Zi Xuan Li

Professor Auda

CSC 22100

April 4th, 2023

**Assignment #2**

***Statement of the problem:***

The purpose of this assignment is to build off the previous assignment and create a class hierarchy of shapes to draw circles, ovals, rectangles, and their intersections in the JavaFX **BorderPane** layout.

***Solution methods:***

The **BorderPane** layout will contain three regions: a top region, a left region, and a center region. The left region will be a **VBox** layout that contains a **Label** object displaying the text "MyColor Palette" and an object titled **MyColorPalette** showing a set of colors from the Enum **MyColor** for color selection. The top region will be a **HBox** layout that contains geometric images of the subclasses that extends **MyShape**. The images will be pressable and when pressed will provide a screen to enter the shape's parameters, after which will draw the shape on the **Canvas** in the center region. The center region will be a **Canvas** layout used for drawing **MyShape** objects and their intersections.

Class **MyPoint** will be used by class **MyShape** to define a point of reference on the **Canvas** and be used by all the subclasses in the hierarchy to define points to create shapes appropriate for each subclass.

Enum **MyColor** will be used by class **MyShape** as well as all the subclasses to define the colors of the shapes. It retains the behavior from assignment #1, and will be used to define the red, green, blue, and opacity components of each color with a value in the range of [0 – 255].

Class **MyShape** will be an abstract class as well as the hierarchy's superclass. It will implement interface **MyShapeInterface**. Methods such as *area*, *perimeter*, *draw*, and *stroke* will be declared as abstract methods, thus must be overridden by each subclass in the hierarchy.

Class **MyRectangle** is one of the subclasses of the hierarchy and extends class **MyShape**. The objective of **MyRectangle** is to create a rectangle of height h and width w, structured on a point (top left corner), that can be filled with a color from Enum **MyColor**.

Class **MyOval** is one of the subclasses of the hierarchy and extends class **MyShape**. The objective of **MyOval** is to create an ellipse defined within a rectangle of height h and width w, structured on a center point, that can be filled with a color from Enum **MyColor**.

***All classes that imported:***

- **javafx.scene.canvas.Canvas**: This class represents an image canvas that can be drawn on using a **GraphicsContext** object.
- **javafx.scene.canvas.GraphicsContext**: This class represents the graphics context for a **Canvas** object, providing methods for drawing shapes, images, and text on the canvas.

- **java.util.Optional**: This class is used to wrap an object that may or may not be present. It allows you to handle cases where a value may be null without throwing a null pointer exception.
- **java.util.ArrayList**: This class is an implementation of the **List** interface that uses an array to store its elements.
- **java.util.Arrays**: This class contains several methods for working with arrays, such as sorting and searching.
- **java.util.List**: This is an interface that defines a list of elements, with methods for accessing, adding, removing, and manipulating the list.
- **javafx.application.Application**: This class is the entry point for JavaFX applications, providing methods for starting and stopping the application.
- **javafx.application.Platform**: This class provides methods for running code on the JavaFX application thread from other threads.
- **javafx.geometry.Insets**: This class represents the padding of a node in the JavaFX scene graph.
- **javafx.scene.Node**: This is the superclass for all nodes in the JavaFX scene graph, representing a graphical element that can be added to a scene.
- **javafx.scene.Scene**: This class represents a scene in a JavaFX application, containing a root node and other properties such as background color and window dimensions.
- **javafx.scene.control.Dialog**: This class represents a modal dialog box that can be used to prompt the user for input or display information.
- **javafx.scene.control.Label**: This class represents a text label in a JavaFX application.
- **javafx.scene.control.TextField**: This class represents a text field that the user can enter text into.
- **javafx.scene.image.Image**: This class represents an image in a JavaFX application.
- **javafx.scene.image.ImageView**: This class represents an image view, which displays an image in a JavaFX application.
- **javafx.scene.layout.***: These are classes that represent various layout containers and positioning options for nodes in the JavaFX scene graph.
- **javafx.scene.paint.Color**: This class represents a color in a JavaFX application.
- **javafx.scene.text.Font**: This class represents a font in a JavaFX application.
- **javafx.scene.transform.Scale**: This class represents a scaling transformation that can be applied to a node in the JavaFX scene graph.
- **javafx.stage.Stage**: This class represents the primary stage of a JavaFX application. It provides methods for controlling the size, position, and visibility of the stage.
- **java.awt.***: These are classes from the Abstract Window Toolkit (AWT) in Java, which provide a set of GUI components for use in Java applications.
- **java.io.FileInputStream:** This class is used to read data from files.

- **java.io.FileNotFoundException**: This is an exception class that is thrown when a file specified by a file path cannot be found or accessed.

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

import java.util.Optional;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.transform.Scale;
import javafx.stage.Stage;

import java.awt.*;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

***Java code:***

MyShape.java

MyShape class is an abstract that serves as the superclass for all the other shape classes. As an abstract class, it cannot be instantiated directly, but instead provides a set of common methods and properties that are inherited by its subclasses.

- **MyShape** defines instance variables for the position and color of the shape, which are common to all shapes.
- The class defines abstract methods for **perimeter()** and **area()**, which must be implemented by each subclass.
- The class defines **setColor()** and **getColor()** methods, which are inherited by all subclasses.
- The class defines a **similarShape()** method, which is used to compare whether two shapes are similar to each other based on their area.
- The class defines **translate()** and **toString()** methods, which are used by all subclasses for moving and printing the shape.
- **MyShape** is the parent class for all the shape subclasses, including **MyPoint**, **MyLine**, **MyRectangle**, **MyOval**, and **MyCircle**.

```java
package com.example.assignment1;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

import java.util.Optional;

public abstract class MyShape implements MyShapeInterface {
    MyPoint p;
    MyColor color;

    MyShape()
    {
        this.p = new MyPoint();
        this.color = MyColor.BLACK;
    }
    MyShape(MyPoint p, MyColor color) {
        setPoint(p);
        setColor(color);
    }
    MyShape(double x ,double y, MyColor color)
    {
        setPoint(p);
        this.color = Optional.ofNullable(color).orElse(MyColor.YELLOW);
    }

    public void setPoint(MyPoint p) {
        this.p = p;
    }
    public void setPoint(double x,double y){p.setPoint(x,y);}

    public void setColor(MyColor color) {
        this.color = color;
    }

    public MyPoint getPoint() {
        return p;
    }

    public MyColor getColor() {return color;}
    public double getX(){return p.getX();}
    public double getY(){return p.getY();}
    public abstract double area();

    //public abstract MyColor getColor();

    public abstract double perimeter();

    public abstract void draw(GraphicsContext gc);
    public abstract void stroke(GraphicsContext GC);

    public abstract MyRectangle getMyBoundingRectangle();
    @Override
    public String toString() {return "This is an object of the MyShape
class";}
```

```
    public abstract boolean containsMyPoint(MyPoint p);
    public abstract boolean similarObject(MyShape S);

    //@Override
    public Canvas drawIntersectMyShapes(double widthCenterCanvas, double
heightCenterCanvas, MyShape s1, MyShape s2, MyColor color) {
        return MyShapeInterface.super.drawIntersectMyShapes(s1, s2,
widthCenterCanvas, heightCenterCanvas, color);
    }
}
```

MyShapeInterface.java

MyShapeInterface is Java interface that defines a set of methods that any class implementing the interface must provide.

- **public MyRectangle getBoundingBox()**: This method should return a **MyRectangle** object that represents the smallest rectangle that completely encloses the shape.
- **public double area()**: This method should return the area of the shape as a **double** value.
- **public double perimeter()**: This method should return the perimeter of the shape as a **double** value.
- **public void setColor(MyColor color)**: This method should set the color of the shape to the specified **MyColor** object.
- **public MyColor getColor()**: This method should return the color of the shape as a **MyColor** object.
- **public boolean isPointEnclosed(MyPoint p)**: This method should return a **boolean** value indicating whether the specified point **p** is enclosed by the shape.
- **public boolean similarObject(MyShape S)**: This method should return a **boolean** value indicating whether the specified **MyShape** object **S** is similar to the shape implementing the interface.
- **public String toString()**: This method should return a **String** representation of the shape.

```
package com.example.assignment1;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

import java.util.ArrayList;
import java.util.List;

public interface MyShapeInterface {

    public abstract MyRectangle getMyBoundingRectangle();
    public abstract boolean containsMyPoint(MyPoint p);

    //boolean containsMyPoint(MyPoint p);
    boolean similarObject(MyShape S);
```

```
    static boolean similarObject(MyShape S1, MyShape S2) {
        String sClassS1 = S1.getClass().toString();
        String sClassS2 = S2.getClass().toString();
        if (sClassS1.equals(sClassS2)) {
            switch (sClassS1) {

                case "class MyRectangle":
                    MyRectangle R1 = (MyRectangle) S1;
                    MyRectangle R2 = (MyRectangle) S2;
                    return R1.getWidth() == R2.getWidth() && R1.getHeight()
== R2.getHeight();

                case "class MyOval":
                    MyOval O1 = (MyOval) S1;
                    MyOval O2 = (MyOval) S2;
                    return O1.getSemiMajor() == O2.getSemiMajor() &&
O1.getSemiMinor() == O2.getSemiMinor();

                case "class MyCircle":
                    MyCircle C1 = (MyCircle) S1;
                    MyCircle C2 = (MyCircle) S2;
                    return C1.getRadius() == C2.getRadius();

                default:
                    return false;
            }
        } else {
            return false;
        }
    }

    static List<MyPoint> intersectMyShapes(MyShape s1, MyShape s2)
    {
        MyRectangle r1 = s1.getMyBoundingRectangle();
        MyRectangle r2 = s2.getMyBoundingRectangle();
        MyRectangle r = overlapMyShapes(r1,r2);

        if(r!=null) {

            double x = r.getTLC().getX();
            double y = r.getTLC().getY();
            double w = r.getWidth();
            double h = r.getHeight();

            List<MyPoint> intersect = new ArrayList<>();

            for (double i = 0; i <= w; i++) {
                double xi = x + i;
                for (double j = 0; j <= h; j++) {
                    MyPoint p = new MyPoint(xi, y + j, null);
                    if (s1.containsMyPoint(p) && s2.containsMyPoint(p)) {
                        intersect.add(p);
                    }
                }
            }
            return intersect;
```

```
        }
        else{return null;}
    }

    static MyRectangle overlapMyShapes(MyShape s1, MyShape s2){

        MyRectangle bound1 = s1.getMyBoundingRectangle();
        MyRectangle bound2 = s2.getMyBoundingRectangle();

        double x1 = bound1.getTLC().getX();
        double y1 = bound1.getTLC().getY();
        double w1 = bound1.getWidth();
        double h1 = bound1.getHeight();

        double x2 = bound2.getTLC().getX();
        double y2 = bound2.getTLC().getY();
        double w2 = bound2.getWidth();
        double h2 = bound2.getHeight();

        if(y1 + h1 < y2 || y2 + h2 < y1)
        {return null;}
        if(x1 + w1 < x2 || x2 + w2 < x1)
        {return null;}
        double x_min = Math.min(x1+w1,x2+w2);
        double x_max = Math.max(x1,x2);
        double y_min = Math.min(y1+h1,y2+h2);
        double y_max = Math.max(y1,y2);

        MyPoint bound_point = new MyPoint(x_max,y_max,null);
        return new MyRectangle(bound_point,Math.abs(x_min-
x_max),Math.abs(y_min-y_max),null);
    }

    public default Canvas drawIntersectMyShapes(MyShape s1, MyShape s2,
double w, double h, MyColor color)
    {

        List<MyPoint> intersect = intersectMyShapes(s1,s2);
        Canvas overlayCV = new Canvas(w,h);
        GraphicsContext overlayGC = overlayCV.getGraphicsContext2D();

        //s1.getMyBoundingRectangle().stroke(overlayGC);
        s1.draw(overlayGC);
        //s2.getMyBoundingRectangle().stroke(overlayGC);
        s2.draw(overlayGC);

        MyRectangle r = overlapMyShapes(s1,s2);
        MyColor colorR = MyColor.AQUA;
        r.setColor(colorR);
        r.stroke(overlayGC);

        if(intersect !=null)
        {
            System.out.println("Intersect");
            for(MyPoint p : intersect)
            {
                p.setColor(color);
```

```
                p.draw(overlayGC);
            }
        }
        else{System.out.println("No Intersect");}
        return overlayCV;
    }

    public static MyRectangle intersectMyRectangles(MyRectangle R1,
MyRectangle R2)
    {
        double x1 = R1.getTLC().getX();
        double y1 = R1.getTLC().getY();
        double w1 = R1.getWidth();
        double h1 = R1.getHeight();

        double x2 = R2.getTLC().getX();
        double y2 = R2.getTLC().getY();
        double w2 = R2.getWidth();
        double h2 = R2.getHeight();

        if(y1+h1 < y2 || y1 > y2 + h2) {return null;}

        if(x1 + w1 < x2 || x1 > x2 + w2) {return null;}

        double xMax = Math.max(x1,x2);
        double yMax = Math.max(y1,y2);
        double xMin = Math.min(x1+w1,x2+w2);
        double yMin = Math.min(y1 +h1,y2 + h2);

        MyPoint p = new MyPoint(xMax,yMax, null);
        return new MyRectangle(p,Math.abs(xMax - xMin), Math.abs(yMax -
yMin), null);
    }

}
```

MyShapeApplication.java

The **MyShapeApplication** class extends the Application class from the javafx.application package, which provides the entry point for a JavaFX application.

- The class has several private instance variables, including a TilePane for displaying the color palette, a Deque for storing **MyShape** objects in a stack, and several doubles for specifying the width and height of the canvas and center pane.
- The class also includes a main method, which launches the JavaFX application by calling the launch method on the Application class.
- The start method initializes the user interface of the application, including the top menu bar and the center pane, and sets up event handlers for the menu items and color tiles.
- The center pane is initially empty but will be populated with a Canvas object when the user draws a shape or selects the "Intersection" menu item.

- The **MyShapeApplication** class also includes several private methods for displaying dialogs and handling user input.
- The **dialogRectangle** method displays a dialog box for the user to input the dimensions and location of a rectangle to be drawn on the canvas. Once the user submits the form, a new MyRectangle object is created with the specified dimensions and location and added to the stack of MyShape objects. This method exists for the line and oval as well.
- The **dialogIntersection** method displays a dialog box prompting the user to draw the intersection of the last two shapes drawn on the canvas. Once the user submits the form, the two most recently drawn shapes are popped off the stack and their intersection is drawn on the canvas.
- The **addCenterCanvas** method takes two MyShape objects and returns a Canvas object representing their intersection.
- Finally, the class includes a public static void main method, which simply calls the launch method on the **MyShapeApplication** class to start the JavaFX application.

```
package com.example.assignment1;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.*;
import java.util.List;

public class MyShapeApplication extends Application{

    public VBox addLeftVBox(double widthLeftCanvas, double heightLeftCanvas,
TilePane TP, MyColor color){

        VBox VB = new VBox();
        VB.setPrefWidth(widthLeftCanvas);
        VB.setPadding(new Insets(5));

        Label lblMyColorPalette = new Label("MyColor Palette");
        lblMyColorPalette.setPrefWidth(widthLeftCanvas);
        lblMyColorPalette.setTextFill(MyColor.WHITE.getJavaFXColor());
        lblMyColorPalette.setBackground(new Background(new
BackgroundFill(Optional.ofNullable(color).orElse(MyColor.GREY).getJavaFXColor
(), CornerRadii.EMPTY, Insets.EMPTY)));
```

```
        VB.getChildren().addAll(lblMyColorPalette, TP);

        return VB;
    }

    public HBox addTopHBox(double widthTopCanvas, double heightTopCanvas,
double widthCenterCanvas, double heightCenterCanvas, BorderPane BP,
MyColorPalette CP, TilePane TP) throws FileNotFoundException {

        HBox HB = new HBox();
        HB.setPrefWidth(widthTopCanvas);
        HB.setPadding(new Insets(5, 5, 5, 5));

        String [] nameImages = new String [] {"Line", "Oval", "Rectangle",
"Intersection"};
        String pathFile =
"C:\\Users\\zixua\\IdeaProjects\\Assignment2\\Geometric Shapes\\";

        Deque<MyShape> stackMyShapes = new ArrayDeque<MyShape>();
        for (String nameImage : nameImages){
            String nameFile = pathFile + nameImage + ".PNG";
            ImageView geometricImage = new ImageView(new Image(new
FileInputStream(nameFile), heightTopCanvas, heightTopCanvas, true, false));

            geometricImage.setOnMouseClicked(e -> {
                switch(nameImage) {
                    /*(case "Arc":
                        dialogArc(widthCenterCanvas, heightCenterCanvas, BP,
CP, TP, stackMyShapes);
                        break;
                    */

                    case "Line":
                        dialogLine(widthCenterCanvas, heightCenterCanvas, BP,
CP, TP, stackMyShapes);
                        break;

                    case "Oval":
                        dialogOval(widthCenterCanvas, heightCenterCanvas, BP,
CP, TP, stackMyShapes);
                        break;

                    /*case "Polygon":
                        dialogPolygon(widthCenterCanvas, heightCenterCanvas,
BP, CP, TP, stackMyShapes);
                        break;
                    */

                    case "Rectangle":
                        dialogRectangle(widthCenterCanvas,
heightCenterCanvas, BP, CP, TP, stackMyShapes);
                        break;

                    /*
                    case "Triangle":
                        dialogTriangle(widthCenterCanvas, heightCenterCanvas,
```

```
BP, CP, TP, stackMyShapes);
                        break;
                    */

                    case "Intersection":
                        dialogIntersection(widthCenterCanvas,
heightCenterCanvas, BP, CP, TP, stackMyShapes);
                        break;
                }
            });

            HB.getChildren().add(geometricImage);
        }

        return HB;
    }


    /*
    public void dialogArc(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShape){

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyOval");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20, 100, 10, 10));

        TextField xCenter = new TextField(); TextField yCenter = new
TextField();
        TextField width = new TextField(); TextField height = new
TextField();
        TextField startingAngle = new TextField(); TextField extentAngle =
new TextField();

        gridDialog.add(new Label("Oval center"), 0, 0);
        gridDialog.add(xCenter, 1, 0);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 0);
        gridDialog.add(yCenter, 1, 1);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas width"),
2, 1);
        gridDialog.add(new Label("Oval Width"), 0,2);
        gridDialog.add(width, 1,2);
        gridDialog.add(new Label("Width as fraction of canvas height"), 2,
2);
        gridDialog.add(new Label("Oval height"), 0, 3);
        gridDialog.add(height, 1, 3);
        gridDialog.add(new Label("Height as fraction of canvas height"), 2,
3);
```

```
        gridDialog.add(new Label("Arc start angle"), 0, 4);
        gridDialog.add(startingAngle, 1, 4);
        gridDialog.add(new Label("In degrees"), 2, 4);
        gridDialog.add(new Label("Arc [extent] angle"), 0, 5);
        gridDialog.add(extentAngle, 1, 5);
        gridDialog.add(new Label("In degrees"), 2, 5);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> xCenter.requestFocus());

        List <String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(xCenter.getText());
geometricImageInputs.add(yCenter.getText());
                geometricImageInputs.add(width.getText());
geometricImageInputs.add(height.getText());
                geometricImageInputs.add(startingAngle.getText());
geometricImageInputs.add(extentAngle.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint pLTC = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            double w = Double.parseDouble(geometricImageInputs.get(2)) *
widthCenterCanvas;
            double h = Double.parseDouble(geometricImageInputs.get(3)) *
heightCenterCanvas;
            double startAngle =
Double.parseDouble(geometricImageInputs.get(4));
            double arcAngle =
Double.parseDouble(geometricImageInputs.get(5));

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyOval O = new MyOval(pTLC, w, h, color);
                        MyArc A = new MyArc(O, startAngle, arcAngle, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        O.stroke(GC);
```

```
                        A.draw(GC);
                        A.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(A);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });

    }
    */

    public void dialogLine(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShape) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyTriangle");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20, 100, 10, 10));

        TextField x1 = new TextField();
        TextField y1 = new TextField();
        TextField x2 = new TextField();
        TextField y2 = new TextField();

        gridDialog.add(new Label("End points as fraction of canvas width and
height"), 0, 0);
        gridDialog.add(new Label("Point 1"), 0, 1);
        gridDialog.add(x1, 1, 1);
        gridDialog.add(y1, 1, 2);
        gridDialog.add(new Label("Point 2"), 0, 2);
        gridDialog.add(x2, 1, 2);
        gridDialog.add(y2, 2, 2);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> x1.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK) {
                geometricImageInputs.add(x1.getText());
geometricImageInputs.add(y1.getText());
                geometricImageInputs.add(x2.getText());
geometricImageInputs.add(y2.getText());
```

```
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint p = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            MyPoint q = new
MyPoint(Double.parseDouble(geometricImageInputs.get(2)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(3)) * heightCenterCanvas, null);

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyLine L = new MyLine(p, q, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        L.draw(GC);
                        L.getMyBoundingRectangle().stroke(GC);

                        stackMyShape.push(L);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }

    public void dialogOval(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes){

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyOval");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
```

```
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10, 10));

        TextField xCenter = new TextField(); TextField yCenter = new
TextField();
        TextField width = new TextField();
        TextField height = new TextField();

        gridDialog.add(new Label("Center"), 0,0);
        gridDialog.add(xCenter, 1, 0);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 0);
        gridDialog.add(yCenter, 1, 1);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas width"),
2, 1);
        gridDialog.add(new Label("width"), 0, 2);
        gridDialog.add(width, 1,2);
        gridDialog.add(new Label("Width as fraction of canvas width"), 2, 2);
        gridDialog.add(new Label("Height"), 0,3);
        gridDialog.add(height, 1,3);
        gridDialog.add(new Label("Height as fraction of canvas height"), 2,
3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> xCenter.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(xCenter.getText());
geometricImageInputs.add(yCenter.getText());
geometricImageInputs.add(width.getText());
geometricImageInputs.add(height.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional <List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint pTLC = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            double w = Double.parseDouble(geometricImageInputs.get(2)) *
widthCenterCanvas;
            double h = Double.parseDouble(geometricImageInputs.get(3)) *
heightCenterCanvas;

            TP.setOnMouseClicked(e -> {
```

```
                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyOval O = new MyOval(pTLC, w, h, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        O.draw(GC);
                        O.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(O);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }

    /*
    public void dialogPolygon(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyPolygon");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        TextField numberSides = new TextField();
        TextField xCenter = new TextField(); TextField yCenter = new
TextField();
        TextField radius = new TextField();

        gridDialog.add(new Label("Number of sides"), 0 , 0);
        gridDialog.add(numberSides, 1, 0);
        gridDialog.add(new Label("Maximum number of sides: " +
MyShapeInterface.maxNumberPolygonSides), 2, 0);
        gridDialog.add(new Label("Center"), 0 ,1);
        gridDialog.add(xCenter, 1, 1);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 1);
        gridDialog.add(yCenter, 1, 2);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas width"),
2, 2);
        gridDialog.add(new Label("Radius"), 0, 3);
        gridDialog.add(radius, 1, 3);
```

```
        gridDialog.add(new Label ("Radius as fraction of the minimum of
canvas width and canvas height"), 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> numberSides.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(numberSides.getText());
geometricImageInputs.add(xCenter.getText());
geometricImageInputs.add(yCenter.getText());
geometricImageInputs.add(radius.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            int N = Integer.parseInt(geometricImageInputs.get(0));
            MyPoint center = new
MyPoint(Double.parseDouble(geometricImageInputs.get(1)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(2)) * heightCenterCanvas, null);
            double r = Double.parseDouble(geometricImageInputs.get(3)) *
Math.min(widthCenterCanvas, heightCenterCanvas);

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyPolygon Y = new MyPolygon(N, center, r, color);

                        GC.clearRect(0,0,widthCenterCanvas,
heightCenterCanvas);
                        Y.draw(GC);
                        Y.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(Y);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }
```

```
    */

    public void dialogRectangle(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyRectangle");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        TextField xPTLC = new TextField(); TextField yPTLC = new TextField();
        TextField width = new TextField();
        TextField height = new TextField();

        gridDialog.add(new Label("Top Left Corner Point"), 0, 0);
        gridDialog.add(xPTLC, 1, 0);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 0);
        gridDialog.add(yPTLC, 1, 1);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas
height"), 2, 1);
        gridDialog.add(new Label("Width"), 0 , 2);
        gridDialog.add(width, 1, 2);
        gridDialog.add(new Label("As fraction of canvas width"), 2, 2);
        gridDialog.add(new Label("Height"), 0, 3);
        gridDialog.add(height, 1, 3);
        gridDialog.add(new Label("As fraction of canvas height"), 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> xPTLC.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(xPTLC.getText());
geometricImageInputs.add(xPTLC.getText());
geometricImageInputs.add(width.getText());
geometricImageInputs.add(height.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();
```

```
        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint pTLC = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            double w = Double.parseDouble(geometricImageInputs.get(2)) *
widthCenterCanvas;
            double h = Double.parseDouble(geometricImageInputs.get(3)) *
heightCenterCanvas;

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyRectangle R = new MyRectangle(pTLC, w, h, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        R.draw(GC);
                        R.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(R);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }

    /*
    public void dialogTriangle(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyTriangle");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        TextField x1 = new TextField(); TextField y1 = new TextField();
        TextField x2 = new TextField(); TextField y2 = new TextField();
        TextField x3 = new TextField(); TextField y3 = new TextField();

        gridDialog.add(new Label("Triangle vertices as fraction of canvas
```

```
width and height"), 0, 0);
        gridDialog.add(new Label("Vertex 1"), 0 , 1);
        gridDialog.add(x1, 1, 1);
        gridDialog.add(y1, 1, 2);
        gridDialog.add(new Label("Vertex 2"), 0 , 2);
        gridDialog.add(x2, 1, 2);
        gridDialog.add(y2, 2, 2);
        gridDialog.add(new Label("Vertex 3"), 0, 3);
        gridDialog.add(x3, 1, 3);
        gridDialog.add(y3, 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> x1.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(x1.getText());
geometricImageInputs.add(y1.getText());
                geometricImageInputs.add(x2.getText());
geometricImageInputs.add(y2.getText());
                geometricImageInputs.add(x3.getText());
geometricImageInputs.add(y3.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint p = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            MyPoint q = new
MyPoint(Double.parseDouble(geometricImageInputs.get(2)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(3)) * heightCenterCanvas, null);
            MyPoint r = new
MyPoint(Double.parseDouble(geometricImageInputs.get(4)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(5)) * heightCenterCanvas, null);

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile: TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyTriangle T = new MyTriangle(p, q, r, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
```

```
                        T.draw(GC);
                        T.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(T);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }
    */

    public void dialogIntersection(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog dialog = new Dialog<>();
        dialog.setTitle("Intersection of 2 MyShape Objects");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        gridDialog.add(new Label("Draw the intersection of the last two
MyShape Objects)"), 0, 0);

        dialog.getDialogPane().setContent(gridDialog);
        dialog.showAndWait().ifPresent(response -> {
            if (response == ButtonType.OK) {
                TP.setOnMouseClicked(e -> {

                    MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                    for (Node tile : TP.getChildren()) {
                        if (tile.getId() == tileId){
                            Pane centerPane = new Pane();

                            MyShape S1 = stackMyShapes.pop();
                            MyShape S2 = stackMyShapes.pop();

centerPane.getChildren().add(addCenterCanvas(widthCenterCanvas,
heightCenterCanvas, S1, S2, color));
                            BP.setCenter(centerPane);
                            break;
                        }
                    }
                });
            }
        });
```

```
    }

    public Canvas addCenterCanvas(double widthCenterCanvas, double
heightCenterCanvas, MyShape S1, MyShape S2, MyColor color){

        return S1.drawIntersectMyShapes(widthCenterCanvas,
heightCenterCanvas, S1, S2, color);
    }

    @Override
    public void start(Stage PS) throws FileNotFoundException {

        double widthCanvas = 800.0; double heightCanvas = 600.0;

        BorderPane BP = new BorderPane();
        Pane topPane = new Pane(); Pane leftPane = new Pane(); Pane
centerPane = new Pane();

        double widthLeftCanvas = 0.3 * widthCanvas; double heightTopCanvas =
0.15 * heightCanvas;
        double widthCenterCanvas = widthCanvas - widthLeftCanvas;
        double heightCenterCanvas = heightCanvas - heightTopCanvas;

        MyColorPalette CP = new MyColorPalette(widthLeftCanvas,
heightCenterCanvas);
        TilePane TP = CP.getPalette();

        Scene SC = new Scene(BP, widthCanvas, heightCanvas,
MyColor.WHITE.getJavaFXColor());
        PS.setTitle("MyShape!");
        PS.setScene(SC);

        MyPoint r = new MyPoint(widthCenterCanvas, 0.5 * heightCenterCanvas,
null);
        MyPoint n = new MyPoint(widthCenterCanvas, heightCenterCanvas, null);
        MyPoint v = new MyPoint(0.5 * widthCenterCanvas, heightCenterCanvas,
null);
        MyPoint h = new MyPoint(0, widthCenterCanvas, null);
        MyPoint u = new MyPoint(0, 0.5 * heightCenterCanvas, null);
        MyPoint t = new MyPoint();
        MyPoint p = new MyPoint(0.5 * widthCenterCanvas, 0, null);
        MyPoint k = new MyPoint(widthCenterCanvas, 0, null);

        MyPoint q = new MyPoint(0.5 * widthCenterCanvas, 0.5 *
heightCenterCanvas, null);

        System.out.println("\nAngle of the line extending from" + r + "to
[origin]" + q + ": " + r.angleX(q));
        System.out.println("Angle of the line extending from" + n + "to
[origin]" + q + ": " + n.angleX(q));
        System.out.println("Angle of the line extending from" + v + "to
[origin]" + q + ": " + v.angleX(q));
        System.out.println("Angle of the line extending from" + h + "to
[origin]" + q + ": " + h.angleX(q));
        System.out.println("Angle of the line extending from" + u + "to
[origin]" + q + ": " + u.angleX(q));
        System.out.println("Angle of the line extending from" + t + "to
```

```
[origin]" + q + ": " + t.angleX(q));
        System.out.println("Angle of the line extending from" + p + "to
[origin]" + q + ": " + p.angleX(q));
        System.out.println("Angle of the line extending from" + k + "to
[origin]" + q + ": " + k.angleX(q));

        MyLine L1 = new MyLine(p, q, null);
        System.out.println("\n" + L1);

        MyLine L2 = new MyLine(r, q, null);
        System.out.println("\n" + L2);

        MyRectangle R = new MyRectangle(p, 0.5 * widthCenterCanvas, 0.5 *
widthCenterCanvas, MyColor.LIME);
        System.out.println("\n" + R);

        MyOval O = new MyOval(q, 0.5 * widthCenterCanvas, 0.5 *
heightCenterCanvas, MyColor.GOLD);
        System.out.println("\n" + O);

        double radius = 0.25 * Math.min(widthCenterCanvas,
heightCenterCanvas);
        MyCircle C = new MyCircle(q, radius, MyColor.GREEN);
        System.out.println("\n" + C);

        /*
        MyTriangle T = new MyTriangle(t, r, v, MyColor.CYAN);
        System.out.println("\n" + T);

        double startAngle = 40;
        double arcAngle = 210;
        MyArc A = new MyArc(O, startAngle, arcAngle, MyColor.GREY);
        System.out.println("\n" + A);
        */

        MyShapeInterface [] shapes = new MyShape [5];
        shapes[0] = L1;
        shapes[1] = L2;
        shapes[2] = R;
        shapes[3] = O;
        shapes[4] = C;
        //shapes[5] = T;
        //shapes[6] = A;
        //shape[7] = Y;

        for (MyShapeInterface shape: shapes){
            System.out.println("\n" + shape);
            System.out.println(shape.getMyBoundingRectangle());
        }

        MyRectangle RR = (MyRectangle) shapes[2];
        MyOval OO = (MyOval) shapes[3];
        MyCircle CC = (MyCircle) shapes[4];
        //MyTriangle TT = (MyTriangle) shapes[5];
        //MyArc AA = (MyArc) shapes[6];
        //MyPolygon PP = (MyPolygon) shapes[7];
```

```
        //System.out.println("\n" + MyShapeInterface.intersectMyShapes(TT,
AA));

        topPane.getChildren().add(addTopHBox(widthCanvas, heightTopCanvas,
widthCenterCanvas, heightCenterCanvas, BP, CP, TP));
        BP.setTop(topPane);

        leftPane.getChildren().add(addLeftVBox(widthLeftCanvas,
heightCenterCanvas, TP, MyColor.BLACK));
        BP.setLeft(leftPane);

        PS.show();
    }

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

MyPoint.java

**MyPoint** is a class that represents a point in a two-dimensional space, with its x and y coordinates stored as double values.

- It is used as a building block for the shapes in the **MyShape** hierarchy, as they are defined by a set of points.
- **MyPoint** has methods to manipulate and retrieve the values of its x and y coordinates, as well as to translate the point by a given amount.
- It also has methods to calculate the distance between two points, the angle between two points, and the distance of the point from the origin.
- **MyPoint** has a draw method that is used to draw a point on a **GraphicsContext** object in the JavaFX library.
- **MyPoint** also has a **toString** method that returns a string representation of the point's coordinates.
- **MyPoint** is associated with the **MyColor** class as each point can be assigned a color, which is represented by a **MyColor** object.

```
package com.example.assignment1;

import javafx.scene.canvas.GraphicsContext;
import java.util.Optional;

public class MyPoint {

    double x, y;
    MyColor color;

    MyPoint(){ setPoint( 0, 0); this.color = MyColor.YELLOW;}
    MyPoint(double x, double y, MyColor color){ setPoint(x, y); this.color =
Optional.ofNullable(color).orElse(MyColor.YELLOW); }
    MyPoint(MyPoint p, MyColor color){ setPoint(p); this.color =
```

```
Optional.ofNullable(color).orElse(MyColor.YELLOW); }

    public void setPoint(double x, double y){ this.x = x; this.y = y; }
    public void setPoint(MyPoint p){ this.x = p.getX(); this.y = p.getY(); }
    public void setColor(MyColor color){ this.color = color;}

    public double getX(){ return x; }
    public double getY(){ return y; }
    public MyColor getColor(){ return color; }

    public void translate(double u, double v){ setPoint(x+u, y+v); }
    public double distanceFromOrigin(){ return Math.sqrt(x * x + y * y); }
    public double distance(MyPoint p){
        double dx = p.getX() - x;
        double dy = p.getY() - y;
        return Math.sqrt(dx * dx + dy + dy);
    }

    public double angleX(MyPoint p) {
        double dx = x - p.getX();
        double dy = y - p.getY();

        double angle = Math.toDegrees(Math.atan2(dy, dx));
        return angle >= 0 ? angle : 360.0 + angle;
    }

    public void draw(GraphicsContext GC) {
        GC.setFill(color.getJavaFXColor());
        GC.fillRect(x, y,1,1);
    }

    @Override
    public String toString(){ return "Point(" + x + ", " + y + ")";}
}
```

MyRectangle.java

**MyRectangle** is a class that represents a square in 2D space. It extends the **MyShape** class and implements the **MyShapeInterface**.

- **MyRectangle** has instance variables for the coordinates of its upper-left corner, its width and height, and its color.
- The class has a constructor that takes in these variables and creates a **MyRectangle** object with the specified dimensions and color.
- The class overrides the **setColor()** method of **MyShape** to set the color of the rectangle.
- The class overrides the **similarObject()** method of **MyShape** to compare its dimensions with those of another **MyShape** object.
- The class overrides the **perimeter()** and **area()** methods of **MyShape** to calculate the perimeter and area of the rectangle.
- **MyRectangle** has a **draw()** method that draws the rectangle using JavaFX's **GraphicsContext**.

- The class has a **toString()** method that returns a string representation of the rectangle's coordinates, dimensions, perimeter, and area.

```
package com.example.assignment1;

import javafx.scene.canvas.GraphicsContext;
import java.util.Optional;

public class MyRectangle extends MyShape{

    MyPoint pTLC;
    double width, height;
    MyColor color;

    MyRectangle(MyPoint p, double width, double height, MyColor color){

        super(new MyPoint(), null);

        this.pTLC = p; this.width = width; this.height = height;
        this.color = Optional.ofNullable(color).orElse(MyColor.YELLOW);
    }

    MyRectangle(MyRectangle R, MyColor color){

        super(new MyPoint(), null);

        this.pTLC = R.getTLC(); this.width = R.getWidth(); this.height =
R.getHeight();
        this.color = Optional.ofNullable(color).orElse(R.getColor());
    }

    @Override
    public void setColor(MyColor color) { this.color = color; }

    public MyPoint getTLC() { return pTLC; }
    public double getWidth() { return width; }
    public double getHeight() { return height; }
    public MyColor getColor() { return color; }

    @Override
    public double perimeter(){ return 2 * (width * height); }
    @Override
    public double area(){ return width * height; }

    @Override
    public void stroke(GraphicsContext GC){

        GC.setStroke(color.getJavaFXColor());
        GC.strokeRect(pTLC.getX(), pTLC.getY(), width, height);
    }

    @Override
    public void draw(GraphicsContext GC){

        GC.setFill(color.getJavaFXColor());
        GC.fillRect(pTLC.getX(), pTLC.getY(), width, height);
    }
```

```
    public MyRectangle getMyBoundingRectangle(){

        return new MyRectangle(pTLC, width, height, null);
    }

    public boolean containsMyPoint(MyPoint p){

        double x = p.getX(); double y = p.getY();
        double xR = pTLC.getX(); double yR = pTLC.getY();

        return (xR <= x && x <= xR + width) && (yR <= y && y <= yR + height);
    }

    public boolean similarObject(MyShape S){

        if (S.getClass().toString().equals("class MyRectangle")){
            MyRectangle R = (MyRectangle) S;
            return (width == R.getWidth() && height == R.getHeight());
        }
        else{
            return false;
        }
    }

    @Override
    public String toString(){

        return "Rectangle Top Left Corner " + pTLC + " Width: " + width
                + " Height " + height + " Perimeter " + perimeter() + " Area
" + area();
    }
}
```

MyOval.java

**MyOval** is a class that represents an oval in 2D space. It extends the **MyShape** class and implements the **MyShapeInterface**.

- **MyOval** is a class that represents an oval shape, defined by a position (MyPoint object), a width and a height (both double values), and a color (MyColor object).
- The constructor of **MyOval** takes in four arguments: a **MyPoint** object representing the position, a double value representing the width, another double value representing the height, and an optional **MyColor** object representing the color.
- The **setWidth** and **setHeight** methods allow the width and height of the oval to be updated after initialization.
- The **getArea** method calculates and returns the area of the oval using the formula π * width * height / 4.
- The **getPerimeter** method calculates and returns the perimeter of the oval using the formula 2 * π * sqrt((width^2 + height^2) / 8).

- The **draw** method allows the oval to be drawn onto a JavaFX GraphicsContext object.
- The **similarObject** method checks if the passed in MyShape object is a MyOval object and whether it has the same width and height as this MyOval object.
- The **toString** method returns a string representation of the oval's position, width, height, area, and perimeter.

```
package com.example.assignment1;

import javafx.scene.canvas.GraphicsContext;
import java.util.Optional;

public class MyOval extends MyShape{
    MyPoint center;
    double width, height;
    double halfWidth, halfHeight;

    double semiMajor, semiMinor;
    MyColor color;

    double focus;
    double eccentricity;

    MyOval(MyPoint center, double width, double height, MyColor color){
        super(new MyPoint(), null);

        this.center = center; this.width = width; this.height = height;
        halfWidth = 0.5 * this.width; halfHeight = 0.5 * this.height;
        semiMajor = Math.max(halfWidth, halfHeight);
        semiMinor = Math.min(halfWidth, halfHeight);

        this.color = Optional.ofNullable(color).orElse(MyColor.YELLOW);

        focus = Math.sqrt(Math.pow(semiMajor, 2) - Math.pow(semiMinor, 2));
        eccentricity = focus / semiMajor;
    }

    MyOval(MyOval O, MyColor color){
        super(new MyPoint(), null);

        this.center = O.getCenter(); this.width = O.getWidth(); this.height =
O.getHeight();
        halfWidth = 0.5 * this.width; halfHeight = 0.5 * this.height;
        semiMajor = Math.max(halfWidth, halfHeight);
        semiMinor = Math.min(halfWidth, halfHeight);

        this.color = Optional.ofNullable(color).orElse(MyColor.YELLOW);

        focus = O.getFocus();
        eccentricity = O.getEccentricity();
    }

    @Override
    public void setColor(MyColor color){ this.color = color; }

    public MyPoint getCenter(){ return center; }
```

```
    public double getWidth(){ return width; }
    public double getHeight(){ return height; }
    public double getSemiMajor(){ return semiMajor; }
    public double getSemiMinor(){ return semiMinor; }

    @Override
    public MyColor getColor(){ return color; }

    public double getFocus(){ return focus; }
    public double getEccentricity(){ return eccentricity; }

    @Override
    public double perimeter(){
        return (Math.sqrt(2) * Math.PI * Math.sqrt(Math.pow(semiMajor, 2) +
Math.pow(semiMinor, 2)));
    }

    @Override
    public double area(){ return Math.PI * semiMajor * semiMinor; }

    @Override
    public void stroke(GraphicsContext GC){
        GC.setStroke(color.getJavaFXColor());
        GC.strokeOval(center.getX() - halfWidth, center.getY() - halfHeight,
width, height);
    }

    public void draw(GraphicsContext GC){
        GC.setFill(color.getJavaFXColor());
        GC.fillOval(center.getX() - halfWidth, center.getY() - halfHeight,
width, height);
    }

    public MyRectangle getMyBoundingRectangle(){
        double x = center.getX() - halfWidth;
        double y = center.getY() - halfHeight;
        MyPoint pLTC = new MyPoint(x, y, null);

        return new MyRectangle(pLTC, width, height, null);
    }

    public boolean containsMyPoint(MyPoint P){

        if (halfWidth == halfHeight) {
            return p.distance(center) <= halfWidth;
        }
        else{
            double hxh = halfWidth * halfHeight;
            double dx = halfHeight * (p.getX() - center.getX());
            double dy = halfWidth * (p.getY() - center.getY());
            return dx * dx + dy * dy <= hxh * hxh;
        }
    }

    public boolean similarObject(MyShape S){

        if (S.getClass().toString().equals("class MyOval")) {
```

```
            MyOval O = (MyOval) S;
            return semiMajor == O.getSemiMajor() && semiMinor ==
O.getSemiMinor();
        }
        else {
            return false;
        }
    }

    @Override
    public String toString(){

        return "Oval Center (" + center.getX() + ", " + center.getY() + ")" +
"\n"
                + "Major Axis " + semiMajor + " Minor Axis" + semiMinor +
"\n"
                + "Perimeter " + perimeter() + "\n"
                + "Area " + area();
     }
}
```

MyCircle.java

**MyCircle** is a class that represents a circle in 2D space. It extends the **MyShape** class and implements the **MyShapeInterface**.

- **MyCircle** is a subclass of **MyOval**, which in turn is a subclass of **MyShape**.
- **MyCircle** extends the functionality of **MyOval** by providing a circle-specific implementation.
- It stores the center point and radius of the circle, along with its color.
- The constructor initializes the circle using the given center point, radius, and color. If the color is null, it defaults to yellow.
- The **similarObject** method compares the radius of this circle with the radius of another **MyCircle** object. If they have the same radius, the method returns true; otherwise, it returns false.
- The **toString** method returns a string representation of the circle, including its center point, radius, perimeter, and area.
- **MyCircle** implements the methods defined in the **MyShape** interface, such as **setColor**, **getColor**, and **draw**.

```
package com.example.assignment1;

import java.util.Optional;

public class MyCircle extends MyOval{
    MyPoint center;
    double radius;
    MyColor color;

    MyCircle(MyPoint p, double radius, MyColor color){
```

```
        super(p, 2.0 * radius, 2.0 * radius, color);

        this.center = p; this.radius = radius;
        this.color = Optional.ofNullable(color).orElse(MyColor.YELLOW);
    }

    @Override
    public void setColor(MyColor color){ this.color = color; }

    public MyPoint getCenter(){ return center; }

    public double getRadius(){ return radius; }

    public MyColor getColor(){ return color; }

    @Override
    public boolean similarObject(MyShape S){

        if (S.getClass().toString().equals("class MyCircle")){
        MyCircle C = (MyCircle) S;
        return radius == C.getRadius();
        }
        else{
            return false;
        }
    }

    @Override
    public String toString(){

        return "Circle Center: (" + center.getX() + ", " + center.getY() +
")" + "\n" +
                "Radius: " + radius + "\n" +
                "Perimeter " + perimeter() + "\n" +
                "Area " + area();
    }
}
```

MyLine.java

**MyLine** is a class that represents a line segment in 2D space. It extends the **MyShape** class and implements the **MyShapeInterface**.

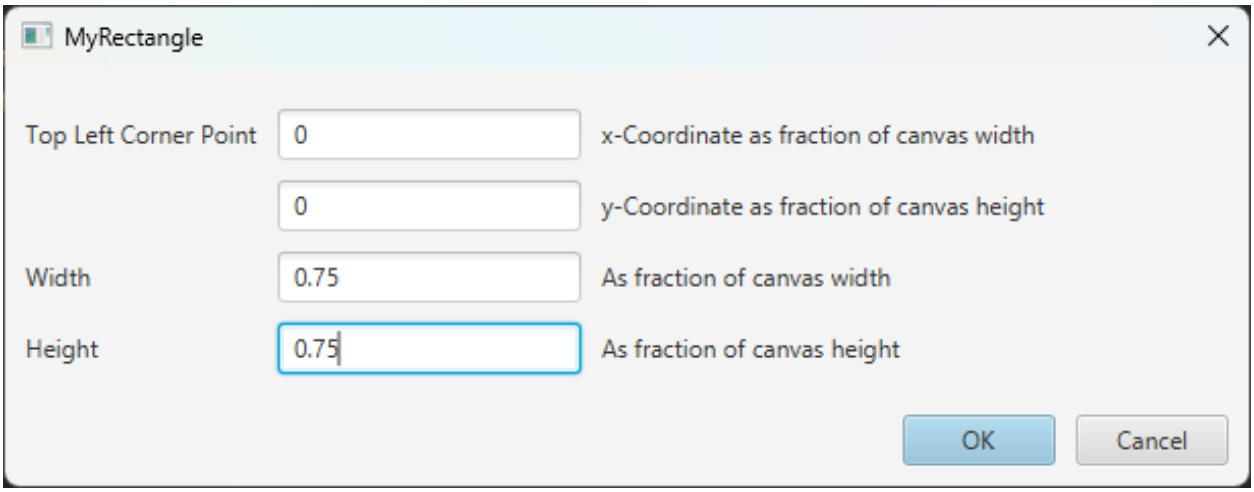
- **MyLine** class has two **MyPoint** objects as instance variables to represent the start and end points of the line segment. It also has a **MyColor** object to represent the color of the line.
- The class has a constructor that takes two **MyPoint** objects and an optional **MyColor** object to create a line segment with the given points and color.
- The class has methods to get and set the start and end points of the line segment, as well as its color. It also has methods to calculate the length and slope of the line segment.
- **MyLine** class implements the **draw()** method from the **MyShape** interface, which draws the line segment on a JavaFX GraphicsContext object.

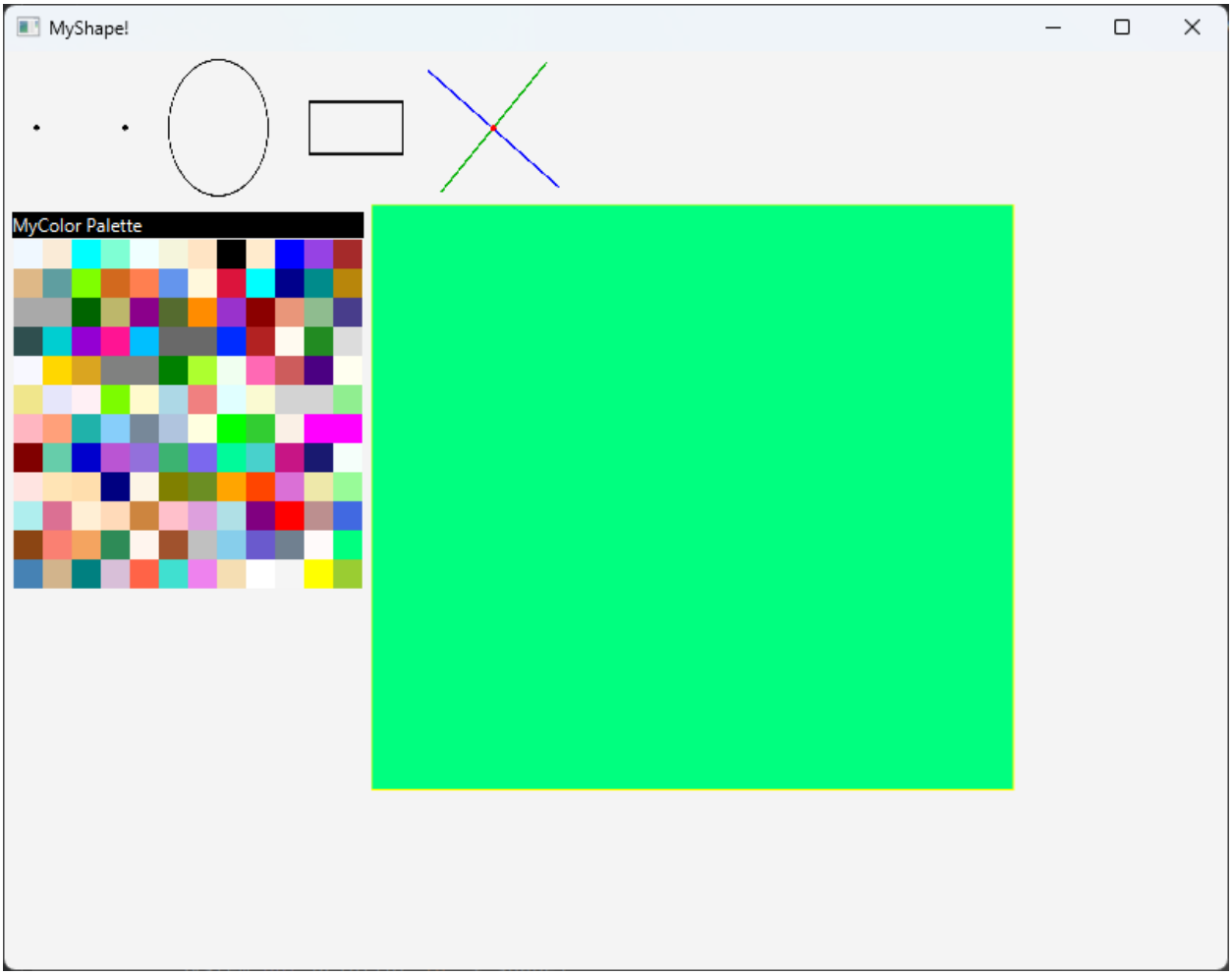
- The class has an overridden **toString()** method that returns a string representation of the line segment, including its start and end points and length.
- MyLine class has an overridden **similarObject()** method from the **MyShape** class that checks if another **MyShape** object is similar to this line segment, which in this case means whether the other object is a **MyLine** with the same start and end points.

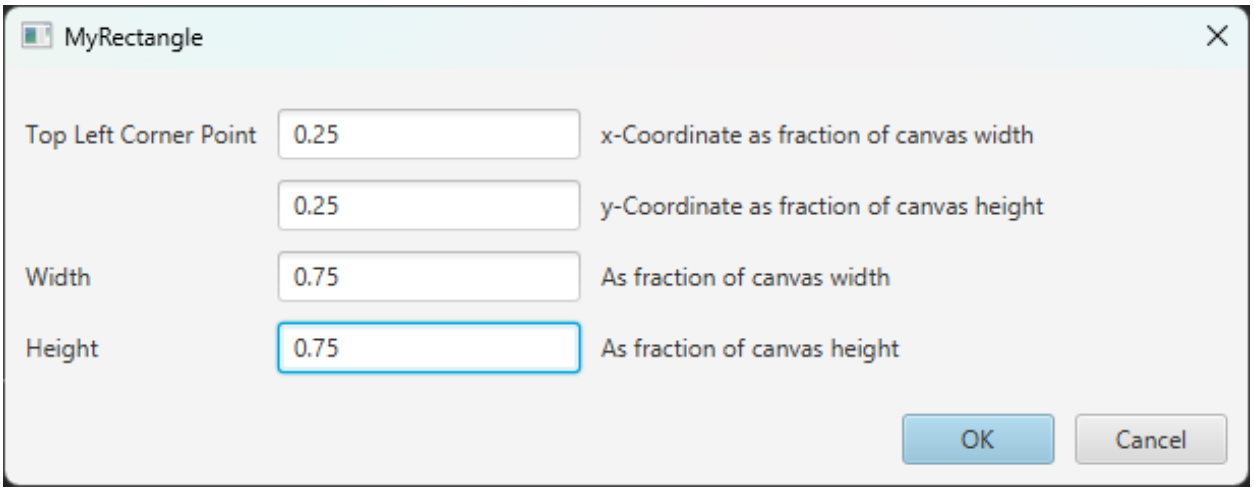
```
package com.example.assignment1;

import javafx.scene.canvas.GraphicsContext;
import java.util.Optional;

public class MyLine extends MyShape{

    MyPoint p1, p2;
    MyPoint [] pLine = new MyPoint[2];
    MyColor color;

    MyLine(MyPoint p1, MyPoint p2, MyColor color){

        super(new MyPoint(), null);

        this.p1 = p1; this.p2 = p2;
        pLine[0] = p1; pLine[1] = p2;
        this.color = Optional.ofNullable(color).orElse(MyColor.YELLOW);
    }

    MyLine(MyLine L, MyColor color){

        super(new MyPoint(), null);

        this.p1 = (L.getLine())[0]; this.p2 = (L.getLine())[1];
        pLine[0] = p1; pLine[1] = p2;
        this.color = Optional.ofNullable(color).orElse(L.getColor());
    }

    @Override
    public void setColor(MyColor color){ this.color = color; };

    public MyPoint [] getLine(){ return pLine; }
    @Override
    public MyColor getColor(){ return color; }

    public double angleX(){ return p1.angleX(p2); }

    public double length(){ return p1.distance(p2); }

    @Override
    public double perimeter() {return length(); }

    @Override
    public double area() { return 0; }

    @Override
    public void stroke(GraphicsContext GC){
```

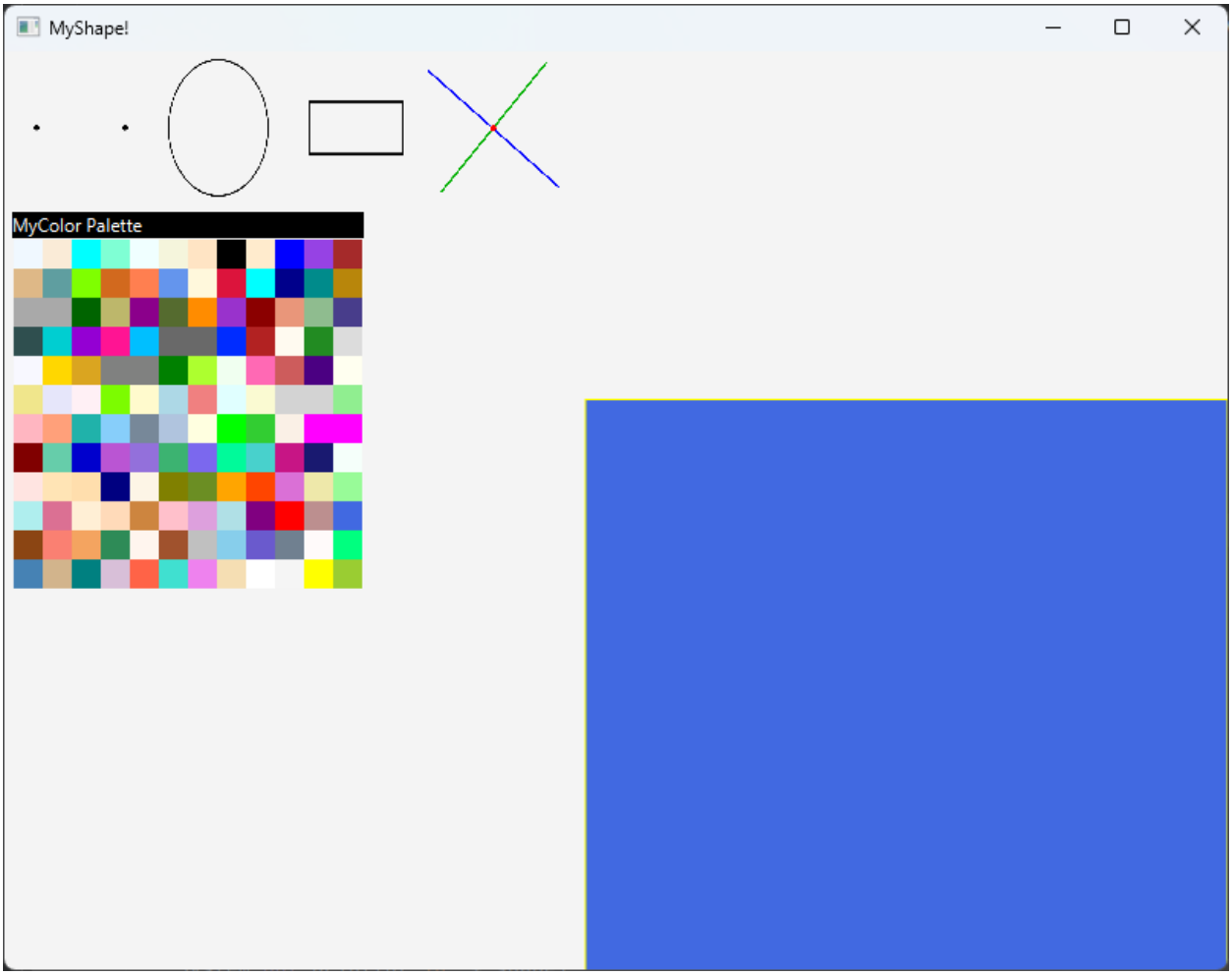
```
        GC.setStroke(color.getJavaFXColor());
        GC.strokeLine(p1.getX(), p1.getY(), p2.getX(), p2.getY());
    }

    public void draw(GraphicsContext GC){

        GC.setStroke(color.getJavaFXColor());
        GC.strokeLine(p1.getX(), p1.getY(), p2.getX(), p2.getY());
    }

    public MyRectangle getMyBoundingRectangle(){

        double x1 = p1.getX(); double y1 = p1.getY();
        double x2 = p2.getX(); double y2 = p2.getY();
        MyPoint pTLC = new MyPoint(Math.min(x1, x2), Math.min(y1, y2), null);

        return new MyRectangle(pTLC, Math.abs(x1 - x2), Math.abs(y1 - y2),
null);
    }

    public boolean containsMyPoint(MyPoint P){

        return (p1.distance(p) + p2.distance(p) == length());
    }

    public boolean similarObject(MyShape S){

        if (S.getClass().toString().equals("class MyLine")){
            MyLine L = (MyLine) S;
            return (this.length() == L.length());
        }
        else {
            return false;
        }
    }

    @Override
    public String toString(){

        return "Line [" + p1 + ", " + p2 + "] Length " + length();
    }
}
```

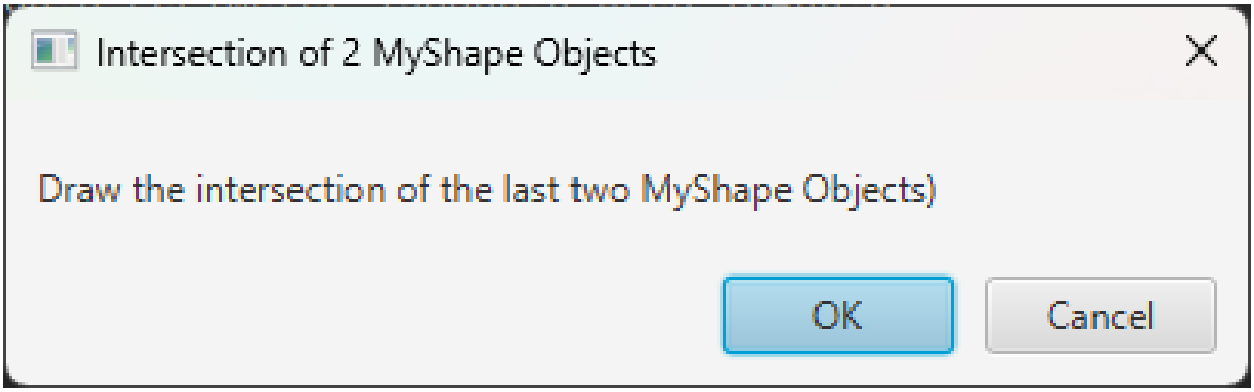
***Output of the code:***

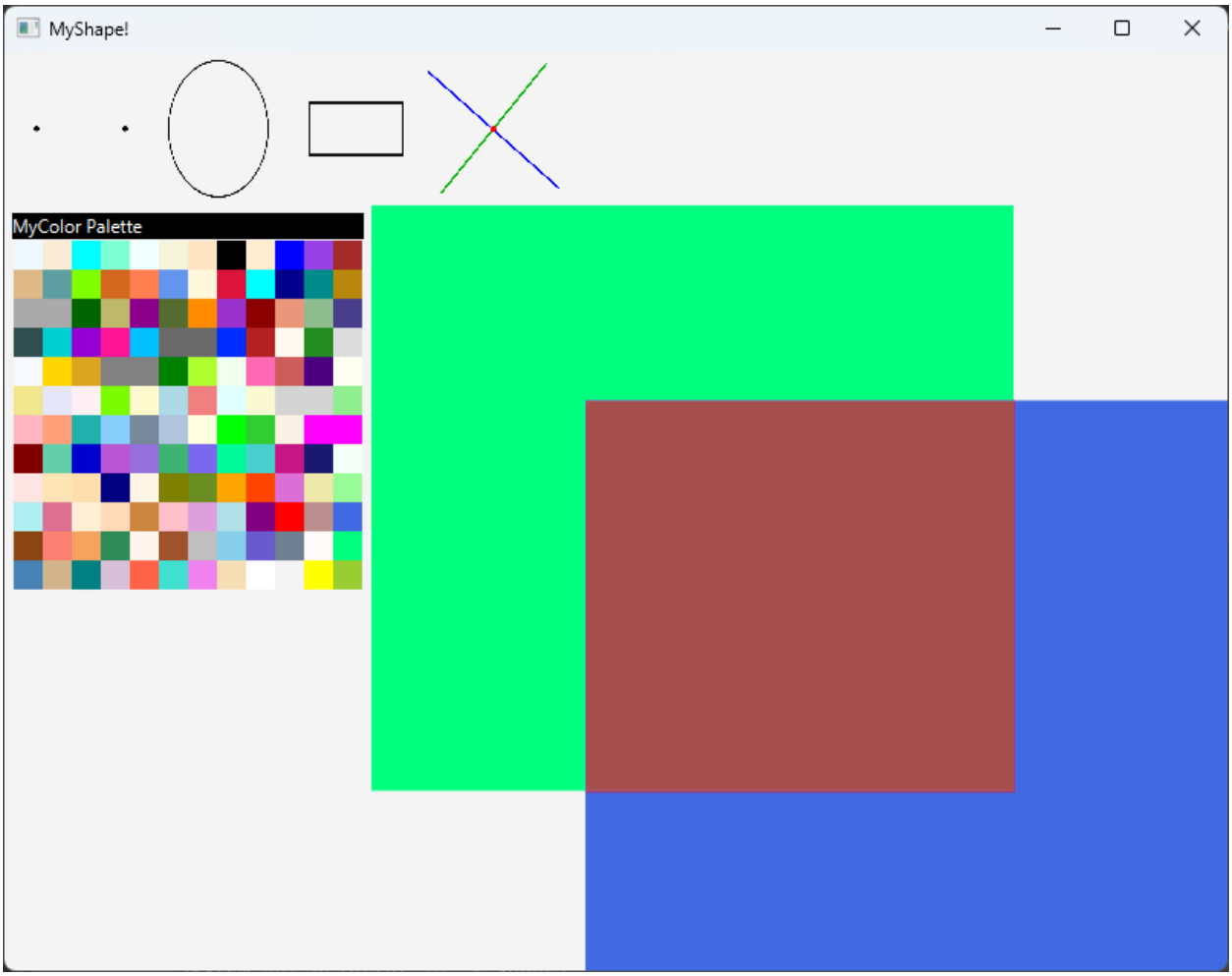

MyRectangle
Top Left Corner Point
0
x-Coordinate as fraction of canvas width
0
y-Coordinate as fraction of canvas height
Width
0.75
As fraction of canvas width
Height
0.75
As fraction of canvas height
OK
Cancel

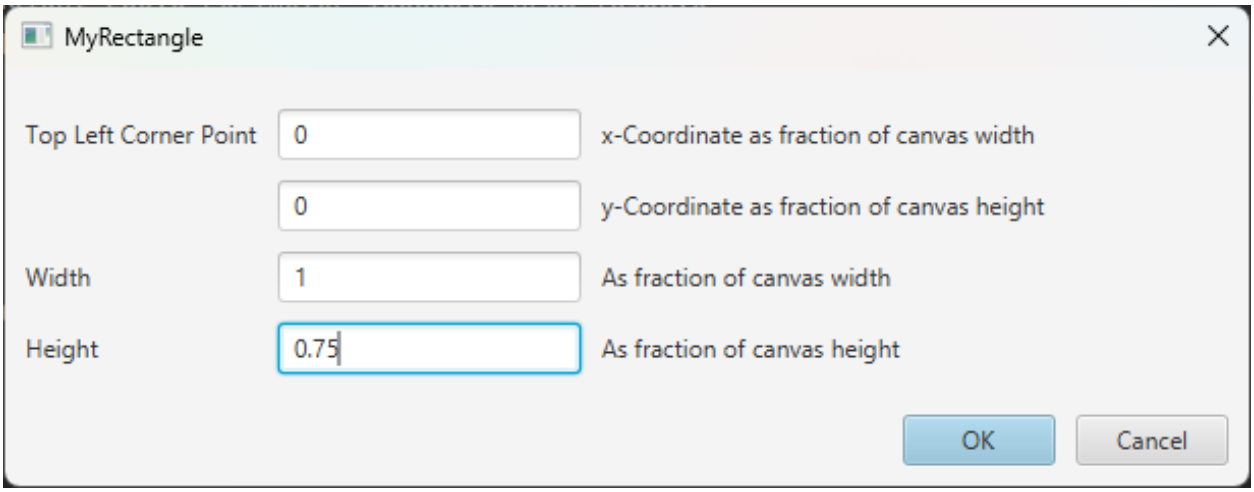

MyShape!
MyColor Palette

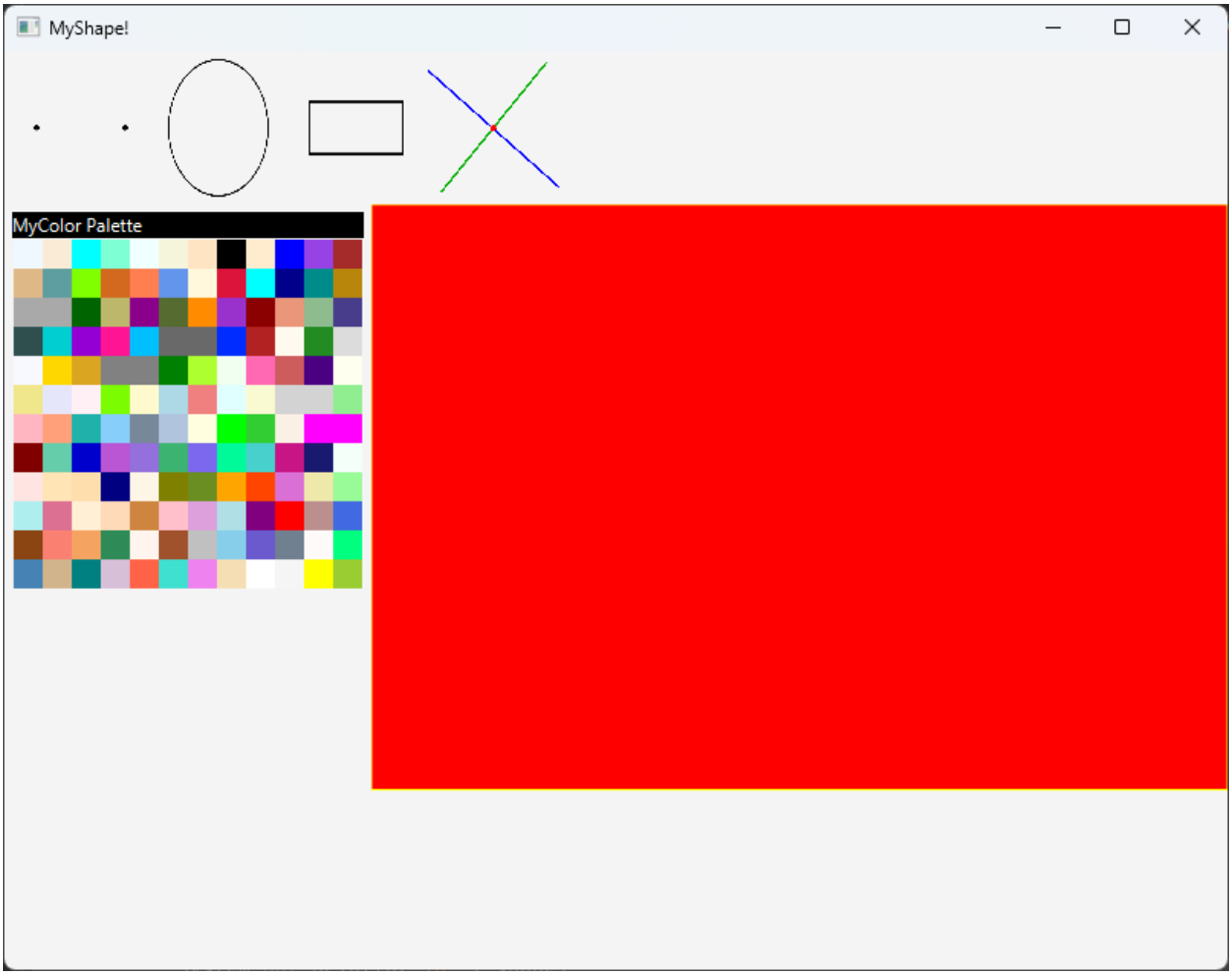

MyRectangle
Top Left Corner Point
0.25
x-Coordinate as fraction of canvas width
0.25
y-Coordinate as fraction of canvas height
Width
0.75
As fraction of canvas width
Height
0.75
As fraction of canvas height
OK
Cancel

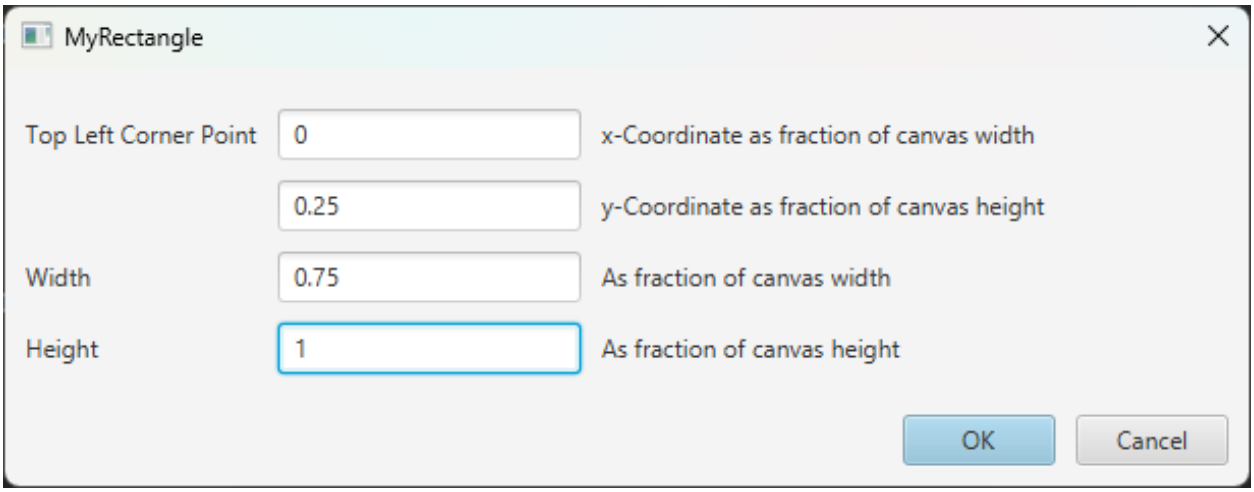

MyShape!
MyColor Palette

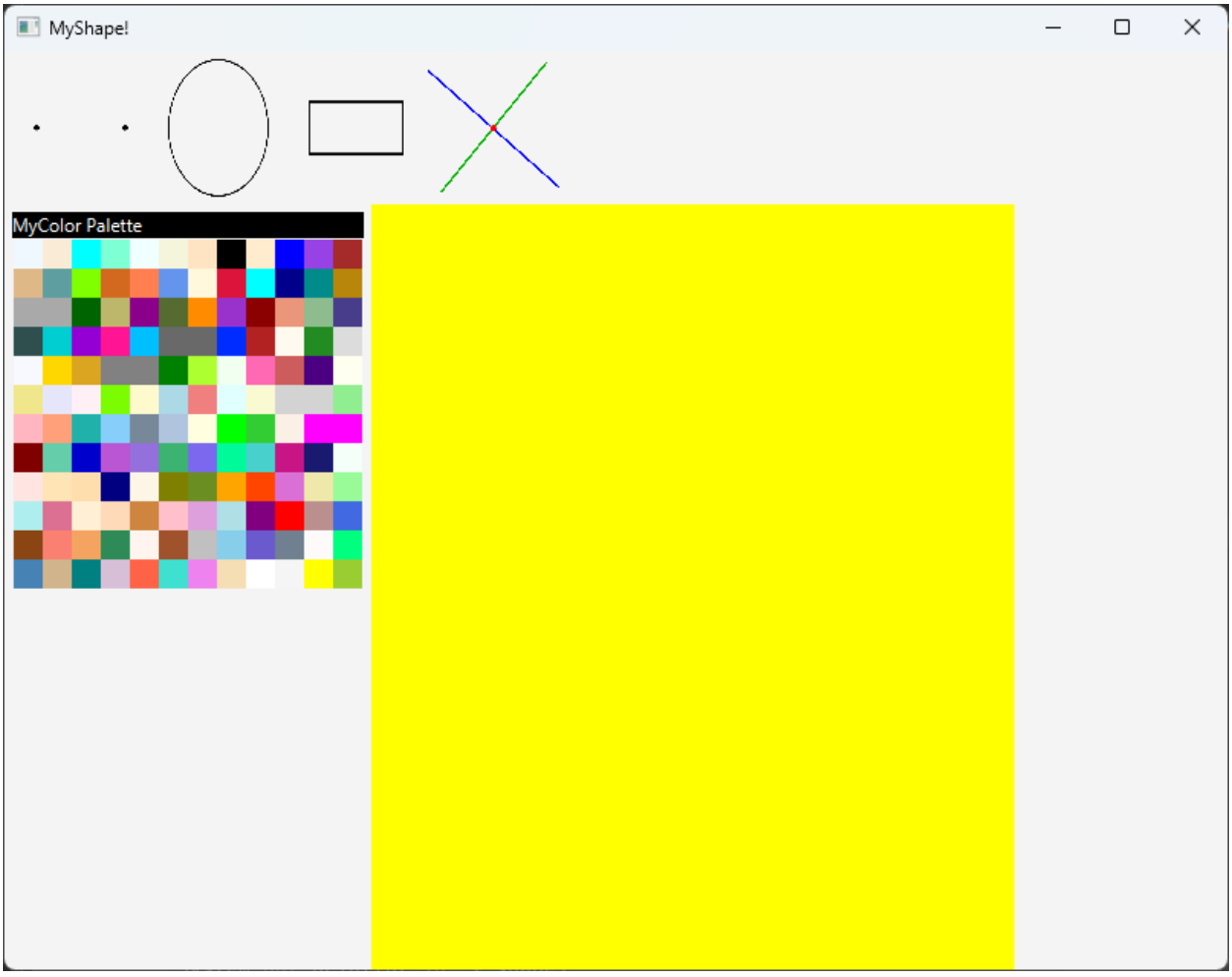

Intersection of 2 MyShape Objects
Draw the intersection of the last two MyShape Objects)
OK
Cancel

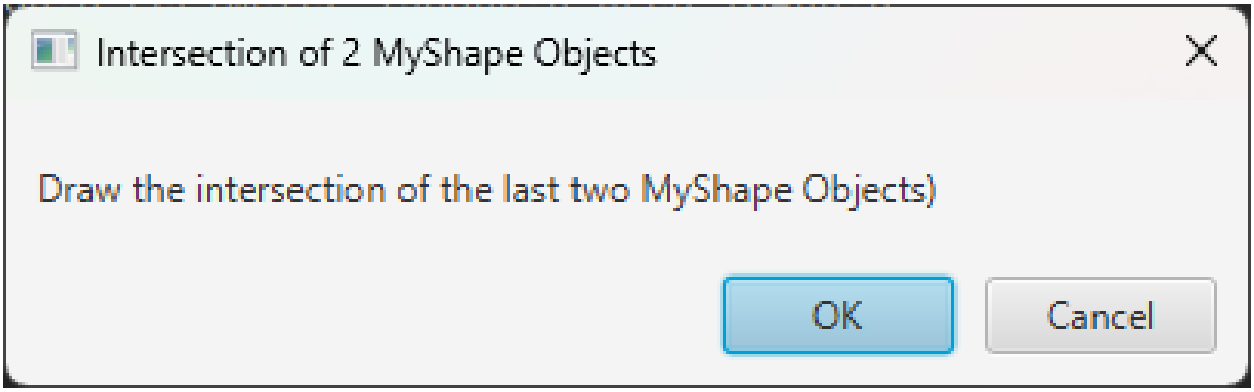

MyShape!
MyColor Palette

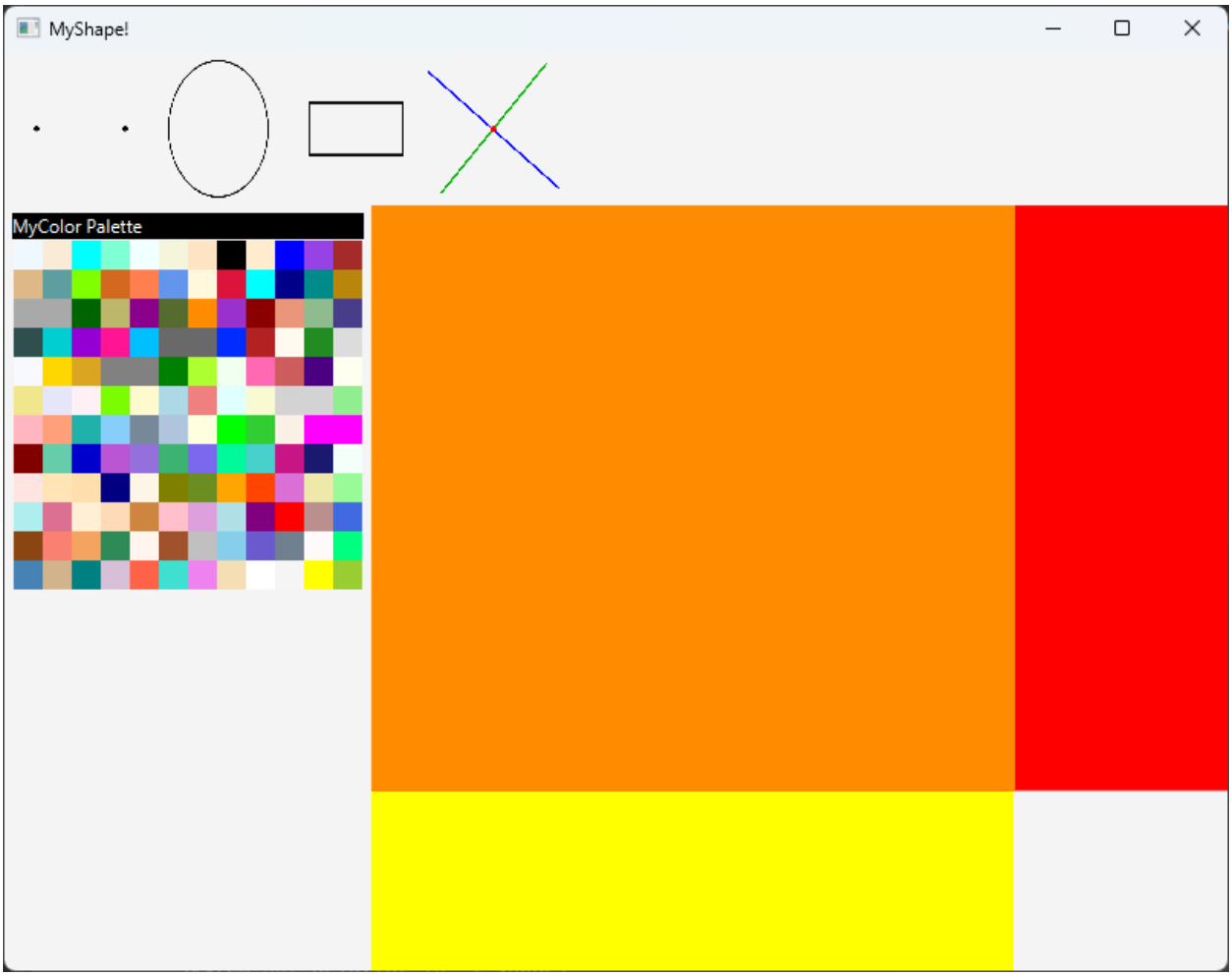

MyRectangle
Top Left Corner Point
0
x-Coordinate as fraction of canvas width
0
y-Coordinate as fraction of canvas height
Width
1
As fraction of canvas width
Height
0.75
As fraction of canvas height
OK
Cancel

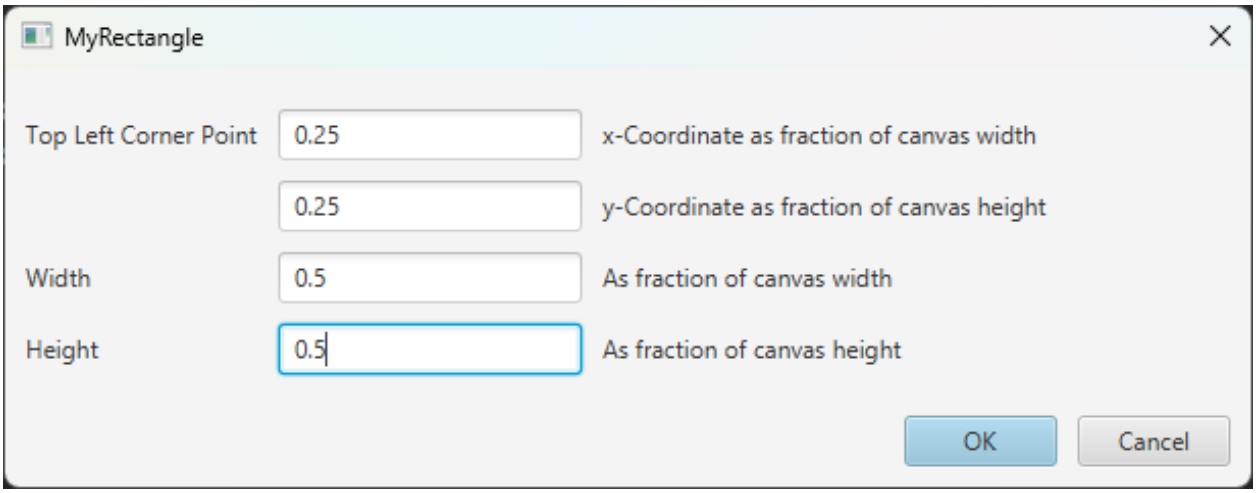

MyShape!
MyColor Palette

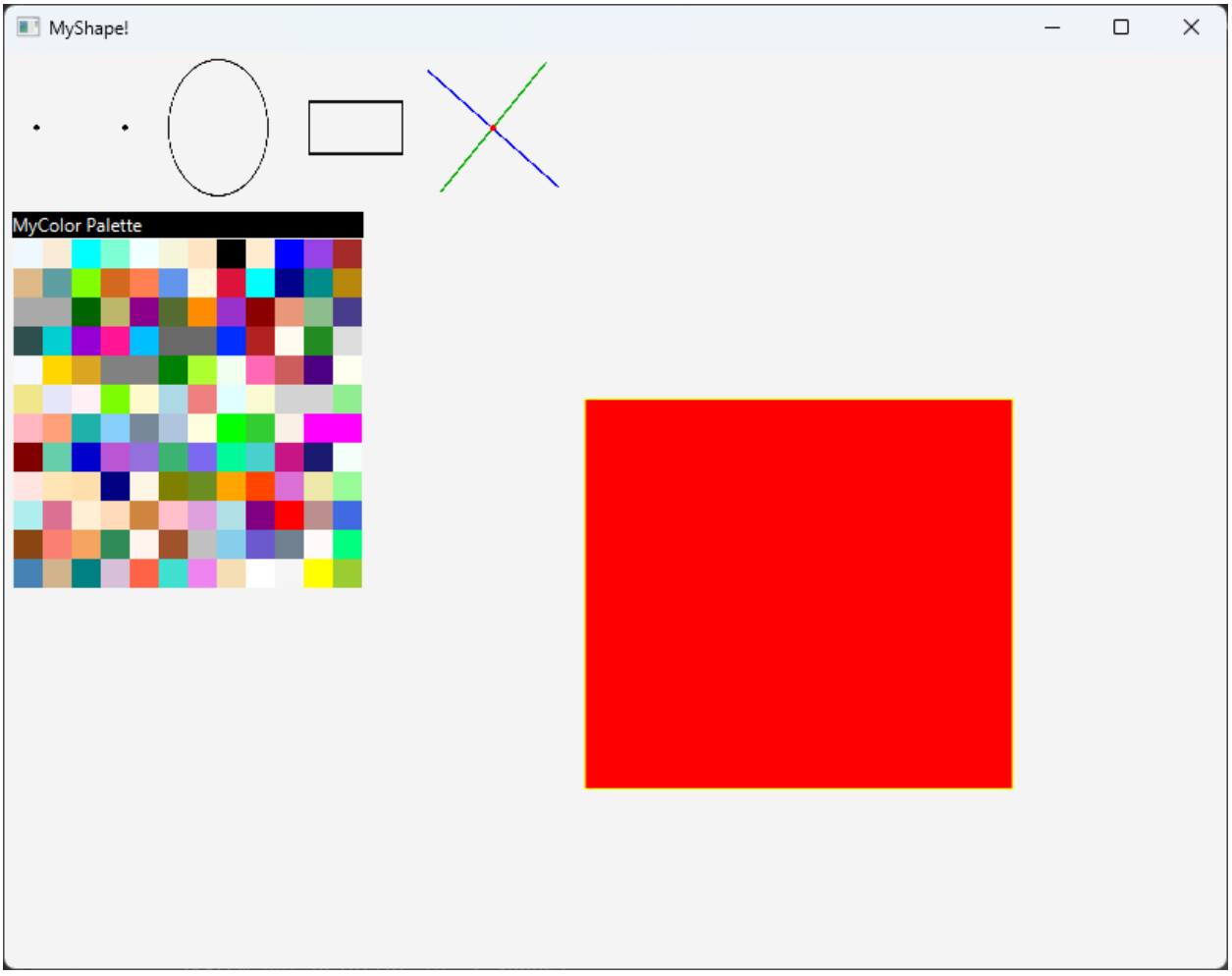

MyRectangle
Top Left Corner Point
0
x-Coordinate as fraction of canvas width
0.25
y-Coordinate as fraction of canvas height
Width
0.75
As fraction of canvas width
Height
1
As fraction of canvas height
OK
Cancel

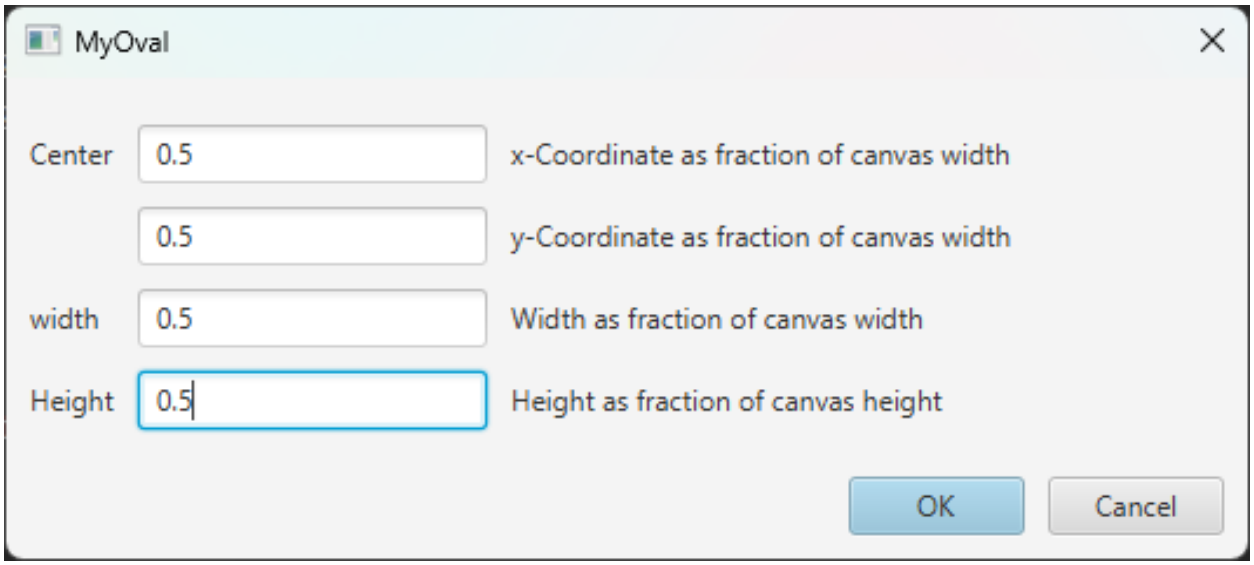

MyShape!
MyColor Palette

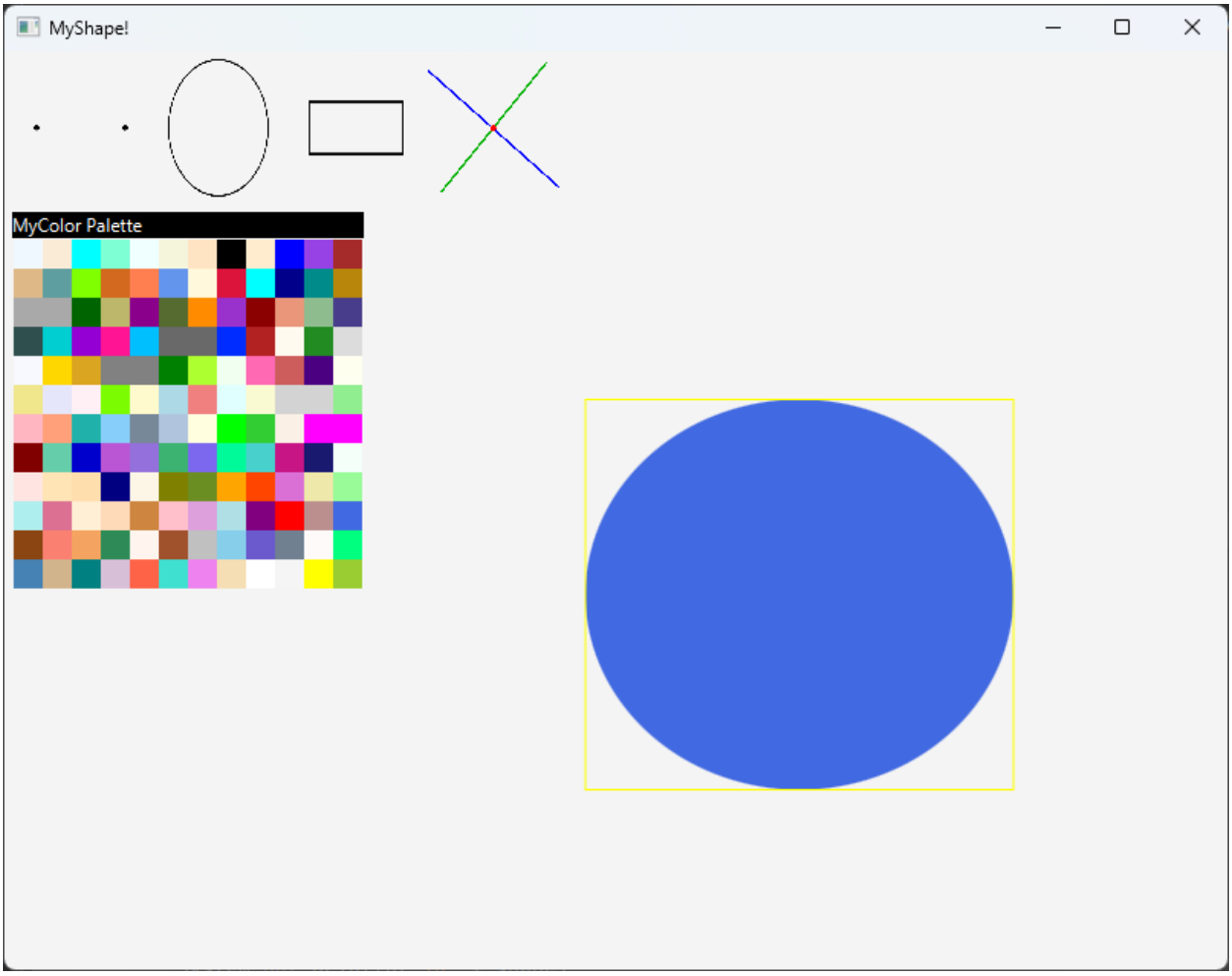

Intersection of 2 MyShape Objects
Draw the intersection of the last two MyShape Objects)
OK
Cancel

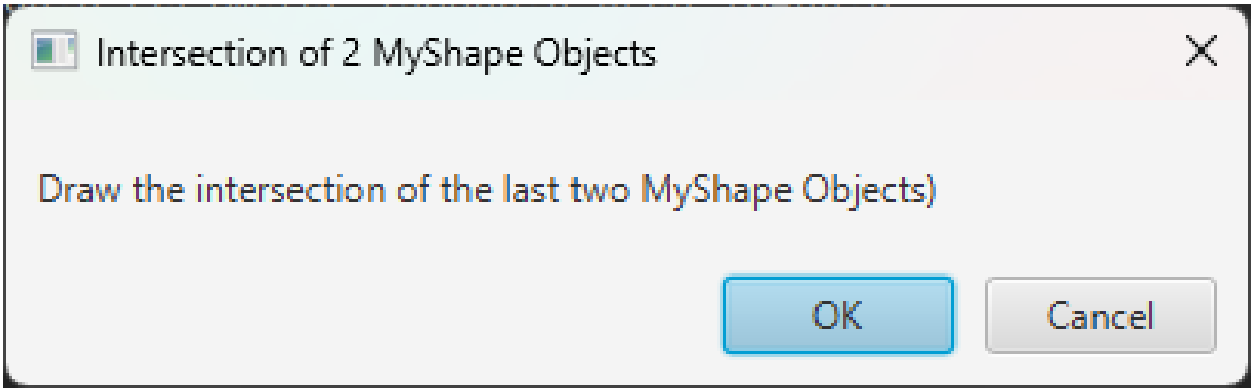

MyShape!
MyColor Palette

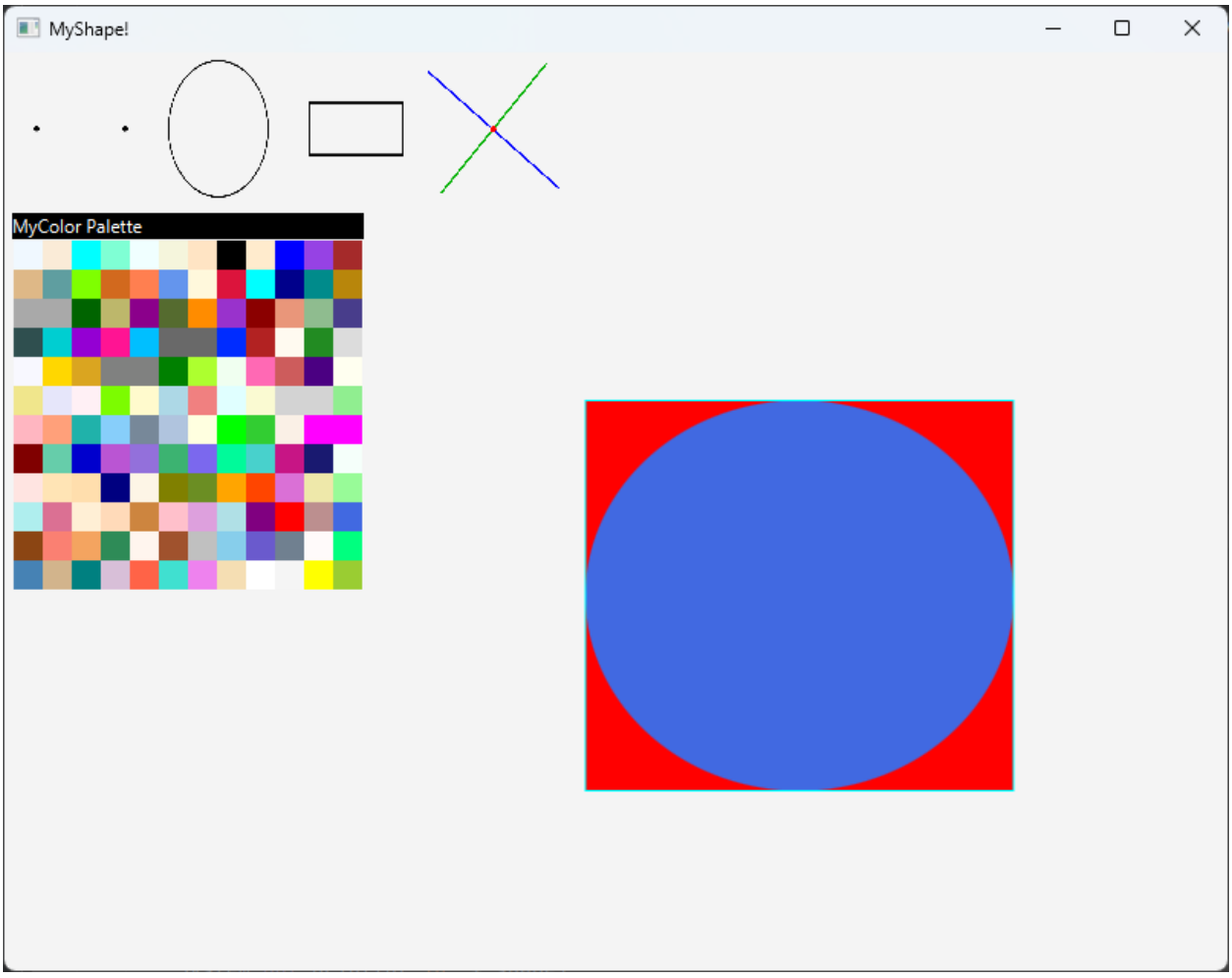

MyRectangle
Top Left Corner Point
0.25
x-Coordinate as fraction of canvas width
0.25
y-Coordinate as fraction of canvas height
Width
0.5
As fraction of canvas width
Height
0.5
As fraction of canvas height
OK
Cancel


MyShape!
MyColor Palette


MyOval
Center
0.5
x-Coordinate as fraction of canvas width
0.5
y-Coordinate as fraction of canvas width
width
0.5
Width as fraction of canvas width
Height
0.5
Height as fraction of canvas height
OK
Cancel


MyShape!
MyColor Palette


Intersection of 2 MyShape Objects
Draw the intersection of the last two MyShape Objects)
OK
Cancel


MyShape!
MyColor Palette