CSc 22100 HW 2

Leji Li

1. The Problem

This assignment is based on the the pervious assignment and adds new classes and interfaces.

New classes:

MyRectangle: This class inherits **MyShape** class. This class defines a rectangle that has a height *h* and width *w*, and centering at point(*x, y*). This class should include the following methods:

- getters and setters for width, height, perimeter and area.
- *toString* method shows the **MyRectangle** object's width, height, perimeter and area.
- *draw* method that draws the retangle on a GraphicsContext object. The *draw* method may draw a rectangle either with color filled or just stroked.

MyOval: This class inherits **MyShape** class and may use **MyRectangle** class. This class is defined by an ellopse inscribed in a rectangle of height *h* and width *w*, centered at point (*x, y*). This class should includes the following methods:

- *getPerimeter, getArea* and *toString.*
- *draw* method. Draw an ellipse that has a height of *h* and width of *w.*

New Interfaces:

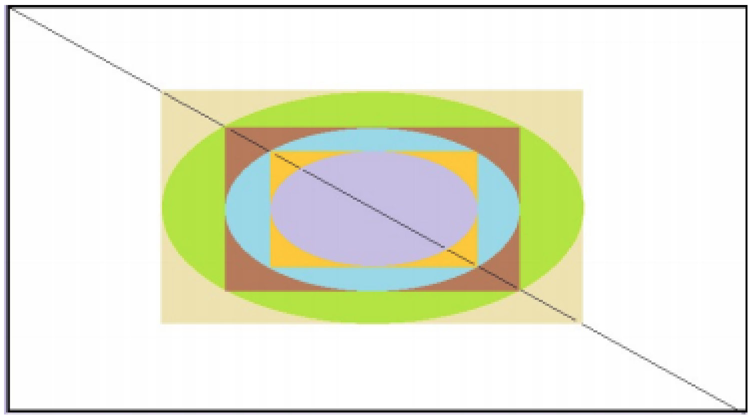
MyPoint: This interface should has the following methods:

- *getPoint, setPoint* can return and set a point (*x, y*).
- *moveTo* moves point (*x, y*) to point (*x+x', y+y'*).
- *distanceTo* calculates the current point to another point.

MyShapePosition: This interface extends interface **MyPoint** and has the following methods:

- *getMyBoundingBox*: return the bounding box of a shape object.
- *doOverlap*: return true if two object overlaps or false otherwise.

The second part of this assignment is to draw the graphic below using the classes above.



Picture should be drawed

2. Solution Methods

MyShape:

This class implements two interfaces, **MyPoint** and **MyShapePosition**. Since all the shape classes inherit this class, implementing the methods on those two interfaces here is more convenient than implementing on the sub-classes.

distanceTo: this method calculates and returns the distance between the current point to another point.

```java
@Override
public double distanceTo(MyPoint point) {
 double[] otherPoint = point.getPoint();
 return Math.sqrt(Math.pow(this.x - otherPoint[0], 2) + Math.pow(this.y - otherPoint[1], 2));
}
```

I pass in the other point as a **MyPoint** object so that this method works between any two classes that implement this interface, instead of just for **MyShape** and its sub-classes. The reason is that, to use this method, a class must implements the **MyPoint** interface, which is that a obejct of that class can be treated as a **MyPoint** object. If the method accepts the **MyPoint** object as the pass-in, this method can be used universally.

doOverlap: this method will check does the pass-in object overlap with the current object or not.

```java
@Override
public boolean doOverlap(MyShapePosition other) {
 if(distanceTo(other) == 0){
 return true;
 }

 MyRectangle myBox = getMyBoundingBox();
 MyRectangle otherBox = other.getMyBoundingBox();
 if((Math.abs(myBox.getWidth() + otherBox.getWidth())/2 < Math.abs(myBox.getX() - otherBox.getX())) &&
  (Math.abs(myBox.getHeight() + otherBox.getHeight())/2 < Math.abs(myBox.getY() - otherBox.getY()))){
  return false;
  }

  return true;
}
```

Similar to the *distanceTo* method, this method accept the **MyShapePosition** object as the pass-in, so that the object of any class implements this interface can be passed in. Inside of this method, I check the center point of two object is equal or not, if equal, then the two objects must overlap. Otherwise, check for the bounding boxes.

Two rectangles are considered not overlaped, should meet these two conditions:

1. the distance between two xs should be larger than half of the sum of their width.
2. the distance between two ys should be larger than half of hte sum of their height.

MyRectangle:

setPerimeter: this method allows setting a new perimeter to a **MyRectangle** object. But since a **MyRectangle** object is defined by the width and length, when the perimeter changes, the width and the height should also change along with the perimeter.

```java
public void setPerimeter(double perimeter) {
 double perimeterOrigin = getPerimeter();
 double ratio = perimeter / perimeterOrigin;
 this.h *= ratio;
 this.w *= ratio;
}
```

I keep the ratio of the modified **MyRectangle** object the same as the original one.

setArea: this method allows setting a new area to a **MyRectangle** object. Similar to the *setPerimeter* method, when changing the area, the width and height should also change.

```java
public void setArea(double area) {
 double areaOrigin = getArea();
 double ratio = Math.sqrt(area/ areaOrigin);
 this.h *= ratio;
 this.w *= ratio;
}
```

The solution is also similar to the *setPerimeter* method, let the new **MyRectangle** object have the same ratio as the original one.

draw: this method draws a rectangle on the **Canvas** object. Using the *fillRect* method is a filled rectangle is wanted or *strokeRect* method to draw the border of the rectangle.

getMyBoundingBox: this method overrides the super's method. In this **MyRectangle** class, this method will return itself.

MyOval:

getInnerBox: I add this method so that I can find a rectangle that having the same width/height ratio but smaller size that can fit into the current ellipse.

```
public MyRectangle getInnerBox(){
 return new MyRectangle(outerBox.getWidth()/Math.sqrt(2),
 outerBox.getHeight()/Math.sqrt(2),
 outerBox.getX(), outerBox.getY(),
 outerBox.getColor());
}
```

draw: this method uses *fillOval* method to draw a filled ellipse on the **Canvas** object.

Main:

The key method is draw all the graphs on a **Canvas** object.

First we should know the height and width of the outest filled rectangle.

```
double graphW = w, graphH = h;
double boxRatio = w / h;
if(ratio < boxRatio){
 graphW = ratio * h;
} else if(ratio > boxRatio){
 graphH = w / ratio;
}
```

Here the *graphW* and *graphH* are the width and the height of the outest rectangle, and the ratio is the width/height ratio of the showing graph that the user wants. By comparing the ratio of the window, which is the boxRatio here, and the ratio of the desired graph, we could know the following:

- if the ratio < boxRatio, which is desired_width/desired_height < window_width/window_height, that means either desired_width < window_width or desired_height > window_height. Both of the cases finally let us use the ratio*h as the width of desired rectangle and h be the height of the desired rectangle.

- if the ratio > boxRatio, similarly, the w is the width of the desierd rectangle and w/ratio is the height of the desired rectangle.

```java
for (int i = 0; i < 3; i++) {
 MyRectangle rectangle = null;
 int length = shapeList.size();
 if(length == 0){
 rectangle = new MyRectangle(0.9*graphW, 0.9*graphH, w/2, h/2, MyColor.Light
Pink);
 }
 else{
 rectangle = ((MyOval)(shapeList.get(length - 1))).getInnerBox();
 }
```

for the rest, we can easily use a loop to put all the rectangles and ellipses into an **ArrayList** object and than draw them on the canvas.

3. Codes developed

Main.java

```java
package com.demo;

import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Modality;
import javafx.stage.Stage;
import shapes.*;

import java.util.ArrayList;

public class Main extends Application implements EventHandler<ActionEvent>
{

 private static double MIN_WIDTH = 200d;
 private static double MIN_HEIGHT = 140d;
 private TextField wTextField, hTextField, ratioTextField;
 private double currentWidth, currentHeight, ratio;
 private Scene popScene;
 private Pane popHolder;
 private Canvas drawCanvas;

 public static void main(String[] args) {
 // write your code here
 launch(args);
 // the following statements used to test the overlap method.
 MyOval e1 = new MyOval(20, 10, 50, 50, MyColor.LightPink);
 MyOval e2 = new MyOval(20, 10, 20, 10, MyColor.LightPink);
 MyRectangle r1 = new MyRectangle(20, 10, 40, 50, MyColor.LightPink);
 System.out.println("e1 is " + e1.toString());
 System.out.println("e2 is " + e2.toString());
 System.out.println("r1 is " + r1.toString());
 System.out.println("Does e1 overlap e2? : " + e1.doOverlap(e2));
 System.out.println("Dose e2 overlap r1? : " + e2.doOverlap(r1));
 }

 @Override
 public void start(Stage primaryStage) throws Exception {
 // Use grid pane to layout the main stage
 // Use a pop up window to show the shape graphics
 // the main stage should be able to accept the width, height and the numbe
r of sides of the polygon from user
 GridPane root = new GridPane();
 root.setAlignment(Pos.CENTER);
root.setHgap(10);
root.setVgap(10);
root.setPadding(new Insets(25,25,25,25));

// use Text describe the view
Text title = new Text("Draw Shapes");
title.setFont(Font.font(20));
root.add(title, 0, 0, 2, 1);
// use Label for the description for better warping
Label description = new Label("Please input the width, height of the pop up window.");
description.setWrapText(true);
root.add(description, 0, 1, 2,1);

// use TextField to accept the user inputs
// use Label to give hints to user
Label wLabel = new Label("Width");
wTextField = new TextField();
root.add(wLabel, 0, 2);
root.add(wTextField, 1, 2);

Label hLabel = new Label("Height");
hTextField = new TextField();
root.add(hLabel, 0, 3);
root.add(hTextField,1,3);

Label ratioLabel = new Label("a/b");
ratioTextField = new TextField();
root.add(ratioLabel, 0, 4);
root.add(ratioTextField,1,4);

// A button
Button btn = new Button("Draw");
HBox btnBox = new HBox(10);
btnBox.setAlignment(Pos.CENTER);
btnBox.getChildren().add(btn);
root.add(btnBox, 0, 5, 2, 1);

```

```java
btn.setOnAction(this);

Scene scene = new Scene(root, 320, 275);
primaryStage.setScene(scene);
primaryStage.setTitle("Draw Shapes");
 primaryStage.show();
 }

 /**
 * This function will return a canvas that have the graph drawn
 * @param w the width of the canvas
 * @param h the height of the canvas
 * @return
 */
 public Canvas getShowCanvas(double w, double h){
 // keep the height and width properties
 // use for the resize
 currentHeight = h;
 currentWidth = w;

 double graphW = w, graphH = h;
 double boxRatio = w / h;
 if(ratio < boxRatio){
 graphW = ratio * h;
 } else if(ratio > boxRatio){
 graphH = w / ratio;
 }

 Canvas canvas = new Canvas(w, h);
 GraphicsContext gc = canvas.getGraphicsContext2D();

 (new MyRectangle(w, h, w/2, h/2, MyColor.Black)).draw(gc, false);
 ArrayList<MyShape> shapeList = new ArrayList<MyShape>(7);
 MyColor[] colors = {
 MyColor.Wheat,
 MyColor.OliveDrab,
 MyColor.FireBrick,
 MyColor.SkyBlue,
 MyColor.Yellow,
 MyColor.Plum
 };
 for (int i = 0; i < 3; i++) {
 MyRectangle rectangle = null;
 int length = shapeList.size();
 if(length == 0){
 rectangle = new MyRectangle(0.9*graphW, 0.9*graphH, w/2, h/2, MyColor.LightPink);
 }
 else{
 rectangle = ((MyOval)(shapeList.get(length - 1))).getInnerBox();
 }
 rectangle.setColor(colors[i*2]);
 shapeList.add(rectangle);
 shapeList.add(new MyOval(rectangle, colors[i*2+1]));
 }

 shapeList.add(new MyLine(0, 0, w, h, MyColor.Black));

 for(MyShape shape : shapeList){
 shape.draw(gc);
 }

 return canvas;
 }

 @Override
 public void handle(ActionEvent event) {
 double w, h;

 w = getValueFromInput(wTextField, MIN_WIDTH, MIN_WIDTH);
 h = getValueFromInput(hTextField, MIN_HEIGHT, MIN_HEIGHT);
 ratio = getValueFromInput(ratioTextField, 0, 2);

 final Stage dialog = new Stage();
 dialog.initModality(Modality.NONE);
 popHolder = new Pane();
```

```java
drawCanvas = getShowCanvas(w, h);
popHolder.getChildren().add(drawCanvas);
popScene = new Scene(popHolder, w, h);

// when the window size changes
// resize the draw shapes
popScene.widthProperty().addListener(new ChangeListener<Number>() {
@Override
public void changed(ObservableValue<? extends Number> observable, Number
oldValue, Number newValue) {
popHolder.getChildren().clear();
popHolder.getChildren().add(getShowCanvas(newValue.doubleValue(), current
Height));
wTextField.setText(String.valueOf(newValue.doubleValue()));
}
});

popScene.heightProperty().addListener(new ChangeListener<Number>() {
@Override
public void changed(ObservableValue<? extends Number> observable, Number
oldValue, Number newValue) {
popHolder.getChildren().clear();
popHolder.getChildren().add(getShowCanvas(currentWidth, newValue.doubleVa
lue()));
hTextField.setText(String.valueOf(newValue.doubleValue()));
}
});

dialog.setScene(popScene);
dialog.setTitle("Finish!");
dialog.show();
}

/**
* this function
* @param input the input TextField object
* @param condition the smallest value of a correct input should be
* @param defaultValue the default value.
```

```
  * @return the value of text in the TextField object if user has the correct input
  * default value if the input cannot be recognized the input
  * or the input value is less than the condition value
  */
  private double getValueFromInput(TextField input, double condition, double defaultValue){
  double result;
  try {
  result = Double.parseDouble(input.getText());
  if( result < condition){
  result = defaultValue;
  }
  input.setText(String.valueOf(result));
  } catch (Exception e){
  result = defaultValue;
  input.setText(String.valueOf(result));
  }
  return result;
  }
}
```

MyPoint.java

```
package shapes;

public interface MyPoint {

 /**
 * return the center of this shape
 * @return the 0 is x, 1 is y ...
 */
 public double[] getPoint();

  /**
  *
  * @param point a double array, whose 0 is x, 1 is y...
  */
  public void setPoint(double[] point);

```

```java
 /**
 * move a point(x, y) to (x + deltaX, y + deltaY)
 * @param deltaX the distance of movement in x-axis
 * @param deltaY the distance of movement in y-axis
 */
 public void moveTo(double deltaX, double deltaY);

 /**
 *
 * @param otherPoint a double array, whose 0 is x, 1 is y ...
 * @return the distance between the hosting point and the given point
 */
 public double distanceTo(MyPoint otherPoint);
}
```

MyShapePosition.java:

```java
package shapes;

public interface MyShapePosition extends MyPoint {

 public MyRectangle getMyBoundingBox();
 public boolean doOverlap(MyShapePosition other);
}
```

MyShape.java:

```java
package shapes;


import javafx.scene.canvas.GraphicsContext;

import java.awt.*;

public abstract class MyShape extends Object implements MyShapePosition {
 private double x;
  private double y;
  private MyColor color;

  public MyShape(double x, double y, MyColor color){
  this.x = x;
  this.y = y;
  this.color = color;
  }
  public MyShape(double x, double y){
  this(x, y, MyColor.Black);
  }
  public MyShape(MyColor color){
  this(0, 0, color);
  }

  public MyShape(){
  this(0, 0);
  }

  public double getX() {
  return x;
  }

  public void setX(double x) {
  this.x = x;
  }

  public double getY() {
  return y;
  }

  public void setY(double y) {
  this.y = y;
  }

  public MyColor getColor() {
  return color;
  }

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

```

```
public abstract void draw(GraphicsContext gc);

@Override
public String toString(){
return "Class MyShapeis the hierarchy’s superclass and inherits the Java class Object. An\n" +
"implementation of the class defines a point (x, y) and the color of the shape. ";
}

@Override
public MyRectangle getMyBoundingBox() {
return new MyRectangle(0, 0, this.x, this.y, this.color);
}

@Override
public boolean doOverlap(MyShapePosition other) {
if(distanceTo(other) == 0){
return true;
}

MyRectangle myBox = getMyBoundingBox();
MyRectangle otherBox = other.getMyBoundingBox();
// checking the center point and the edges
// two rectangles are considered not touching each other should meet the two conditions:
// the distance between two xs should be larger than half of the sum of their width.
// the distance between two ys should be larger than half of hte sum of their height.
if((Math.abs(myBox.getWidth() + otherBox.getWidth())/2 < Math.abs(myBox.getX() - otherBox.getX())) &&
(Math.abs(myBox.getHeight() + otherBox.getHeight())/2 < Math.abs(myBox.getY() - otherBox.getY()))){
return false;
}

return true;
}

```

```java
@Override
public double[] getPoint() {
return new double[]{this.x, this.y};
}

@Override
public void setPoint(double[] point) {
this.x = point[0];
this.y = point[1];
}

@Override
public void moveTo(double deltaX, double deltaY) {
this.x += deltaX;
 this.y += deltaY;
 }

 @Override
 public double distanceTo(MyPoint point) {
 double[] otherPoint = point.getPoint();
 return Math.sqrt(Math.pow(this.x - otherPoint[0], 2) + Math.pow(this.y -
otherPoint[1], 2));
 }
}
```

MyLine.java

```java
package shapes;

import javafx.scene.canvas.GraphicsContext;

public class MyLine extends MyShape {
 private double x1, y1;
 private double x2, y2;

 public MyLine(double x1, double y1, double x2, double y2){
 this.x1 = x1;
 this.x2 = x2;
 this.y1 = y1;
 this.y2 = y2;
}

public MyLine(double x1, double y1, double x2, double y2, MyColor color){
this(x1, y1, x2, y2);
setColor(color);
}

public double getLength(){
return Math.sqrt(Math.pow(this.x2 - this.x1, 2) + Math.pow(this.y2 -
this.y1,2));
}

public double get_xAngle(){
return Math.toDegrees(Math.atan2((this.y2 - this.y1), (this.x2 -
this.x1)));
}

@Override
public String toString(){
return "MyLine{length=" + getLength()+ ", _xAngle=" + get_xAngle() + "}";
}

@Override
public void draw(GraphicsContext gc) {
gc.setStroke(getColor().toFXPaintColor());
gc.strokeLine(this.x1, this.y1, this.x2, this.y2);
}

/**
* the 0 and 1 is the first point, 2 and 3 is the second one
* @return a double array holds two points
*/
@Override
public double[] getPoint() {
return new double[]{this.x1, this.y1, this.x2, this.y2};
}

@Override
```

```java
 public MyRectangle getMyBoundingBox() {
 return new MyRectangle(Math.abs(this.x1 - this.x2), Math.abs(this.y1 - this.y2), (this.x1 + this.x2)/2, (this.y1 + this.y2)/2, this.getColor());
 }

 @Override
 public void setPoint(double[] point) {
 this.x1 = point[0];
 this.y1 = point[1];
 this.x2 = point[2];
 this.y2 = point[3];
 }

 @Override
 public void moveTo(double deltaX, double deltaY) {
 this.x1 += deltaX;
 this.x2 += deltaX;
 this.y1 += deltaY;
 this.y2 += deltaY;
 }
}
```

MyPolygon.java

```java
package shapes;

import javafx.scene.canvas.GraphicsContext;

public class MyPolygon extends MyShape {
 private int N;
 private double r;

 public MyPolygon(double x, double y, int N, double r){
  super(x, y);
  this.N = N;
  this.r = r;
  }

  public MyPolygon(double x, double y, int N, double r, MyColor color){
  this(x, y, N, r);
setColor(color);
}

public double getArea(){
return this.N * Math.pow(this.r, 2) * Math.tan(Math.toRadians(Math.PI / this.N));
}

public double getPerimeter(){
return this.getSide() * this.N;
}

public double getAngle(){
return (180d * this.N - 360d) / this.N;
}

public double getSide(){
return 2d * this.r * Math.tan(Math.toRadians(Math.PI / this.N));
}

public static double getInradiusByCircumradius(double R, int N){
return R * Math.cos(Math.toRadians(180d / N));
}

public static double getCircumradiusByInradius(double r, int N){
return r / Math.cos(Math.toRadians(180d / N));
}

@Override
public String toString() {
return "MyPolygon{" +
"side=" + getSide() +
", angle=" + getAngle() +
", perimeter=" + getPerimeter() +
", area=" + getArea() +
'}';
}

```

```
@Override
public void draw(GraphicsContext gc){
draw(gc, false);
}

public void draw(GraphicsContext gc, boolean isFill) {
double R = getCircumradiusByInradius(this.r, this.N);
double innerAngle = 360d / this.N;
double Xs[] = new double[this.N];
double Ys[] = new double[this.N];

for (int i = 0; i < this.N; i++) {
Ys[i] = getY() + R * Math.sin(Math.toRadians(innerAngle * i - 90));
Xs[i] = getX() + R * Math.cos(Math.toRadians(innerAngle * i - 90));
}

if(isFill){
gc.setFill(getColor().toFXPaintColor());
gc.fillPolygon(Xs, Ys, this.N);
} else {
gc.setStroke(getColor().toFXPaintColor());
gc.strokePolygon(Xs, Ys, this.N);
}
}

@Override
public MyRectangle getMyBoundingBox() {
double R = getCircumradiusByInradius(this.r, this.N);
return new MyRectangle(R, R, this.getX(), this.getY(), this.getColor());
}
}
```

MyRectangle.java

```
package shapes;

import javafx.scene.canvas.GraphicsContext;

public class MyRectangle extends MyShape {
```

```java
double h, w;

/**
*
 * @param height
 * @param width
 * @param x
 * @param y
 * @param color
 */
 public MyRectangle(double width, double height, double x, double y, MyColor color){
 super(x, y, color);
 this.h = height;
 this.w = width;
 }

 public MyRectangle(double height, double width, MyColor color){
 this(height, width, 0, 0, color);
 }

 public double getHeight() {
 return h;
 }

 public void setHeight(double h) {
 this.h = h;
 }

 public double getWidth() {
 return w;
 }

 public void setWidth(double w) {
 this.w = w;
 }

 public double getPerimeter() {
```

```java
return 2 * (this.h + this.w);
}

public void setPerimeter(double perimeter) {
double perimeterOrigin = getPerimeter();
double ratio = perimeter / perimeterOrigin;
this.h *= ratio;
this.w *= ratio;
}

public double getArea() {
return this.h * this.w;
}

public void setArea(double area) {
double areaOrigin = getArea();
double ratio = Math.sqrt(area/ areaOrigin);
this.h *= ratio;
this.w *= ratio;
}

public String toString(){
return "MyRectangle{" +
"width=" + this.w +
", height=" + this.h +
", perimeter=" + getPerimeter() +
", area=" + getArea() +
"}";
}

@Override
public void draw(GraphicsContext gc) {
draw(gc, true);
}

public void draw(GraphicsContext gc, boolean isFill){
if(isFill){
gc.setFill(this.getColor().toFXPaintColor());
gc.fillRect(getX() - this.w/2, getY() - this.h/2, this.w, this.h);
} else {
gc.setStroke(this.getColor().toFXPaintColor());
gc.strokeRect(getX() - this.w/2, getY() - this.h/2, this.w, this.h);
}
}

@Override
public MyRectangle getMyBoundingBox() {
return this;
}
}
```

MyOval.java

```java
package shapes;

import javafx.scene.canvas.GraphicsContext;

public class MyOval extends MyShape {
 private MyRectangle outerBox;

 public MyOval(double width, double height, double x, double y, MyColor colo
r){
 super(x, y, color);
 outerBox = new MyRectangle(width, height, x, y, color);
 }

 public MyOval(MyRectangle box, MyColor color){
 this(box.getWidth(), box.getHeight(), box.getX(), box.getY(), color);
 }

 public double getPerimeter(){
 double a = outerBox.getWidth() / 2;
 double b = outerBox.getHeight() / 2;
 return Math.PI * (3*(a+b) - Math.sqrt((3*a+b)*(a+3*b)));
 }

 public MyRectangle getInnerBox(){
```

```java
return new MyRectangle(outerBox.getWidth()/Math.sqrt(2), outerBox.getHeight()/Math.sqrt(2), outerBox.getX(), outerBox.getY(), outerBox.getColor());
}

public double getArea(){
return Math.PI * outerBox.getWidth() * outerBox.getHeight() / 4;
}

public String toString(){
return "MyOval{"+
"width="+ outerBox.getWidth()+
", height=" + outerBox.getHeight() +
", perimeter=" + getPerimeter() +
", area=" + getArea() +
"}";
}

@Override
public void draw(GraphicsContext gc) {
gc.setFill(getColor().toFXPaintColor());
gc.fillOval(getX() - outerBox.getWidth()/2, getY() -
outerBox.getHeight()/2, outerBox.getWidth(), outerBox.getHeight());
}

@Override
public MyRectangle getMyBoundingBox() {
return this.outerBox;
}

}
```

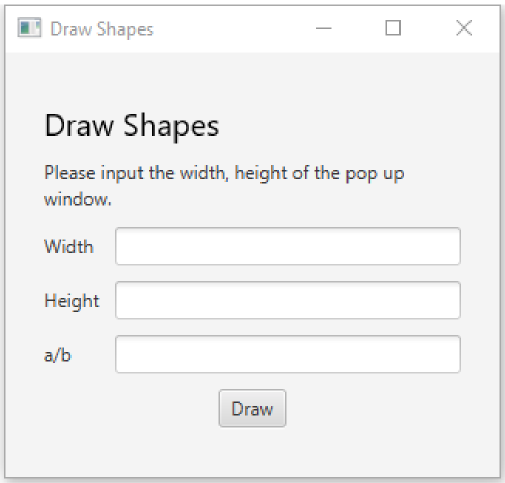
MyColor.java

```java
 package shapes;

import java.awt.*;

public enum MyColor {
 Red(255, 0, 0),
 FireBrick(178,34,34),
 IndianRed(205,92,92),
 LightCoral(240,128,128),
  LightPink(255,182,193),
  Pink(255,192,203),
  Green(0,255,0),
  LightGreen(144,238,144),
  PaleGreen(152,251,152),
  OliveDrab(85,107,47),
  MediumAquamarine(0,250,154),
  Turquoise(64,224,208),
  Blue(0,0,255),
  DarkBlue(0,0,139),
  RoyalBlue(65,105,225),
  SkyBlue(135,206,235),
  Azure(240,255,255),
  White(255,255,255),
  Black(0,0,0),
  Gray(128,128,128),
  LightGray(211,211,211),
  Yellow(255,255,0),
  Wheat(255,231,186),
  Plum(255,187,255);
  private int r, g, b;

  private MyColor(){
  this(0, 0,0);
  }

  private MyColor(int r, int g, int b){
  setColor(r, g, b);
  }

  public Color toAWTColor(){
  return new Color(r, g, b);
  }

  public javafx.scene.paint.Color toFXPaintColor(){
  return javafx.scene.paint.Color.rgb(r, g, b);
```

```java
}

public void setColor(int hex){
this.r = (hex & 0xFF0000) >> 16;
this.g = (hex & 0xFF00) >> 8;
this.b = hex & 0xFF;
}

public void setColor(int r, int g, int b){
this.r = r;
this.g = g;
this.b = b;
}

public int getHexColor(){
return ((0xFF0000 & (r << 16)) | (0x00FF00 & (g << 8)) |b);
}
}
```

4. Outputs

The main window of the application, which allows the user types in the width and height of the pop-up window, which is used to show the result graph.


Draw Shapes

Draw Shapes

Please input the width, height of the pop up window.

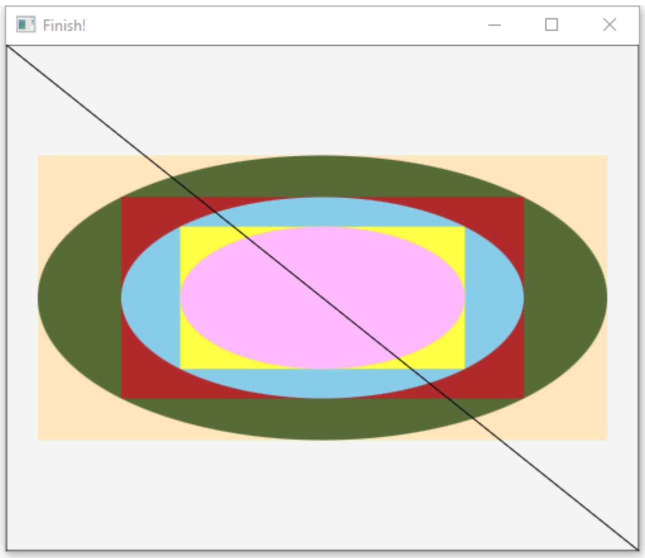
Width

Height

a/b

Draw

Main window



Result graph

Overlap test:

```
e1 is MyOval{width=20.0, height=10.0, perimeter=48.44210548835644, area=157.07963267948966}
e2 is MyOval{width=20.0, height=10.0, perimeter=48.44210548835644, area=157.07963267948966}
r1 is MyRectangle{width=20.0, height=10.0, perimeter=60.0, area=200.0}
Does e1 overlap e2? : false
Dose e2 overlap r1? : true
```

output of the overlap test