getY();
        offsetX = initX - (x - initX);
        offsetY = initY - (y - initY);
        growX = (x - initX) * 2;
        growY = (y - initY) * 2;
    }

    @Override
    public Boolean dragUpdate() {
        return false;

    }
}

/*------------------------------END OF FILLED POLY SHAPE CLASS--------------------------- */

/*----------------------------------LINE-SHAPE CLASS---------------------------------- */

class LineShape implements ShapeObject {

    private Point2D start, end;
    private Color color;

    public LineShape(Point2D start, Point2D end, Color color) {
```

```
        this.start = start;
        this.end = end;
        this.color = color;
    }

    @Override
    public void draw(GraphicsContext g) {
        g.setStroke(color);
        g.setFill(color);
        g.setLineWidth(2);
        g.strokeLine(start.getX(), start.getY(), end.getX(), end.getY());
    }

    @Override
    public Boolean dragUpdate() {
        return false;

    }
}

/*---------------------------------END OF LINE-SHAPE CLASS-------------------------------- */

/*---------------------------------LINE-SHAPE CLASS-------------------------------- */

class lineSegmentShape implements ShapeObject {

    private double width;
    private Color color;
    private Point2D start, end;
```

```
    public lineSegmentShape(double width, Color color, Point2D start, Point2D end) {
        this.width = width;
        this.color = color;
        this.start = start;
        this.end = end;
    }

    @Override
    public void draw(GraphicsContext g) {
        g.setStroke(color);
        g.setFill(color);
        g.setLineWidth(width);
        g.strokeLine(start.getX(), start.getY(), end.getX(), end.getY());
    }

    @Override
    public Boolean dragUpdate() {
        return true;

    }
}

/*--------------------------------ENF OF LINE-SHAPE CLASS-------------------------------- */

/*---------------------------------------MAIN CLASS---------------------------------------- */

public class SimplePaintObjects extends Application {

    static ColorTool currentColor;
    static ShapeTool currentTool;
```

```
    double initX, initY;
    Boolean dragging = false;
    ShapeObject currentShapeObject;

    private ArrayList<ShapeObject> shapes = new ArrayList<>(); // ARRAY LIST OF SHAPE
OBJECTS

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

    // ROOT PANE FOR HOLDING THE CANVAS, TOOLBAR, AND COLOR BAR
    private Parent makeRootPane() {
        HBox root = new HBox();
        root.getChildren().add(makeCanvas());
        root.getChildren().add(makeToolPane());
        root.getChildren().add(makeColorPane());
        return root;
    }

    private final Color[] palette = {
            Color.BLACK, Color.RED, Color.GREEN, Color.BLUE,
            Color.CYAN, Color.MAGENTA, Color.YELLOW
    };

    /*
     * LOOPS OVER THE PALETTE ARRAY AND ADDS EACH COLOR AT INDEX I TO
     * THE COLORPANE VBOX. THIS CREATES THE BAR WHERE WE SELECT A
     * COLOR TO DRAW WITH
     */
```

```
    private Node makeColorPane() {
        VBox colorPane = new VBox();

        /* THE FOR LOOP TO ITERATE OVER ARRAY */
        for (int i = 0; i < 7; i++) {
            colorPane.getChildren().add(
                    addMouseHandlerToColorTool(new ColorTool(palette[i])));
        }

        /* SETS THE FIRST COLORS AS THE DEFAULT SELECTED COLOR WHEN APP FIRST
OPENS */
        currentColor = (ColorTool) colorPane.getChildren().get(0);

        currentColor.activate();

        /*
         * ADDS THE CLEAR BUTTON TO THE HBOX.
         * THIS IS DONE BY ADDING AN ACTION TOOL INSTANCE WITH THE TEXT CLEAR
AND A
         * REFERENCE TO THE CLEARCANVASANDSHAES FUNCTION TO THE HBOX. THIS
GIVES THE
         * BUTTON
         * FUNCTIONALITY WHEN CLICKED ON.
         */
        colorPane.getChildren().add(new ActionTool("Clear", this::clearCanvasAndShapes));

        return colorPane;
    }

    /*
```

```
    * CREATES THE TOOL PANE USING A NEW INSTANCE OF A CERTAIN TOOL TO THE
TOOLPANE
    * FIRST FOUR INCLUDE A POINT TOOL WITH DIFFERENT WIDTHS IN EACH ONE
FROM 2-6.
    * THE REST ARE CREATED USING THEIR RESPECRED TOOL CLASSES.
    */
   private Node makeToolPane() {
      VBox toolPane = new VBox();
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new PointTool(2)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new PointTool(4)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new PointTool(6)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new PointTool(8)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new LineTool(0.0f, 0.0f,
35.0, 35.0)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new Rectangletool(40,
40)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new OvalTool(20, 20)));
      toolPane.getChildren().add(addMouseHandlerToShapeTool(new
RoundedRectangleTool(40, 40)));

      /* SETS THE FIRST TOOL AS THE DEFAULT WHEN APP FIRST OPENS */
      currentTool = (ShapeTool) toolPane.getChildren().get(0);
      currentTool.activate();

      return toolPane;
   }

   /*
    * MOUSE HANDLER FOR COLORS.
    * FIRST: DEACTIVATES THE CURRENT COLOR
```

```
     * SECOND: SETS CURRENTCOLOR TO THE CLICKED COLOR
     * THIRD: ACTIVATES THE SELECTED COLOR.
     */
    private ColorTool addMouseHandlerToColorTool(ColorTool tool) {
        tool.setOnMousePressed((e) -> {
            this.currentColor.deactivate();
            this.currentColor = tool;
            tool.activate();
        });
        return tool;
    }

    /*
     * MOUSE HANDLER FOR TOOLS.
     * FIRST: DEACTIVATES THE CURRENT TOOL
     * SECOND: SETS CURRENTTOOL TO THE CLICKED TOOL
     * THIRD: ACTIVATES THE SELECTED TOOL.
     */
    private ShapeTool addMouseHandlerToShapeTool(ShapeTool tool) {
        tool.setOnMousePressed((e) -> {
            this.currentTool.deactivate();
            this.currentTool = tool;
            tool.activate();
        });
        return tool;
    }

    GraphicsContext g;

    /*
```

```
 * CREATES CANVAS AND SETS UP MOUSE EVENTS TO ACTUAL CANVAS WHERE
 * WE WILL BE DRAWING.
 */
private Node makeCanvas() {
    Canvas canvas = new Canvas(600, 500);
    g = canvas.getGraphicsContext2D();
    clearCanvas();
    canvas.setOnMousePressed(e -> mousePressed(e));
    canvas.setOnMouseDragged(e -> mouseDragged(e));
    canvas.setOnMouseReleased(e -> mouseReleased(e));
    return canvas;
}

/* sets the x and y variables to where we clicked */
private void mousePressed(MouseEvent e) {
    double x = e.getX();
    double y = e.getY();
    initX = x;
    initY = y;
    dragging = true;
}

/*
 * Draws whatevers is in the shapes list.
 * checks to see if we have selected a point tool or a non-point tool.
 */
private void mouseDragged(MouseEvent e) {
    clearCanvas();
    for (int i = 0; i < shapes.size(); i++) {
        shapes.get(i).draw(g);
```

```
        }
        double x = e.getX();
        double y = e.getY();

        currentShapeObject = currentTool.getPaintShape(); // Instantiates the shape object using get method

        /*
         * IF DRAGUPDATE IS TRUE, DO FIRST BLOCK. THE ONLY TOOL WITH DRAG UPDATE
         * SET TO TRUE ARE THE POINT TOOLS. IF IT'S A POINT TOOL, WE WANT TO CONSTANTLY
         * UPDATE THE INITIAL
         * X AND Y SO THAT THE LINE IS CONTINUOUSLY BEING DRAWN.
         */
        if (currentShapeObject.dragUpdate()) {
            currentTool.draw(g, currentColor.getColor(), new Point2D(initX, initY), new Point2D(x, y));
            shapes.add(currentShapeObject);
            initX = x;
            initY = y;
        } else {
            /*
             * ALLOWS US TO BE ABLE TO SEE WHAT IS GOING TO BE DRAWN BEFORE IT IS ACTUALLY
             * COMMITED TO THE CANVAS. THE CANVAS ONLY INCLUDES WHAT'S IN THE LIST.
             */
            currentTool.draw(g, currentColor.getColor(), new Point2D(initX, initY), new Point2D(x, y));
```

```
            currentShapeObject.draw(g);
        }

    }

    /*
     * COMMITS THE ABOVE CODE TO THE CANVAS AFTER WE RELEASE THE MOUSE.
ONCE WE
     * RELEASE
     * THE MOUSE, THE SHAPE WE DREW IS ACTUALLY ON THE CANVAS. THE CANVAS
REDRAWS
     * ITSELF WITH
     * WHATEVER IS IN THE LIST.
     */
    private void mouseReleased(MouseEvent e) {
        if (currentShapeObject.dragUpdate() == false) {
            /*
             * SINCE NON-POINT TOOLS HAVE DRAGUPDATE SET TO FALSE, WE ARE
             * COMMITING THESE SHAPES TO THE LIST BEFORE THEY ARE ACTUALLY
DRAWN TO
             * CANVAS.
             */
            shapes.add(currentShapeObject);
        }

    }

    /* SIMPLY CLEARS THE CANVAS */
    private void clearCanvas() {
        g.setFill(Color.WHITE);
```

```
        g.fillRect(0, 0, g.getCanvas().getWidth(), g.getCanvas().getHeight());

    }

    /*
     * CLEARS THE CANVAS AND THE SHAPES IN THE LIST.
     * BEFORE THIS METHOD I WAS HAVING ISSUES WITH THE LIST NOT
     * CLEARING PROPERLY AND HAVING SHAPES POP UP AGAIN WHEN DRAWING
AFTER CLEARING
     */
    private void clearCanvasAndShapes() {
        clearCanvas();
        shapes.clear();
    }

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.setScene(new Scene(makeRootPane()));
        primaryStage.setTitle("Vertical Slice Paint");
        primaryStage.setResizable(false);
        primaryStage.show();
    }
}
```