Simple Paint Objects Write-up

Write-up Questions

1) How did the additional code requirements hinder your solution to this problem?

No additional code requirements hindered my solution to this problem; for, a specified and well defined list of requirements are easy to target code against and behaviors that are outside of specification are considered "bugs".

The only issue I have with the homework prompt is that an approach to how one may approach the problem would be offered, but later abandoned in favor for an optimal, better, solution: a great tactic to lead students a stray if they read section by section coding to the requirements and to later find out what they had isn't optimal and won't be accepted for grading...

`NOTE TO SELF: READ THE ENTIRE HOMEWORK PROMPT BEFORE STARTING THE ASSIGNMENT.`

2) What are some advantages to these code requirements going forward?

As answered in question prompt one, having code requirements defines the acceptable state, it defines a scope of work where behaviors outside of what has been prescribed in the requirements documentation are "bugs", and (if well written) little should be left up to interpretation.

3) Where do you think the required class hierarchy gets in the way of code reuse?

> (To think about this problem, where did you find the hierarchy tedious in terms of typing code that is almost the same from one class to the next, e.g., the classes derived from FilledPolyShape)

As it stands now... With the current required class hierarchy that gets in the way of code reuse is by way of not having a generic class that will create the icons of each of the drawing tools and action (PointTool, LineTool, RectangleTool OvalTool, RoundedRectangleTool, and the Action "ClearTool").

Example: "Icons can be made generically"

ActionTool

> [!code] ActionTool - `makeActionToolTitleName()`

```
public ActionTool(String cmdName, Runnable action) {
   super(SimplePaintObjects.TOOL_RECT_FG);
   makeActionToolTitleName(cmdName); // Can be made generically
   this.action = action;
}

/**
* Creates a tool that is used to perform an action,
* such as clearing the canvas: the tool is a button with a label that
* says "Clear". The button is colored with the specified color.
* @param cmdName the text to be displayed on the button
*/
private void makeActionToolTitleName(String cmdName) {
   Label commandName = new Label(cmdName);
   commandName.setTextFill(SimplePaintObjects.TOOL_FG);
   commandName.setFont(Font.font(
           "Verdana",
           FontWeight.BOLD,
           20));
   this.getChildren().add(commandName);
}
```

PointTool

[!code] PointTool - `makePointToolIcon()`

```
public PointTool(int penWidth) {
   super(SimplePaintObjects.TOOL_RECT_FG);
   this.penWidth = penWidth;
   lineSegmentShape = new LineSegmentShape(
           new Point2D(0, 0),
           new Point2D(0, 0),
           Color.BLACK,
           penWidth);
   makePointToolIcon(penWidth); // Can be made generically
}

private void makePointToolIcon(int penWidth) {
   Ellipse toolIcon = new Ellipse(penWidth, penWidth);
   toolIcon.setStroke(SimplePaintObjects.TOOL_FG);
   toolIcon.setFill(SimplePaintObjects.TOOL_FG);
   this.getChildren().add(toolIcon);
}
```

Appendix

```
import java.util.ArrayList;
import javafx.application.Application;
```

```java
import javafx.geometry.Insets;
import javafx.geometry.Point2D;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundFill;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.Node;
import javafx.scene.paint.Color;
import javafx.scene.Parent;
import javafx.scene.Scene;
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;

/**
 * The class AbstractTool is, as you should expect, an abstract class,
 * i.e., a class which cannot be instantiated, that is the base class for other
 * tool classes. You will want to instantiate a javafx.scene.shape.Rectangle
 * object to represent the background of the tool and you will want the ability
 * to set this Rectangle's color. Since all tools can be both activated and
 * deactivated, you will want to implement methods here to activate and
 * deactivate the tool.
 * Note that this only involves changing the properties of the base Rectangle.
 */
abstract class AbstractTool extends StackPane {
    /**
     * (i.e., 20% larger than the size of the rectangle that is the background
     * of the tool icon)
     */
    private static final double scaleRectangle = 1.2;
    Rectangle rectangle;

    public AbstractTool(Color color) {
        rectangle = new Rectangle();
        rectangle.setWidth(SimplePaintObjects.CELL_W);
        rectangle.setHeight(SimplePaintObjects.CELL_H);
        rectangle.setFill(color);
        rectangle.setStroke(color);
        this.getChildren().add(rectangle);
    }

    public void activate() {
        rectangle.setWidth(SimplePaintObjects.CELL_W * scaleRectangle);
        rectangle.setHeight(SimplePaintObjects.CELL_W * scaleRectangle);
    }
    public void deactivate() {
        rectangle.setWidth(SimplePaintObjects.CELL_W);
        rectangle.setHeight(SimplePaintObjects.CELL_W);
    }
}

/**
 * ColorTool sets the color within the application. When a color activates, it
 * should deactivate any other ColorTool objects. There are a number of ways to
 * do this. One reasonably clean way is to store the active ColorTool object in
 * your main class. Thus, you only need to deactivate the current tool, set the
 * current tool to the new ColorTool object, then activate the new tool.
 * Note it is not necessary to also save the color that the tool represents,
 * rather, implement a getter for the color to be called, as necessary, on the
 * current color tool.
 */
class ColorTool extends AbstractTool {
    private Color color;

    public ColorTool(Color color) {
        super(color);
        this.color = color;
    }

    public Color getColor() {
        return color;
    }
}

/*
 * For this assignment there is only one ActionTool, namely, the clear button.
 * An ActionTool object performs an action when depressed. Since this action
 * may be a method outside of the Tool class hierarchy, you will need to pass
 * this action in as an object of type Runnable. This is much easier to do than
 * it sounds. Simply pass a method reference, e.g., this::myClearAction as the
 * parameter to the constructor. In your main class you will implement a method
 * myClearAction() that implements the clear function. We will discuss how this
 * works in more detail later in the course. In order to execute the action on
 * a mouse press, you will need to call the run() method on the Runnable object
 * that you have stored within your ActionTool.
 */
class ActionTool extends AbstractTool {
    private Runnable action;

    /**
     * Creates a tool that is used to perform an action, such as clearing the
     * canvas: the tool is a button with a label that says "Clear".
     *
     * @param cmdName the text to be displayed on the button
     * @param action  the action to be performed when the tool is activated
     */
    public ActionTool(String cmdName, Runnable action) {
        super(SimplePaintObjects.TOOL_RECT_FG);
        makeActionToolTitleName(cmdName);
        this.action = action;
    }

    /**
     * Creates a tool that is used to perform an action,
     * such as clearing the canvas: the tool is a button with a label that
     * says "Clear". The button is colored with the specified color.
     * @param cmdName the text to be displayed on the button
     */
    private void makeActionToolTitleName(String cmdName) {
        Label commandName = new Label(cmdName);
        commandName.setTextFill(SimplePaintObjects.TOOL_FG);
        commandName.setFont(Font.font(
                "Verdana",
                FontWeight.BOLD,
                20));
        this.getChildren().add(commandName);
    }

    /**
     * Creates a tool that is used to perform an action, such as clearing the
     * canvas: the tool is a button with a label that says "Clear".
     *
     * @param cmdName the text to be displayed on the button
     * @param action  the action to be performed when the tool is activated
     */
    public void activate() {
        super.activate();
        action.run();
    }
}

/**
 * A tool that draws a shape on the canvas. The shape is drawn when the
 * user drags the mouse from one point to another. The shape is drawn
 * with the current drawing color.
 */
abstract class ShapeTool extends AbstractTool {
    /**
     * Creates a tool that is used to draw a shape on the canvas.
     *
     * @param color
     */
    public ShapeTool(Color color) {
        super(color);
    }

    /**
     * Draws a shape on the canvas. The shape is drawn from
     * the point (startX,startY) to the point (endX,endY).
     * <ul>
     * <li>The shape is drawn with the current drawing color.</li>
     * <li>The shape is filled with the current drawing color.</li>
     * </ul>
     *
     * @param gc    the graphics context for drawing on the canvas
     * @param color the color to use for drawing the shape
     * @param start the starting point for the shape
     * @param end   the ending point for the shape
     */
    abstract public void draw(
            // the graphics context for drawing on the canvas
            GraphicsContext gc,
            // the color to use for drawing the shape
            Color color,
            // the starting point for the shape (in canvas coordinates)
            Point2D start,
            // the ending point for the shape (in canvas coordinates)
            Point2D end);

    abstract public ShapeObject getPaintShape();
}

/**
 * A tool that draws a line on the canvas. The line is drawn when the
 * user drags the mouse from one point to another. The line is drawn
 * with the current drawing color.
 */
class PointTool extends ShapeTool {
    int penWidth;
    LineSegmentShape lineSegmentShape;

    /**
     * Creates a tool that is used to draw a point on the canvas.
     * The point is drawn:
     * when the user clicks the mouse.
     * with the current drawing color.
     * as a small circle with the current drawing color.
     * The size of the circle is determined by the value of the penWidth field.
     * The penWidth field:
     * is set to 2 when the tool is created.
     * can be changed by calling the setPenWidth() method.
     * is used by the draw() method.
     * The draw() method is:
     * called by the mousePressed() method in the PaintPanel class.
     * not called by the mouseDragged() method in the PaintPanel class.
     * not called by the mouseReleased() method in the PaintPanel class.
     * not called by the mouseMoved() method in the PaintPanel class.
     *
     * @param penWidth the width of the pen to be used for drawing the point
     */
    public PointTool(int penWidth) {
        super(SimplePaintObjects.TOOL_RECT_FG);
        this.penWidth = penWidth;
        lineSegmentShape = new LineSegmentShape(
                new Point2D(0, 0),
                new Point2D(0, 0),
                Color.BLACK,
                penWidth);
        makePointToolIcon(penWidth);
    }

    private void makePointToolIcon(int penWidth) {
        Ellipse toolIcon = new Ellipse(penWidth, penWidth);
        toolIcon.setStroke(SimplePaintObjects.TOOL_FG);
        toolIcon.setFill(SimplePaintObjects.TOOL_FG);
        this.getChildren().add(toolIcon);
    }

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

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

/**
 * LineTool is a tool that is used to draw a line on the canvas.
 */
class LineTool extends ShapeTool {
    private static final int line_tool_icon_stroke_width = 2;
    private static final int line_tool_icon_inset = 20;
    LineShape lineShape;

    /**
     * Creates a tool that is used to draw a line on the canvas.
     * The line is drawn:
     * when the user drags the mouse from one point to another.
     * with the current drawing color.
     * as a line with the current drawing color.
     * The draw() method is:
     * called by the mousePressed() method in the PaintPanel class.
     * called by the mouseDragged() method in the PaintPanel class.
     * called by the mouseReleased() method in the PaintPanel class.
     * not called by the mouseMoved() method in the PaintPanel class.
     *
     * @param color the color to use for drawing the line
     */
    public LineTool() {
        super(SimplePaintObjects.TOOL_RECT_FG);
        lineShape = new LineShape(
                new Point2D(0, 0),
                new Point2D(0, 0),
                Color.BLACK,
                2);
        makeLineToolIcon(); // Icon for the tool
    }

    /**
     * Creates an icon for the tool.
     */
    private void makeLineToolIcon() {
        Line toolIcon = new Line(
                0, // start X
                0, // start Y
                   // end X
                SimplePaintObjects.CELL_W - line_tool_icon_inset,
                // end Y
                SimplePaintObjects.CELL_H - line_tool_icon_inset);
        // color of the line icon
        toolIcon.setStroke(SimplePaintObjects.TOOL_FG);
        toolIcon.setStrokeWidth(line_tool_icon_stroke_width);
        this.getChildren().add(toolIcon);
    }

    /**
     * Draws a line on the canvas.
     * The line is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(
            GraphicsContext gc,
            Color color,
            Point2D start,
            Point2D end) {
        lineShape = new LineShape(start, end, color, 2);
    }

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

}

/**
 * RectangleTool is a tool that is used to draw a rectangle on the canvas.
 */
class RectangleTool extends ShapeTool {
    private static final int rect_tool_icon_side_length = 35;
    RectangleShape rectangleShape;

    /**
     * Creates a tool that is used to draw a rectangle on the canvas.
     * The rectangle is drawn:
     * when the user drags the mouse from one point to another.
     * with the current drawing color.
     * as a rectangle with the current drawing color.
     * The draw() method is:
     * called by the mousePressed() method in the PaintPanel class.
     * called by the mouseDragged() method in the PaintPanel class.
     * called by the mouseReleased() method in the PaintPanel class.
     * not called by the mouseMoved() method in the PaintPanel class.
     *
     * @param color the color to use for drawing the rectangle
     */
    public RectangleTool() {
        super(SimplePaintObjects.TOOL_RECT_FG);
        rectangleShape = new RectangleShape(0.0,
                0.0, 0.0, 0.0, Color.BLACK);
        // Icon for the tool
        Rectangle toolIcon = new Rectangle(
                rect_tool_icon_side_length, // width
                rect_tool_icon_side_length); // height
        toolIcon.setStroke(SimplePaintObjects.TOOL_FG);
        toolIcon.setFill(SimplePaintObjects.TOOL_FG);
        this.getChildren().add(toolIcon);
    }

    /**
     * Draws a rectangle on the canvas.
     * The rectangle is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(
            GraphicsContext gc,
            Color color,
            Point2D start,
            Point2D end) {
        rectangleShape = new RectangleShape(
                start.getX(),
                start.getY(),
                end.getX(),
                end.getY(),
                color);
    }

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

/**
 * OvalTool is a tool that is used to draw an oval on the canvas.
 */
class OvalTool extends ShapeTool {
    private static final int oval_tool_icon_radius_length = 20;
    OvalShape ovalShape;
    /**
     * Creates a tool that is used to draw an oval on the canvas.
     * The oval is drawn:
     * when the user drags the mouse from one point to another.
     * with the current drawing color.
     * as an oval with the current drawing color.
     * The draw() method is:
     * called by the mousePressed() method in the PaintPanel class.
     * called by the mouseDragged() method in the PaintPanel class.
     * called by the mouseReleased() method in the PaintPanel class.
     * not called by the mouseMoved() method in the PaintPanel class.
     *
     * @param color the color to use for drawing the oval
     */
    public OvalTool() {
        super(SimplePaintObjects.TOOL_RECT_FG);
        ovalShape = new OvalShape(
                0.0,
                0.0,
                0.0,
                0.0,
                Color.BLACK);
        // Icon for the tool
        Ellipse toolIcon = new Ellipse(
                oval_tool_icon_radius_length, // radiusX
                oval_tool_icon_radius_length); // radiusY
        toolIcon.setStroke(SimplePaintObjects.TOOL_FG);
        toolIcon.setFill(SimplePaintObjects.TOOL_FG);
        this.getChildren().add(toolIcon);
    }

    /**
     * Draws an oval on the canvas.
     * The oval is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(
            GraphicsContext gc,
            Color color,
            Point2D start,
            Point2D end) {
        ovalShape = new OvalShape(
                start.getX(), start.getY(), end.getX(), end.getY(), color);
    }

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

/**
 * RoundedRectangleTool is a tool that is used to draw a rounded rectangle on
 * the canvas.
 */
class RoundedRectangleTool extends ShapeTool {
    private static final int rRect_tool_icon_arc_diameter = 10;
    private static final int rRect_tool_icon_side_length = 35;
    RoundedRectangleShape roundedRectangleShape;

    /**
     * Creates a tool that is used to draw a rounded rectangle on the canvas.
     * The rounded rectangle is drawn:
     * when the user drags the mouse from one point to another.
     * with the current drawing color.
     * as a rounded rectangle with the current drawing color.
     * The draw() method is:
     * called by the mousePressed() method in the PaintPanel class.
     * called by the mouseDragged() method in the PaintPanel class.
     * called by the mouseReleased() method in the PaintPanel class.
     * not called by the mouseMoved() method in the PaintPanel class.
     *
     * @param color the color to use for drawing the rounded rectangle
     */
    public RoundedRectangleTool() {
        super(SimplePaintObjects.TOOL_RECT_FG);
        roundedRectangleShape = new RoundedRectangleShape(
                0.0,
                0.0,
                0.0,
                0.0,
                Color.BLACK);
        // Icon for the tool
        makeRoundRectangleToolIcon();
    }

    private void makeRoundRectangleToolIcon() {
        Rectangle toolIcon = new Rectangle(
                rRect_tool_icon_side_length, // width
                rRect_tool_icon_side_length); // height
        toolIcon.setArcWidth(rRect_tool_icon_arc_diameter);
        toolIcon.setArcHeight(rRect_tool_icon_arc_diameter);
        toolIcon.setStroke(SimplePaintObjects.TOOL_FG);
        toolIcon.setFill(SimplePaintObjects.TOOL_FG);
        this.getChildren().add(toolIcon);
    }

    /**
     * Draws a rounded rectangle on the canvas.
     * The rounded rectangle is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(
            GraphicsContext gc,
            Color color,
            Point2D start,
            Point2D end) {
        roundedRectangleShape = new RoundedRectangleShape(
                start.getX(),
                start.getY(),
                end.getX(),
                end.getY(),
                color);
    }

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

/**
 * The ShapeObject interface defines:
 * - The methods that all ShapeObjects must implement.
 * - Each shape object must know how to draw itself when provided
 * with a GraphicsContext object.
 * - This means that each shape must know its origin,
 * dimension, and color.
 *
 * The interface also defines
 * - the boolean dragUpdate() method that returns true if the object
 * should update the objects list while the mouse is dragging.
 */
interface ShapeObject {
    /**
     * Draws the shape on the canvas.
     *
     * @param gc the GraphicsContext object to use for drawing
     */
    public void draw(GraphicsContext gc);

    /**
     * Returns true if the object should update the objects list
     * while the mouse is dragging.
     *
     * @return true if the object should update the objects list
     *         while the mouse is dragging
     */
    public boolean dragUpdate();
}

/**
 * LineSegmentShape is a shape that is used to draw a line on the canvas.
 *
 * The PointTool:
 * - draws LineSegmentShapes which are nothing more than lines extending
 * from wherever the mouse was when the last mouse event fired to
 * wherever the mouse is when the current mouse event fired.
 * LineSegmentShapes have:
 * - a width and color as well as a start and end location.
```

```
  */
class LineSegmentShape implements ShapeObject {
    private int penWidth;
    private Point2D start;
    private Point2D end;
    private Color color;

    /**
     * Creates a line segment shape.
     *
     * @param start    the start point of the line segment
     * @param end      the end point of the line segment
     * @param color    the color of the line segment
     * @param penWidth the width of the line segment
     */
    public LineSegmentShape(
            Point2D start,
            Point2D end,
            Color color,
            int penWidth) {
        this.penWidth = penWidth;
        this.start = start;
        this.end = end;
        this.color = color;
    }

    /**
     * Draws the line on the canvas.
     * The line is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(GraphicsContext gc) {
        gc.setLineWidth(penWidth);
        gc.setStroke(color);
        gc.setFill(color);
        gc.strokeLine(start.getX(), start.getY(), end.getX(), end.getY());
    }

    /**
     * update the objects list while dragging the mouse for
     * lines and points (but not for rectangles and ovals)
     *
     * @return true if the object should update the objects list
     *         while the mouse is dragging
     */
    @Override
    public boolean dragUpdate() {
        return true;
    }
}

/**
  * The LineShape class isn't all that different from the LineSegmentShape
```

```
 * class except that its width is fixed and it is not updated on drag.
 * This is created by the LineTool object.
 */
class LineShape implements ShapeObject {
    private Point2D start;
    private Point2D end;
    private Color color;
    private int penWidth;

    /**
     * Creates a LineShape.
     *
     * @param start the start point of the line
     * @param end   the end point of the line
     * @param color the color of the line
     */
    public LineShape(Point2D start, Point2D end, Color color, int penWidth) {
        this.penWidth = penWidth;
        this.start = start;
        this.end = end;
        this.color = color;
    }

    /**
     * Draws the line on the canvas.
     * The line is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(GraphicsContext gc) {
        gc.setStroke(color);
        gc.setLineWidth(penWidth);
        gc.strokeLine(start.getX(), start.getY(), end.getX(), end.getY());
    }

    /**
     * Don't update the list while dragging the line tool around the canvas
     *
     * @return true if the object should update the objects list
     *         while the mouse is dragging
     */
    @Override
    public boolean dragUpdate() {
        return false;
    }
}

/**
 * FilledPolyShape class represents the detail of the filled polygon shapes.
 *
 * The size of the polygon based on the mouse coordinates is identical for all
 * filled polygon shapes, hence the code to do this should be
 * in this base class.
 */
```

```
abstract class FilledPolyShape implements ShapeObject {
    Color color;
    double xInital;
    double yInital;
    double xPoints;
    double yPoints;
    double xDiff;
    double yDiff;

    public FilledPolyShape(
            double xInital,
            double yInital,
            double xPoints,
            double yPoints,
            Color color) {
        this.xInital = xInital;
        this.yInital = yInital;
        this.xPoints = xPoints;
        this.yPoints = yPoints;
        this.color = color;
        xDiff = Math.abs(xPoints - xInital);
        yDiff = Math.abs(yPoints - yInital);
    }

    /**
     * Returns true if the object should update the objects list
     * while the mouse is dragging.
     *
     * @return true if the object should update the objects list
     *         while the mouse is dragging
     */
    @Override
    public boolean dragUpdate() {
        return false;
    }
}

/**
 * RectangeShape create the rectangle shape.
 * The size of the rectangle based on the mouse coordinates is identical
 * for all rectangle shapes, hence the code to do this should be in this
 * base class
 */
class RectangleShape extends FilledPolyShape {
    /**
     * Creates a RectangleShape.
     */
    public RectangleShape(
            double xInital,
            double yInital,
            double xPoints,
            double yPoints,
            Color color) {
        super(xInital, yInital, xPoints, yPoints, color);
    }

    /**
     * Draws the rectangle on the canvas.
     * The rectangle is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(GraphicsContext gc) {
        // Set the fill color for the rectangle to be drawn
        gc.setFill(this.color);
        /*
         * Draw the rectangle from the point (startX,startY)
         * to the point (endX,endY)
         * The rectangle is drawn from the top left corner
         * to the bottom right corner.
         */
        gc.fillRect(xInital - xDiff, yInital - yDiff, 2 * xDiff, 2 * yDiff);
    }
}

/**
 * OvalShape create the oval shape.
 * The size of the oval based on the mouse coordinates is identical for all
 * oval shapes, hence the code to do this should be in this base class
 */
class OvalShape extends FilledPolyShape {
    /**
     * Creates a OvalShape.
     */
    public OvalShape(
            double xInital,
            double yInital,
            double xPoints,
            double yPoints,
            Color color) {
        super(xInital, yInital, xPoints, yPoints, color);
    }

    /**
     * Draws the oval on the canvas.
     * The oval is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(GraphicsContext gc) {
        // Set the fill color for the oval to be drawn
        gc.setFill(this.color);
        /*
         * Draw the oval from the point (startX,startY)
         * to the point (endX,endY)
         * The oval is drawn from the top left corner
         * to the bottom right corner.
         */
        gc.fillOval(xInital - xDiff, yInital - yDiff, 2 * xDiff, 2 * yDiff);
    }
}

/**
 * RoundedRectangleShape create the rounded rectangle shape. The size of the
 * rounded rectangle based on the mouse coordinates is identical for all
 * rounded rectangle shapes, hence the code to do this should be in this base
 * class
 */
class RoundedRectangleShape extends FilledPolyShape {
    /**
     * Creates a RoundedRectangleShape.
     */
    public RoundedRectangleShape(
            double xInital,
            double yInital,
            double xPoints,
            double yPoints,
            Color color) {
        super(xInital, yInital, xPoints, yPoints, color);
    }

    /**
     * Draws the rounded rectangle on the canvas.
     * The rounded rectangle is drawn from the point (startX,startY)
     * to the point (endX,endY).
     */
    @Override
    public void draw(GraphicsContext gc) {
        // Set the fill color for the rounded rectangle to be drawn
        gc.setFill(this.color);
        /*
         * Draw the rounded rectangle from the point (startX,startY) to the
         * point (endX,endY) The rounded rectangle is drawn from the top left
         * corner to the bottom right corner.
         */
        gc.fillRoundRect(xInital - xDiff,
                yInital - yDiff,
                2 * xDiff,
                2 * yDiff,
                20,
                20);
    }
}

/**
 * A program where the user can sketch in a variety of colors and tools.
 * A color palette list of paint tools is shown along the right edge of the
 * canvas. The user can select a drawing color by clicking on a color in the
 * palette, and in the column to the left (of the color palette) a set of
 * drawing tools. Under the color palette is a "Clear button" that the user
 * can click to clear the sketch. The user draws, with the selected tool, by
 * clicking and dragging in a large white area that occupies most of the
 * canvas.
 */
public class SimplePaintObjects extends Application {
    public static void main(String[] args) {
        Application.launch(args);
    }

    GraphicsContext gc; // The graphics context for drawing on the canvas.
    private Point2D start; // The start point of the mouse drag event.
    private Point2D end; // The end point of the mouse drag event.
    private ColorTool currentColorTool; // The current color tool.
    private ShapeTool currentShapeTool; // The current shape tool.
    private ArrayList<ShapeObject> drawnShapeObjects = new ArrayList<>();

    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Constants - static defined constants in lieu of magic numbers
    //
    static final Color TOOL_RECT_FG = Color.LIGHTCORAL;
    static final Color TOOL_RECT_BG = Color.WHITE;
    static final Color TOOL_FG = Color.LEMONCHIFFON;
    static final int CELL_W = 60;
    static final int CELL_H = CELL_W; // square cells
    static final int PADDING = 5; // padding between cells
    static final int APPLICATION_W = 2 * (CELL_W * PADDING);
    static final int APPLICATION_H = (int) ((double) (APPLICATION_W / 1.5));
    static final int CANVAS_H = APPLICATION_H + (CELL_H + PADDING*3) * 2;
    static final int CANVAS_W = APPLICATION_W;
    static final Color[] palette = {
            Color.BLACK,
            Color.RED,
            Color.GREEN,
            Color.BLUE,
            Color.CYAN,
            Color.MAGENTA,
            Color.YELLOW
    };
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /**
     * The canvas is the area where the user draws.
     * The canvas is a white rectangle that occupies most of the window.
     *
     * @return the canvas to the caller of this method (makeCanvas)
     */
    private Node makeCanvas() {
        Canvas canvas = new Canvas(CANVAS_W, CANVAS_H);
        // Get the graphics context for drawing on the canvas
        gc = canvas.getGraphicsContext2D();
        clearCanvas(); // clear the canvas to white background color (default)
        canvas.setOnMousePressed(e -> mousePressed(e));
        canvas.setOnMouseDragged(e -> mouseDragged(e));
        canvas.setOnMouseReleased(e -> mouseReleased(e));
        return canvas; // return the canvas to the caller
                       // of this method (makeCanvas)
```

```
    }

    /**
     * The color palette is a list of colored rectangles
     * that the user can click on to select a color.
     *
     * @return the color palette and the clear button to the
     *         caller of this method
     */
    private Node makeColorPane() {
        VBox colorPane = new VBox();
        colorPane.setPadding(new Insets(
                PADDING,
                PADDING,
                PADDING,
                PADDING));
        colorPane.setSpacing(PADDING);
        // background fill for the tool pane (column)
        BackgroundFill bgFill = new BackgroundFill(
                Color.WHITE,
                null,
                null);
        // set the background fill for the tool pane (column)
        colorPane.setBackground(new Background(bgFill));
        makeColorPalette(colorPane);
        currentColorTool = (ColorTool) colorPane.getChildren().get(0);
        currentColorTool.activate(); // activate the first color tool
        // Add a clear button to the color palette
        colorPane.getChildren().add( //
            addMouseHandlerToClearTool(
                new ActionTool("Clear", this::clearCanvas)));
        return colorPane;
    }

    /**
     * The color palette is a list of colored rectangles
     * that the user can click on to select a color.
     *
     * @return the color palette to the caller
     *         of this method (makeColorPalette)
     */
    private void makeColorPalette(VBox colorPane) {
        for (Color color : palette) {
            colorPane.getChildren().add(addMouseHandlerToColorTool(
                    new ColorTool(color)));
        }
    }

    /**
     * Adds a mouse handler to a ColorTool object.
     * The mouse handler is a lambda expression.
     *
     * @param tool the ColorTool object to which the mouse handler is added
     * @return the ColorTool object with the mouse handler added to it
     */
    private ColorTool addMouseHandlerToColorTool(ColorTool tool) {
        tool.setOnMousePressed((e) -> {
            this.currentColorTool.deactivate();
            this.currentColorTool = tool;
            tool.activate();
        });
        return tool;
    }

    /**
     * The tool pane is a list of tools that the user can click on to select a
     * tool.
     *
     * @param tool   the ShapeTool object to which the mouse handler is added
     * @param action the action to be performed when the mouse is pressed on
     *               the tool object
     * @return the tool pane to the caller of this method (makeColorPane)
     */
    private ActionTool addMouseHandlerToClearTool(ActionTool tool) {
        tool.setOnMousePressed(value -> {
            tool.activate();
            clearCanvas();
            drawnShapeObjects.clear();
        });
        tool.setOnMouseReleased(value -> tool.deactivate());
        return tool;
    }

    /**
     * The current tool that is selected for drawing. This is null if no tool
     * is selected.
     *
     * @return toolPane
     */
    private Node makeToolPane() {
        VBox toolPane = new VBox();
        /*
         * Set the padding and spacing between the cells in
         * the tool pane (VBox)
         */
        toolPane.setPadding(new Insets(
                PADDING,
                PADDING,
                PADDING,
                PADDING));
        // space between cells in the tool pane (VBox)
        toolPane.setSpacing(PADDING);
        // background fill for the tool pane (column)
        BackgroundFill bgFill = new BackgroundFill(
                Color.WHITE,
                null,
                null);
        // set the background fill for the tool pane (column)
        toolPane.setBackground(new Background(bgFill));
        makeToolPalette(toolPane);
        currentShapeTool = (ShapeTool) toolPane.getChildren().get(0);
        currentShapeTool.activate(); // activate the first color tool
        return toolPane;
    }

    /**
     * The tool palette is a list of colored rectangles
     * that the user can click on to select a tool.
     *
     * @return the tool palette to the caller
     *         of this method (makeToolPalette)
     */
    private void makeToolPalette(VBox toolPane) {
        // add the pen tool sizes to the tool pane (VBox)
        addPenToolSizes_2_4_6_8(toolPane);
        // add a line tool to the tool pane (VBox)
        toolPane.getChildren().add(
                addMouseHandlerToShapeTool(new LineTool()));
        // add a rectangle tool to the tool pane (VBox)
        toolPane.getChildren().add(
                addMouseHandlerToShapeTool(new RectangleTool()));
        // add an oval tool to the tool pane (VBox)
        toolPane.getChildren().add(
                addMouseHandlerToShapeTool(new OvalTool()));
        // add a rounded rectangle tool to the tool pane (VBox)
        toolPane.getChildren().add(
                addMouseHandlerToShapeTool(new RoundedRectangleTool()));
    }

    private ShapeTool addMouseHandlerToShapeTool(ShapeTool tool) {
        tool.setOnMousePressed((e) -> {
            this.currentShapeTool.deactivate();
            this.currentShapeTool = tool;
            tool.activate();
        });
        return tool;
    }

    /**
     * Adds the pen tool sizes to the tool pane.
     * The pen tool sizes are 2, 4, 6, and 8.
     *
     * @param toolPane // the tool pane to which the pen tool sizes are added
     */
    private void addPenToolSizes_2_4_6_8(VBox toolPane) {
        for (int penWidth = 2; penWidth <= 8; penWidth += 2) {
            toolPane.getChildren().add(
                    addMouseHandlerToShapeTool(
                            new PointTool(penWidth)));
        }
    }
    /**
     * The root pane is a BorderPane that contains the canvas,
     * the color palette, and the tool pane in the center.
     * The root pane is the root node of the scene graph.
     * The root pane is returned to the caller of this method (start).
     *
     * @return the root pane to the caller of this method (makeRootPane)
     */
    private Parent makeRootPane() {
        HBox root = new HBox();
        // padding around the root pane (window)
        root.setPadding(new Insets(PADDING));
        /*
         * space between children of the root pane
         * (the canvas and the tool pane)
         */
        root.setSpacing(PADDING);
        // background color of the root pane is light gray (default)
        BackgroundFill bgFill = new BackgroundFill(
                Color.LIGHTGRAY,
                null,
                null);
        // set the background color of the root pane (window)
        root.setBackground(new Background(bgFill));
        // We want our canvas and our two VBoxes
        root.getChildren().add(makeCanvas());
        root.getChildren().add(makeToolPane());
        root.getChildren().add(makeColorPane());
        return root;
    }

    /**
     * Called when the user clicks the mouse on the canvas.
     * This method is not called when the user drags the mouse.
     *
     * @param event the mouse event that was generated when the user clicked
     */
    private void mousePressed(MouseEvent event) {
        // save the start point
        start = new Point2D(event.getX(), event.getY());
    }

    /**
     * Called when the user drags the mouse on the canvas.
     * This method is not called when the user clicks the mouse.
     *
     * @param event the mouse event that was generated when the user dragged
     */
    private void mouseDragged(MouseEvent event) {
        clearCanvas();
        // draw all the shapes that have been drawn so far
        for (ShapeObject shapeObject : drawnShapeObjects) {
            shapeObject.draw(gc); // draw the shape object on the canvas (gc)
        }
```

```
            end = new Point2D(event.getX(), event.getY());
            if (currentShapeTool.getPaintShape().dragUpdate()) {
                currentShapeTool.draw(
                        // the graphics context of the canvas (gc)
                        gc,
                        // the current color of the color tool (currentColorTool)
                        currentColorTool.getColor(),
                        /*
                         * the previous point (prevX, prevY) where the mouse was
                         * dragged
                         */
                        start,
                        /*
                         * the current point (x, y) where the mouse is being
                         * dragged
                         */
                        end);
                /*
                 * add the shape to the list of drawn shapes (drawnShapeObjects)
                 * commit the drawn shape onto the canvas
                 */
                drawnShapeObjects.add(currentShapeTool.getPaintShape());
                /*
                 * save the current point as the previous point
                 * (prevX, prevY) for the next mouse drag event
                 */
                start = end;
            } else {
                currentShapeTool.draw(
                        // the graphics context of the canvas (gc)
                        gc,
                        // the current color of the color tool (currentColorTool)
                        currentColorTool.getColor(),
                        /*
                         * the start point (startX, startY) where the mouse was
                         * pressed
                         */
                        start,
                        /*
                         * the current point (x, y) where the mouse is being
                         * dragged
                         */
                        end);
                // draw the shape object on the canvas (gc) - see the preview
                currentShapeTool.getPaintShape().draw(gc);
            }
        }

        /**
         * Called when the user releases the mouse button on the canvas.
         * This method is not called when the user clicks the mouse.
         *
         * @param event the mouse event that was generated when the user released
         */
```

```
    private void mouseReleased(MouseEvent event) {
        if (currentShapeTool.getPaintShape().dragUpdate() == false) {
            drawnShapeObjects.add(currentShapeTool.getPaintShape());
        }
    }

    /**
     * Clear the canvas by filling it with white.
     * This is called when the user clicks the "Clear" button.
     * It is also called when the program starts.
     * It is not called when the user is drawing.
     * It is not called when the user is selecting a color or a tool.
     * It is not called when the user is moving the mouse.
     */
    private void clearCanvas() {
        gc.setFill(Color.WHITE);
        gc.fillRect(0, 0, CANVAS_W, CANVAS_H);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setScene(new Scene(makeRootPane()));
        primaryStage.setTitle("Simple Paint Objects");
        primaryStage.show();
    }
}
```