CSC 22100: Exercise 2

Jorge L Roldan Roldan

July 12, 2018

Contents

List of Figures

1 Instructions

1.1 Part 1: Amend hierarchy of Java Classes

Amend the hierarchy of Java classes in Exercise 1 as follows:

Polygon $is-a$ Shape;
Rectangle $is-a$ Shape;
Oval $is-a$ Shape;
Circle $is-a$ Oval;

1.2 Part 2: Interfaces

Interface ShapeInterface, interface PositionInterface, and interface ShapePositionInterface are specified in connection with the class hierarchy.

1.3 Part 3: ShapeInterface

Interface ShapeInterface includes appropriate abstract, static, and/or default methods that describe the intrinsic functions and behaviors of the specific objects in the class hierarchy, including:
a. getArea − describes the area of an object in the class hierarchy;
b. getPerimeter − describes the perimeter of an object in the class hierarchy

1.4 Part 4: PositionInterface

Interface PositionInterface includes appropriate abstract, static, and/or default methods that describe the positional functions and behaviors of the specific object types of the class hierarchy, including:

a. getPoint − returns the point (x, y);
b. moveTo − moves point (x, y) to point (x + Δx, y + Δy);
c. distanceTo − returns distance from point (x, y) to a point

1.5 Part 5: ShapePositionInterface

The abstract class Shape implements interface ShapePositionInterface which extends interface ShapeInterface and interface PositionInterface. Interface ShapePositionInterface includes appropriate abstract, static, and/or default methods that describe the functions and behaviors of the specific objects types of the class hierarchy, including:
a. getBoundingBox : returns the bounding rectangle of an object in the class
hierarchy; b. doOverlap : returns true if two objects in the class hierarchy overlap.

1.6 Part 6: Geometric output

Build a class Application that processes polymorphically the subclasses in the hierarchy to draw the geometric object shown:

2 Solution Method

2.1 Interfaces

First, the three interfaces coded and their final result are shown in figures 1, 2, 3 and 4. The method signature they contained are specified in the instruction section above. Interface ShapePosition-Interface extends the other two interfaces which is shown is line 5 using the keyword **extends**.
In order to implement the method doOverlap(), the method intercept from the Shape class from javafx was used, two myClass arguments were passed to doOverlap() and the private method determineSubclass(line 11 - 35) was used to find out the subclass of each of the myClass objects passed to construct their respective objects from the Shape class. Doing this, the method Shape.intersect was used to determine whether two objects were overlapping.

2.2 Classes

For the most part, the classes remained almost the same from exercise 1. A significant difference is that myShape class (figure 5 and 6) is abstract since it does not implement all the abstract methods of the interfaces it implements. Some of the new methods are moveTo and distanceTo, whose requirements are specified in the instruction section.

Another method included in this exercise is the getBoundingBox and getPoint. These methods were implemented for classes myPolygon, myRectangle, and myOval in figures 8, 9, and 12.In order to implement getPoint, since it required to return a point data type with both x and y coordinate, a Point Class was created as shown in figure 14.

2.3 Polymorphism Test

In the codes from figure 15,16, 17, the application built processes polymorphically the subclasses in the hierarchy to draw the geometric object shown in the outputs sections.

2.4 doOverlap Test Codes

Finally an application was built to test the performance of doOverlap method. This code is include in figures 18 ,19, 20.

3 Codes developed

3.1 Interfaces Codes

```
1  public interface ShapeInterface {
2
3      // area method
4      public double getArea();
5
6      // perimeter method
7      public double getPerimeter();
8  }
9
```

Figure 1: Shape interface

```java
public interface PositionInterface {

    // getPoint method
    Point getPoint();

    // moveTo method
    void moveTo(double newX, double newY);

    // distanceTo method
    double distanceTo(double targetX,double  targetY);       // MUST FIX RETURN
}
```

Figure 2: Position Interface

```java
import javafx.scene.paint.Color;
import javafx.scene.shape.*;

public interface ShapePositionInterface
        extends ShapeInterface, PositionInterface {

    // getBoundingBox method
    myRectangle getBoundingBox(double canvasWidth, double canvasHeight,Color color1);

    // determine appropiate subclass of myShape
    private Shape determineSubclass(myShape myShape1){
        Shape shapeTest1;

        if( myShape1 instanceof myCircle){
            myCircle myCircle1 = (myCircle) myShape1;
            shapeTest1 = new Circle(myCircle1.getX(),myCircle1.getY(),myCircle1.getRadius());
        }
        else if(myShape1 instanceof myOval){
            myOval myOval1 = (myOval) myShape1;
            shapeTest1 = new Ellipse(myOval1.getX(),myOval1.getY(),myOval1.getAxisA(),
                                     myOval1.getAxisB());
        }
        else if(myShape1 instanceof myRectangle){
            myRectangle myRectangle1 = (myRectangle) myShape1;
            shapeTest1 = new  Rectangle(myRectangle1.getWidth(),myRectangle1.getHeight(),
                    myRectangle1.getWidth(),myRectangle1.getHeight());
        }
        else {
            myPolygon myPolygon1 = (myPolygon) myShape1;
            shapeTest1 = new Polygon(myPolygon1.getX(),myPolygon1.getX());
        }

        return shapeTest1;

    }
```

Figure 3: ShapePositionInterface

```java

    // doOverlap method
    default boolean doOverlap(myShape myShape1, myShape myShape2){

        Shape shapeTest1;
        shapeTest1 =  determineSubclass(myShape1);

        Shape shapeTest2;
        shapeTest2 =  determineSubclass(myShape2);

        Shape interceptedShapes = Shape.intersect(shapeTest1,shapeTest2);

        return (interceptedShapes.getBoundsInLocal().getWidth() != -1);
    }
}
```

Figure 4: ShapePositionInterface Part2

3.2 Classes Codes

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public abstract class  myShape implements ShapePositionInterface{

    // instance variable
    private double x;
    private double y;
    private Color strokeColor;

    //constructor
    public myShape(double x, double y, Color strokeColor){
        this.x = x;
        this.y = y;
        this.strokeColor = strokeColor;
    }

    // get methods
    public double getX(){ return this.x; }

    public double getY(){
        return this.y;
    }

    // getColor method
    public Color getColor() { return this.strokeColor; }

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

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

    public void setColor(Color strokeColor){ this.strokeColor  = strokeColor; }

```

Figure 5: myShape Part1

```java
    // shift methods
    public void shiftX(double deltaX) { this.x = this.x + deltaX; }

    public void shiftY(double deltaY) { this.y = this.y + deltaY; }

    // toString method
    @Override
    public String toString(){
        return String.format("%s: %n%s %.1f%n%s %.1f%n%s%s",
                "shape","x:",getX(),"y:",getY(),"color: ",getColor());
    }

    // draw method
    public void draw(GraphicsContext gc, double canvasWidth, double canvasHeight){
        // set the color for the current arc
        gc.setFill(strokeColor);
        gc.fillRect( x: x - canvasWidth/2, y: y - canvasHeight/2,canvasWidth,canvasHeight);
    }

    // moveTo method
    public void moveTo(double newX, double newY){
        setX(getX() + newX);
        setY(getY() + newY);
    }

    // distanceTo method
    public double distanceTo(double targetX,double  targetY){
        return  Math.sqrt(Math.pow(getX()-targetX,2) + Math.pow(getY()-targetY,2));
    }
}
```

Figure 6: myShape part2

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public class myPolygon extends myShape {
    private double radius;
    private double[] xPoints;
    private double[] yPoints;
    private int nPoints;

    // constructor
    public myPolygon(double x, double y, Color strokeColor,
                     double radius, double[] xPoints, double[] yPoints,
                     int nPoints){
        super(x,y,strokeColor);

        this.radius = radius;
        this.xPoints = xPoints;
        this.yPoints = yPoints;
        this.nPoints = nPoints;
    }

    // getAngle() method
    public double getAngle() { return 180.0*(nPoints-2)/nPoints; }

    // getSide() method
    public double getSide(){
        double side = 0;
        side = Math.sqrt(Math.pow(xPoints[1]-xPoints[0],2) + Math.pow(yPoints[1]-yPoints[0],2));
        return side;
    }

    // getPerimeter() method
    public double getPerimeter(){...}

    // getArea() method
    public double getArea() { return (1.0/2)*nPoints*Math.pow(radius,2)*Math.sin(2*Math.PI/nPoints); }

```

Figure 7: myPolygon part1

```
58          double angleInRadians = Math.toRadians(angleInDegrees);
59
60          for(int counter = 0; counter < nPoints ; counter++ ){
61              xPoints[counter] = super.getX() + radius*Math.sin(counter*angleInRadians);
62              yPoints[counter] = super.getX() - radius*Math.cos(counter*angleInRadians);
63          }
64
65          System.out.println("x     y");
66          for(int counter = 0; counter < nPoints ; counter++ ){
67              System.out.printf("%.1f %.1f%n",xPoints[counter],yPoints[counter]);
68          }
69
70          // set the color for the current arc
71          gc.setFill(super.getColor());
72
73          // Draw Polygon
74          gc.strokePolygon(xPoints,yPoints,nPoints);
75          gc.fillPolygon(xPoints,yPoints,nPoints);
76      }
77
78      // getPoint method
79      public Point getPoint(){
80          Point point1 = new Point(getX(),getY());
81
82          return point1;
83      }
84      // getBoundingBox method
85      public myRectangle getBoundingBox(double canvasWidth, double canvasHeight,Color color1){
86          double y1 = yPoints[0];
87          double halfHeight = Math.sqrt(Math.pow(super.getX() - xPoints[0],2)
88                                  + Math.pow(super.getY()-yPoints[0],2));
89          myRectangle rectangle1 = new myRectangle(super.getX(),super.getY(),color1,
90                                  width: 2*halfHeight, height: 2*halfHeight);
91
92          return rectangle1;
93      }
94  }
```

Figure 8: myPolygon part2

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public class myRectangle extends myShape {
    private double width;
    private double height;

    // constructor
    public myRectangle(double x, double y, Color strokeColor, double width,double  height){
        super(x,y,strokeColor);

        this.width = width;
        this.height = height;
    }

    // get width method
    public double getWidth(){ return width;}

    // get height method
    public double getHeight(){ return height;}

    // ======================================
    // IMPLEMENTING METHODS FROM INTERFACES
    // ======================================
    // getPoint method
    public Point getPoint(){
        Point point1 = new Point( xPoint: (getX() + getWidth()/2)/2, yPoint: (getY()+ getHeight()/2)/2);
        return point1;
    }

    // getBoundingBox method
    public myRectangle getBoundingBox(double canvasWidth, double canvasHeight, Color color1){
        myRectangle rectangle1 = new myRectangle( x: super.getX() + width,super.getY(),
                color1,getWidth(),getHeight());

        return rectangle1;
    }

```

Figure 9: myRectangle part1

```java
    // area method
    public double getArea(){ return getWidth()*getHeight();}

    // getPerimeter() method
    public double getPerimeter(){return 2*(getHeight() + getWidth());}

    // draw method
    @Override
    public void draw(GraphicsContext gc,double canvasWidth, double canvasHeight) {
        // set the color for the current arc
        gc.setFill(super.getColor());

        // Draw Rectangle
        gc.strokeRect( x: super.getX() - canvasWidth/2, y: super.getY()- canvasHeight/2,width,height);
        gc.fillRect( x: super.getX() - canvasWidth/2, y: super.getY()- canvasHeight/2,width,height);
    }
}
```

Figure 10: myRectangle part2

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public class myOval extends myShape{
    private double axisA;
    private double axisB;

    // constructor
    public myOval(double x, double y, Color strokeColor, double axisA, double axisB){
        super(x,y,strokeColor);
        this.axisA = axisA;
        this.axisB = axisB;
    }

    // return axisA method
    public double getAxisA(){ return this.axisA; }

    // return axisB method
    public double getAxisB(){ return this.axisB; }

    @Override
    public void draw(GraphicsContext gc,double width, double height) {
        // set the color for the current arc
        gc.setFill(super.getColor());

        // Draw circle
        gc.strokeOval( x: super.getX() - width/2, y: super.getY()- height/2,
                w: 2*getAxisA(), h: 2*getAxisB());
        gc.fillOval( x: super.getX() - width/2, y: super.getY()- height/2,
                w: 2*getAxisA(), h: 2*getAxisB());
    }

```

Figure 11: myOval part1

```java
    // toString method
    @Override
    public String toString(){
        return String.format("%s%n%s%.1f%n%s%.1f%s%.1f%n", "Oval: "
              ,"RadiusX: ",getAxisA(),"RadiusY: ",getAxisB(),"area: ", getArea());
    }
    // ======================================
    // IMPLEMENTING METHODS FROM INTERFACES
    // ======================================
    // getPoint method
    public Point getPoint(){
        Point point1 = new Point( xPoint: (getX() + getAxisA())/2, yPoint: (getY()+getAxisB())/2);

        return point1;
    }

    // getBoundingBox method
    public myRectangle getBoundingBox(double canvasWidth, double canvasHeight,Color color1){
        myRectangle rectangle1 = new myRectangle(getX() ,getY() ,color1,
                                  width: 2*getAxisA(), height: 2*getAxisB());

        return rectangle1;
    }

    // perimeter method
    public double getPerimeter(){
        return 2*Math.PI*Math.sqrt((Math.pow(getAxisA(),2)+
                                    Math.pow(getAxisA(),2))/2);
    }

    // area method
    public double getArea(){ return Math.PI*axisA*axisB; }

}
```

Figure 12: myOval part2

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public class myCircle extends myOval{

    // constructor
    public myCircle(double x, double y, Color strokeColor, double axisA,double  axisB){
        super(x,y,strokeColor,axisA,axisB);
    }

    // perimeter method
    public double getPerimeter() { return 2*Math.PI*getRadius(); }
    // return radius method
    public double getRadius(){ return getAxisA(); }
    // toString method
    @Override
    public String toString(){
       return String.format("%s %s%n%s%.1f%n%s%.1f", "circle",
               super.toString(),"radius: ",getRadius(),"area: ", getArea());
    }
    // draw method
    @Override
    public void draw(GraphicsContext gc,double width, double height) {
        // set the color for the current arc
        gc.setFill(super.getColor());

        // Draw circle
        gc.strokeOval( x: super.getX() - width/2, y: super.getY()- height/2,
                w: 2*getRadius(), h: 2*getRadius());
        gc.fillOval( x: super.getX()- width/2, y: super.getY()- height/2,
                w: 2*getRadius(), h: 2*getRadius());
    }
    // getPoint method
    public Point getPoint(){
        Point point1 = new Point(getX(),getY());
        return point1;
    }

}
```

Figure 13: myCircle

```java
public class Point {
    public double xPoint;
    public double yPoint;

    Point(double xPoint, double yPoint){
        this.xPoint = xPoint;
        this.yPoint = yPoint;
    }

    //toString() method
    @Override
    public String toString(){
        return String.format("Point (x,y): (%.2f,%.2f)",xPoint,yPoint);
    }

}
```

Figure 14: Point

3.3 Polymorphism Test Codes

```
import javafx.fxml.FXML;
import javafx.scene.canvas.Canvas;
import javafx.event.ActionEvent;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class DrawShapesController {
    @FXML private Canvas canvas;
    @FXML
    void drawShapesButtonPressed(ActionEvent event) {

        // canvas dimensions and declaration
        GraphicsContext gc = canvas.getGraphicsContext2D();
        double centerX = canvas.getWidth()/2;
        double centerY = canvas.getHeight()/2;
        double canvasWidth = canvas.getWidth();
        double canvasHeight = canvas.getHeight();

        // create Color array
        Color colorList[] = {Color.BEIGE, Color.LIGHTGREEN,Color.BROWN,
                Color.LIGHTBLUE,Color.YELLOW,Color.VIOLET};

        // SHAPES DIMENSION 1
        double radiusX1 = canvasWidth/2.7;
        double radiusY1 = canvasHeight/3.0;
        myOval oval1 = new myOval( x: centerX+canvasWidth/2 - radiusX1,
                y: centerY + canvasHeight/2 - radiusY1 ,colorList[1],radiusX1,radiusY1);
        myRectangle rectangleForOval1 = oval1.getBoundingBox(canvasWidth,canvasHeight,colorList[0]);

        // SHAPES DIMENSION 2
        double radiusX2 = (rectangleForOval1.getWidth()/Math.sqrt(2))/2;
        double radiusY2 = (rectangleForOval1.getHeight()/Math.sqrt(2))/2;
        myOval oval2 = new myOval( x: centerX+canvasWidth/2 - radiusX2,
                y: centerY + canvasHeight/2 - radiusY2 ,colorList[3],radiusX2,radiusY2);
        myRectangle rectangleForOval2 = oval2.getBoundingBox(canvasWidth,canvasHeight,colorList[2]);

```

Figure 15: DrawShapesController for Polymorphism1

```
        // SHAPES DIMENSION 3
        double radiusX3 = (rectangleForOval2.getWidth()/Math.sqrt(2))/2;
        double radiusY3 = (rectangleForOval2.getHeight()/Math.sqrt(2))/2;
        myOval oval3 = new myOval( x: centerX+canvasWidth/2 - radiusX3,
                y: centerY + canvasHeight/2 - radiusY3 ,colorList[5],radiusX3,radiusY3);
        myRectangle rectangleForOval3 = oval3.getBoundingBox(canvasWidth,canvasHeight,colorList[4]);


//        rectangleForOval3.draw(gc,canvasWidth,canvasHeight);
//        oval3.draw(gc,canvasWidth,canvasHeight);

        // Initialize array with myShapes objects
        myShape[] myShapeList = new myShape[6];
        myShapeList[0] = rectangleForOval1;
        myShapeList[1] = oval1;
        myShapeList[2] = rectangleForOval2;
        myShapeList[3] = oval2;
        myShapeList[4] = rectangleForOval3;
        myShapeList[5] = oval3;

        System.out.println("================================================");
        System.out.printf("myShapes objects processed polymorphically:%n");
        System.out.print("================================================");
```

Figure 16: DrawShapesController for Polymorphism2

```
60
61          double rectangleNumber = 0;
62          double ovalNumber = 0;
63
64          for (myShape myshapeTest : myShapeList){
65
66              if (myshapeTest instanceof myRectangle) {
67                  rectangleNumber++;
68                  System.out.printf("%n%n%s %.0f", "Rectangle Shape: ", rectangleNumber);
69
70              }
71              if (myshapeTest instanceof myOval) {
72                  ovalNumber++;
73                  System.out.printf("%n%n%s %.0f", "Oval Shape " , ovalNumber);
74              }
75
76              System.out.printf("%n%s %n%s%.2f %n%s%.2f ",
77                      myshapeTest.getPoint(),"Area: ", myshapeTest.getArea(),
78                      "Circumference: ", myshapeTest.getPerimeter());
79
80              myshapeTest.draw(gc,canvasWidth,canvasHeight);
81          }
82      }
83  }
84
```

Figure 17: DrawShapesController for Polymorphism3

3.4 doOverlap Test Codes

```
1   import javafx.fxml.FXML;
2   import javafx.scene.canvas.Canvas;
3   import javafx.event.ActionEvent;
4   import javafx.scene.canvas.GraphicsContext;
5   import javafx.scene.paint.Color;
6
7   public class DrawShapesController {
8       @FXML private Canvas canvas;
9       @FXML
10      void drawShapesButtonPressed(ActionEvent event) {
11
12          // canvas dimensions and declaration
13          GraphicsContext gc = canvas.getGraphicsContext2D();
14          double centerX = canvas.getWidth() / 2;
15          double centerY = canvas.getHeight() / 2;
16          double canvasWidth = canvas.getWidth();
17          double canvasHeight = canvas.getHeight();
18
19          // create Color array
20          Color colorList[] = {Color.BEIGE, Color.LIGHTGREEN, Color.BROWN, Color.LIGHTBLUE,
21                  Color.YELLOW, Color.VIOLET};
22
23          //
24          System.out.println("Figure 1: Lightgreen oval ");
25          System.out.println("Figure 2: Lightblue oval ");
26          System.out.println("Figure 3: Brown circle ");
27          System.out.println("Figure 4: Yellow circle ");
28
29
```

Figure 18: DrawShapesController for doOverlap1

```
30      // EXAMPLE WHERE FIGURES OVERLAP
31      // SHAPES DIMENSION 1
32      double radiusX1 = canvasWidth / 5;
33      double radiusY1 = canvasHeight / 7.5;
34      myOval oval1 = new myOval( x: centerX + canvasWidth / 3 - canvasWidth / 6,
35              y: centerY + canvasHeight / 2 - canvasWidth / 6, colorList[1], radiusX1, radiusY1);
36
37      // SHAPES DIMENSION 2
38      myOval oval2 = new myOval( x: centerX + canvasWidth / 3 + canvasWidth / 12,
39              y: centerY + canvasHeight / 2 + canvasWidth / 12, colorList[3], radiusX1, radiusY1);
40
41      oval1.draw(gc, canvasWidth, canvasHeight);
42      oval2.draw(gc, canvasWidth, canvasHeight);
43
44
45      // EXAMPLE WHERE FIGURES DO NOT OVERLAP
46      // SHAPES DIMENSION 3
47      double radiusX2 = canvasWidth / 9;
48      double radiusY2 = canvasHeight / 9;
49      myOval oval3 = new myOval( x: centerX + canvasWidth / 2 - canvasWidth / 6,
50              y: centerY + canvasHeight / 2 - canvasWidth / 4, colorList[2], radiusX2, radiusX2);
51
52      // SHAPES DIMENSION 4
53      myOval oval4 = new myOval( x: centerX + canvasWidth / 2 + canvasWidth / 12,
54              y: centerY + canvasHeight / 2 + canvasWidth / 16, colorList[4], radiusY2, radiusY2);
55
56      oval3.draw(gc, canvasWidth, canvasHeight);
57      oval4.draw(gc, canvasWidth, canvasHeight);
58
```

Figure 19: DrawShapesController for doOverlap2

```
61      if (oval3.doOverlap(oval1, oval2)) {
62          System.out.println("figures 1 overlaps figure 2!");
63      } else {
64          System.out.println("figures 1 DO NOT!! overlap figure 2!");
65      }
66
67      if (oval3.doOverlap(oval1, oval3)) {
68          System.out.println("figures 1 overlaps figure 3!");
69      } else {
70          System.out.println("figures 1 DO NOT!! overlap figure 3!");
71      }
72      if (oval3.doOverlap(oval1, oval4)) {
73          System.out.println("figures 1 overlaps figure 4!");
74      } else {
75          System.out.println("figures 1 DO NOT!! overlap figure 4!");
76      }
77
78      if (oval3.doOverlap(oval2, oval3)) {
79          System.out.println("figures 2 overlaps figure 3!");
80      } else {
81          System.out.println("figures 2 DO NOT!! overlap figure 3!");
82      }
83      if (oval3.doOverlap(oval2, oval4)) {
84          System.out.println("figures 2 overlaps figure 4!");
85      } else {
86          System.out.println("figures 2 DO NOT!! overlap figure 4!");
87      }
```

Figure 20: DrawShapesController for doOverlap3

4 Outputs

The outputs of code from figures 15,16, 17 are shown in figures from 21 to 24.

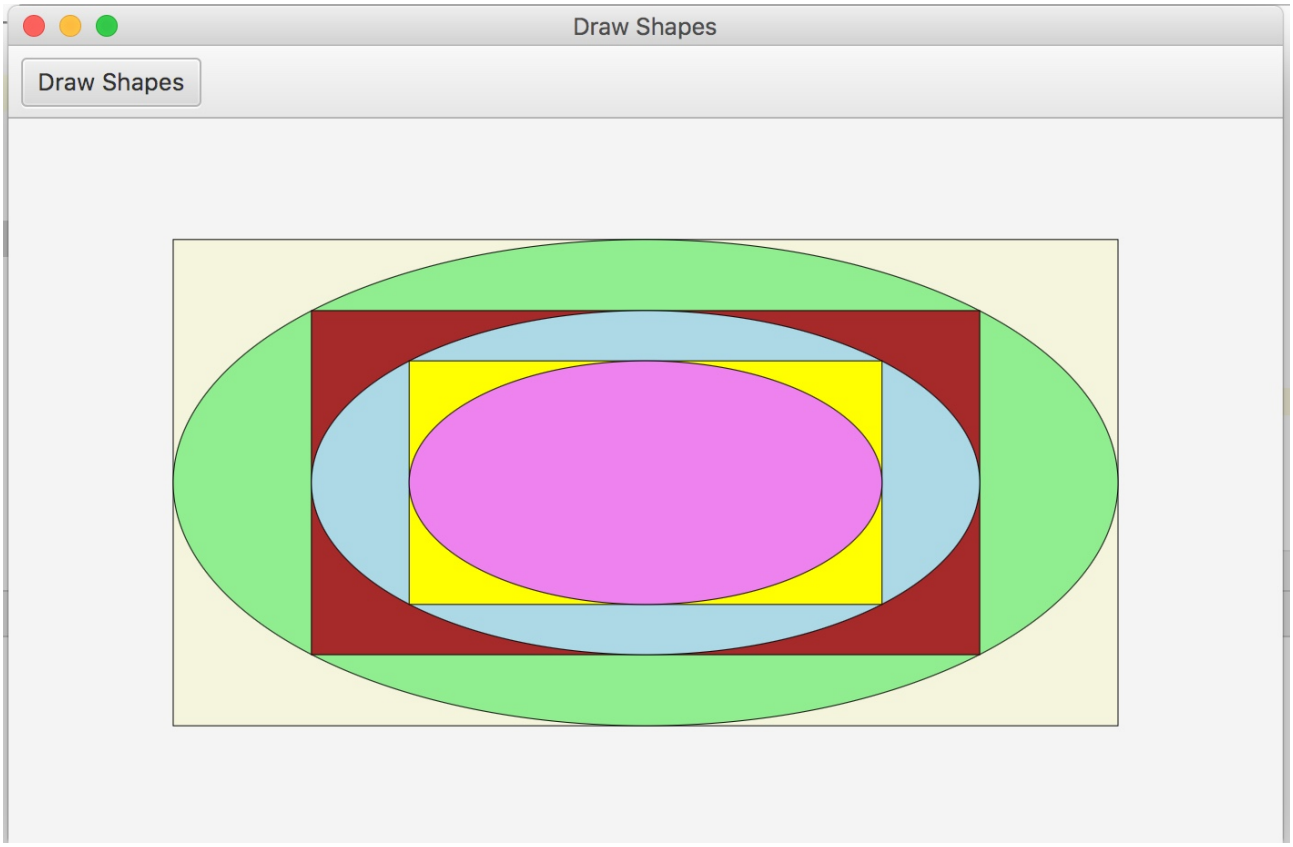

Figure 21: Output of DrawShapesController for Polymorphism

```
================================================
myShapes objects processed polymorphically:
================================================

Rectangle Shape:  1
Point (x,y): (350.00,200.00)
Area: 138271.60
Circumference: 1570.37

Oval Shape  1
Point (x,y): (350.00,200.00)
Area: 108598.26
Circumference: 1628.97

Rectangle Shape:  2
Point (x,y): (350.00,200.00)
Area: 69135.80
Circumference: 1110.42

Oval Shape  2
Point (x,y): (350.00,200.00)
Area: 54299.13
Circumference: 1151.86

Rectangle Shape:  3
Point (x,y): (350.00,200.00)
Area: 34567.90
Circumference: 785.19

Oval Shape  3
Point (x,y): (350.00,200.00)
Area: 27149.57
Circumference: 814.49
```

Figure 22: Output of DrawShapesController for Polymorphism

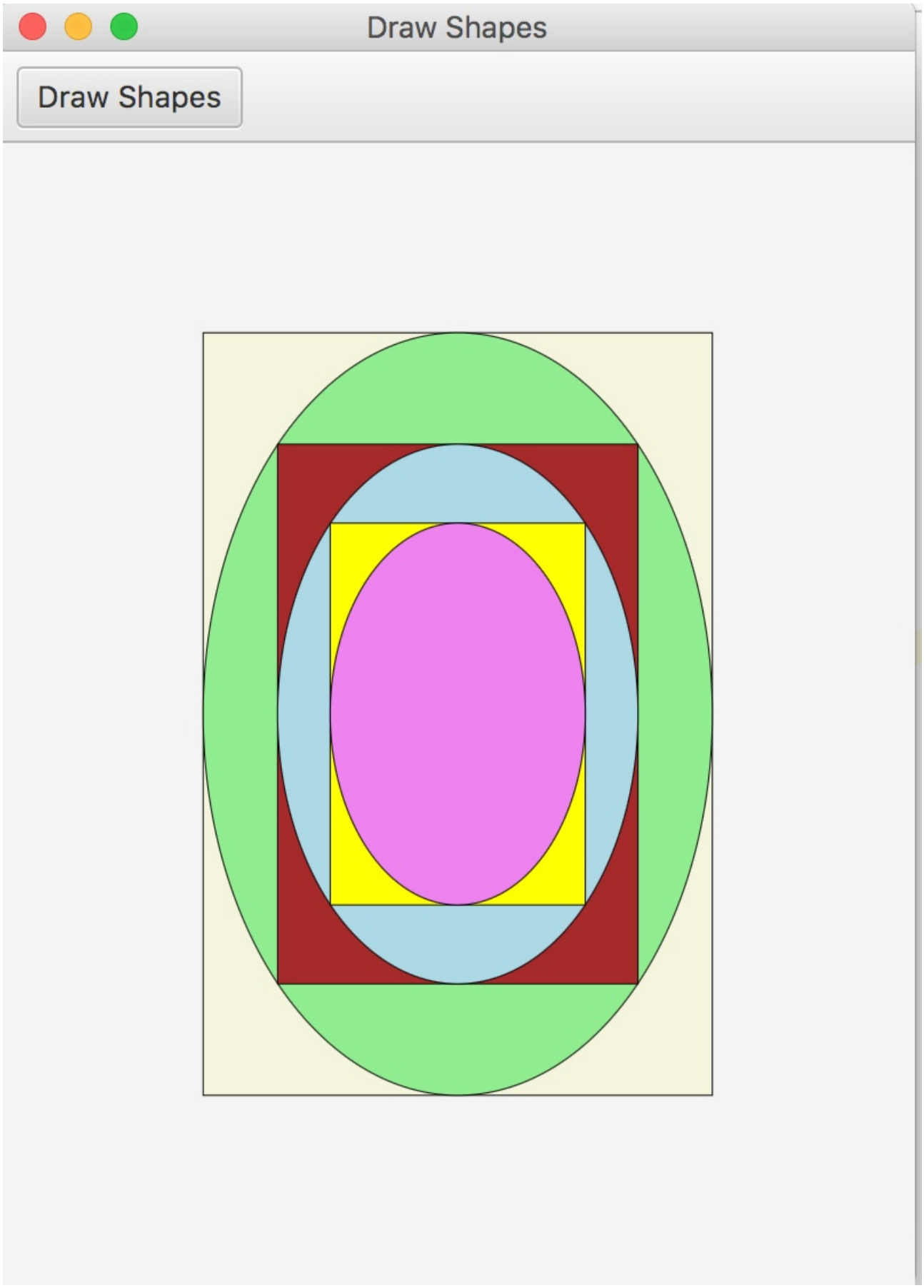

Figure 23: Output of DrawShapesController for Polymorphism

```
================================================
myShapes objects processed polymorphically:
================================================

Rectangle Shape:  1
Point (x,y): (150.00,250.00)
Area: 74074.07
Circumference: 1111.11

Oval Shape  1
Point (x,y): (150.00,250.00)
Area: 58177.64
Circumference: 698.13

Rectangle Shape:  2
Point (x,y): (150.00,250.00)
Area: 37037.04
Circumference: 785.67

Oval Shape  2
Point (x,y): (150.00,250.00)
Area: 29088.82
Circumference: 493.65

Rectangle Shape:  3
Point (x,y): (150.00,250.00)
Area: 18518.52
Circumference: 555.56

Oval Shape  3
Point (x,y): (150.00,250.00)
Area: 14544.41
Circumference: 349.07
```

Figure 24: Output of DrawShapesController for Polymorphism

4.1 doOverlap Test results

The results of codes 18, 19, 20 are shown in figures 25, and 26.

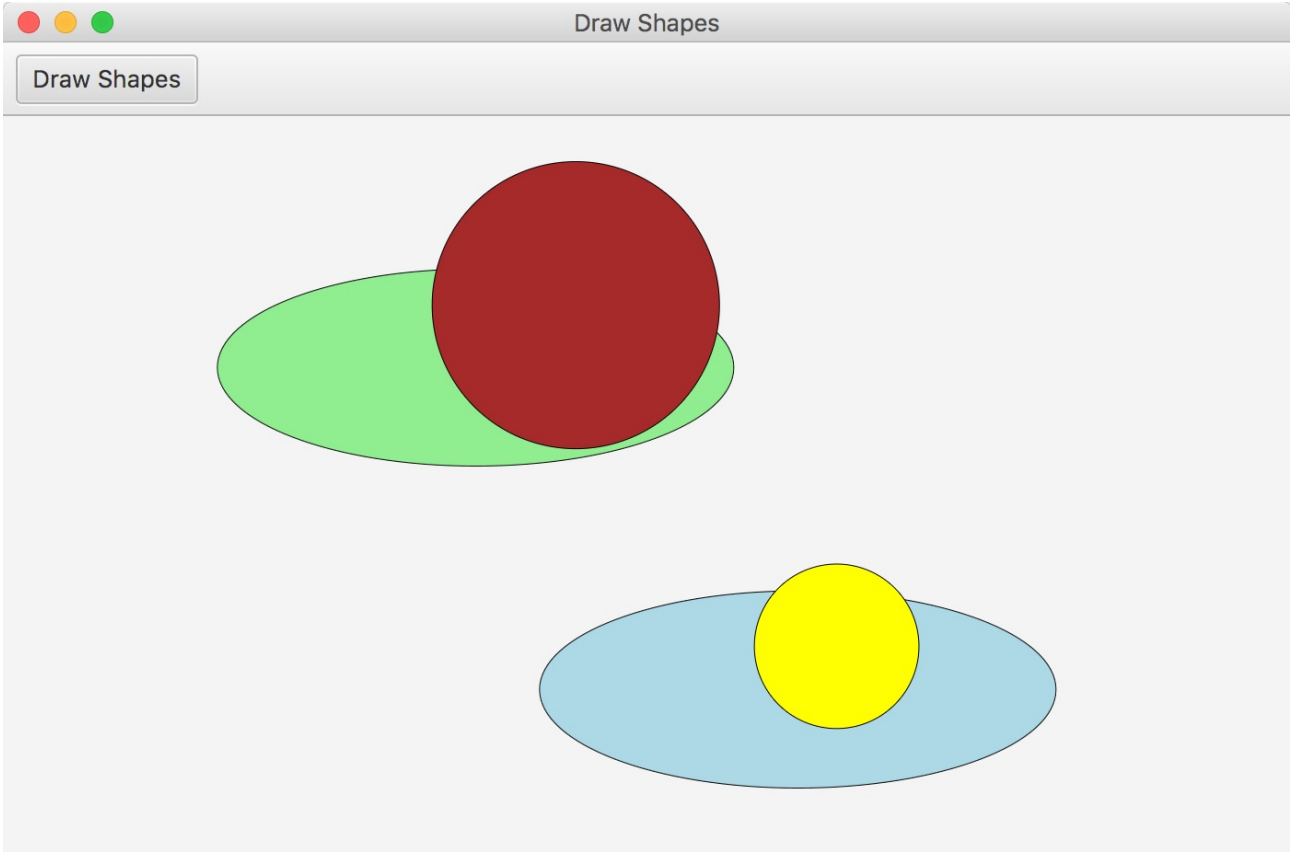

Figure 25: geometric configuration doOverlap

```
Figure 1: Lightgreen oval
Figure 2: Lightblue oval
Figure 3: Brown circle
Figure 4: Yellow circle
figures 1 DO NOT!! overlap figure 2!
figures 1 overlaps figure 3!
figures 1 DO NOT!! overlap figure 4!
figures 2 DO NOT!! overlap figure 3!
figures 2 overlaps figure 4!
figures 3 DO NOT!! overlap figure 4!
```

Figure 26: geometric configuration doOverlap written

5 References

[1] Deitel, Paul J., and Harvey M. Deitel. Java: How to Program Early Objects. Pearson, 2018.