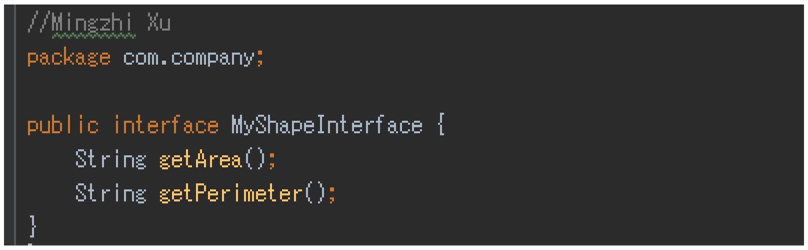
Mingzhi Xu

CSC 22100 Software Design Laboratory Fall 2018

Exercise 2

In this exercise, we are to create a hierarchy of Java classes MyShape, MyPolygon, and MyCircle from exercise 1, and we will be modifying these classes and adding MyRectangle and MyOval class in this exercise. We will also implement interfaces such that the MyShapeInterface interface will contain abstract method getArea which describes the area of an object in the class hierarchy and getPerimeter which describes the perimeter of an object in the class hierarchy.

Code for interface MyShapeInterface

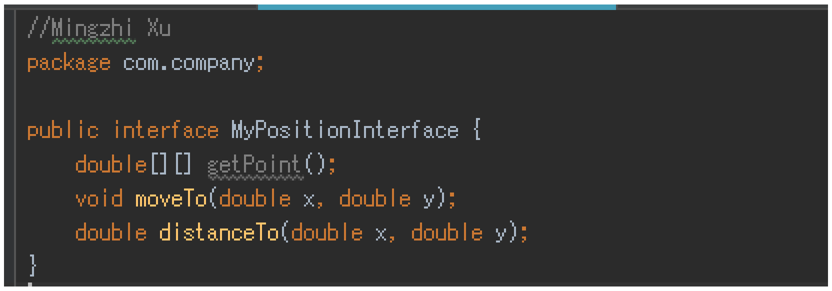


```java
//Mingzhi Xu
package com.company;

public interface MyShapeInterface {
    String getArea();
    String getPerimeter();
}
```

Since interface methods are public abstract there is no need to write it out since Java automatically initialize for you and since both methods describes an object, so the type would be String.

Another interface we will implement will be the MyPositionInterface which includes appropriate methods that describe the positional functions and behaviors of the specific object type of the class hierarchy. The interface includes public abstract methods getPoint which returns the point (x, y), moveTo which moves point (x, y) to the point (x + Δx, y + Δy), and distanceTo returns distance from point (x, y) to a point.

Code for interface MyPositionInterface

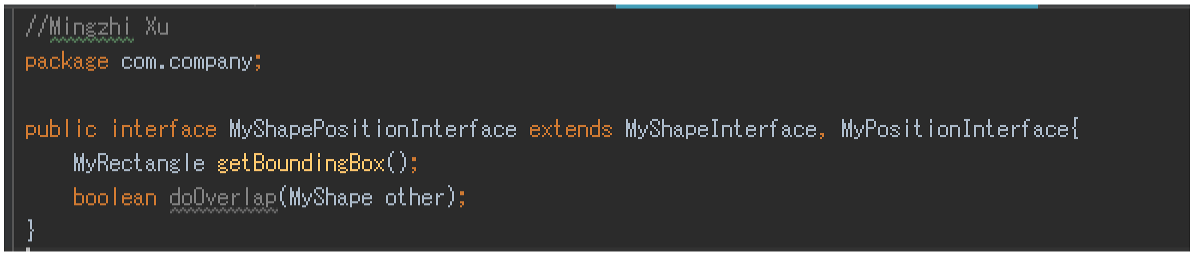


```
//Mingzhi Xu
package com.company;

public interface MyPositionInterface {
    double[][] getPoint();
    void moveTo(double x, double y);
    double distanceTo(double x, double y);
}
```

Since interface methods are public abstract there is no need to write it out since Java automatically initialize for you and getPoint returns (x, y) so I use a double array to store x points and y points, since moveTo moves the object and does not return anything it is type void and distanceTo returns a distance so I use double.

The third interface we will implement is MyShapePositionInterface which extends the interfaces MyShapeInterface and MyPositionInterface, and what this does is that the so the public abstract methods in the previous two interfaces can be declared and used later in the class hierarchy. MyShapePositionInterface will contain methods getBoundingBox which returns a bounding rectangle of an object in the class hierarchy, and doOverlap which returns true or false if two objects in the class hierarchy overlap.

Code for interface MyShapePositionInterface



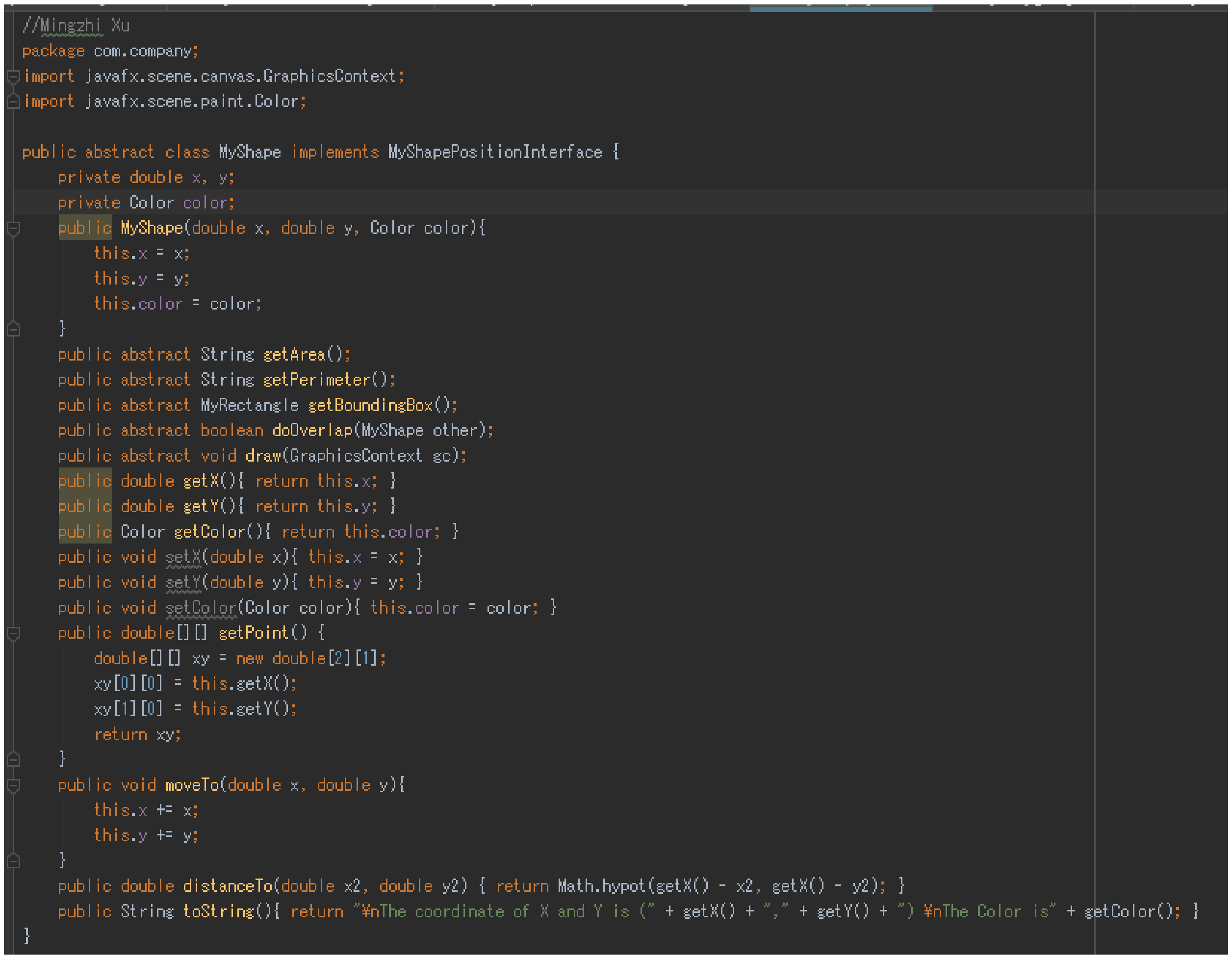
```
//Mingzhi Xu
package com.company;

public interface MyShapePositionInterface extends MyShapeInterface, MyPositionInterface{
    MyRectangle getBoundingBox();
    boolean doOverlap(MyShape other);
}
```

Since interface methods are public abstract there is no need to write it out since Java automatically initialize for you and getBoundingBox is suppose to return a bounding rectangle so

I declare it as a MyRectangle type which will be created later, and doOverlap will compare two MyShape objects to check if their bounding box overlaps and returns true or false so the type is Boolean.

Next, we will modify the MyShape class that was used in exercise 1 but MyShape class will be an abstract super class in this exercise so it contains abstract methods that will needed to be override later in the class hierarchy.

Code for MyShape abstract class

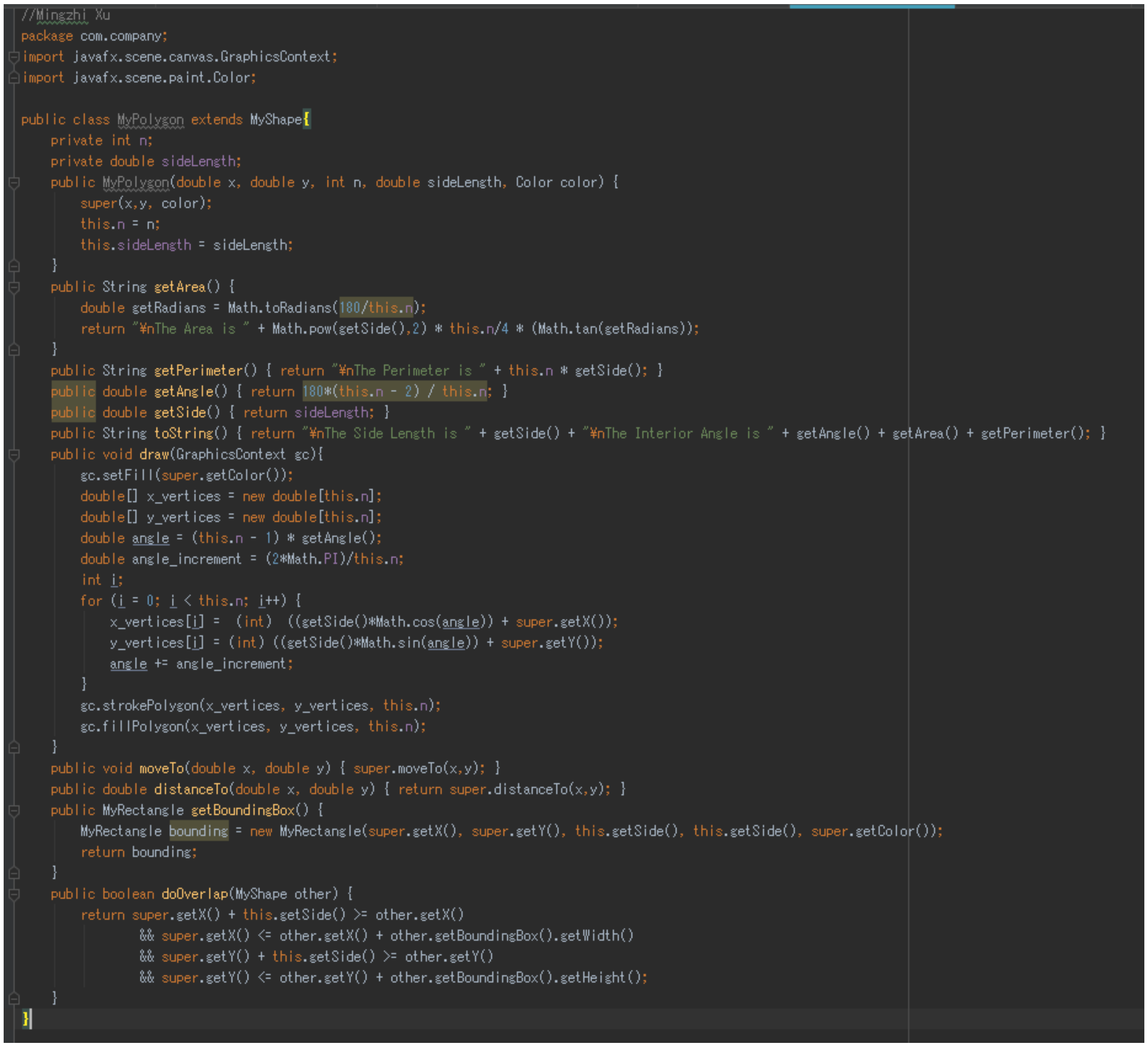


```java
//Mingzhi Xu
package com.company;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public abstract class MyShape implements MyShapePositionInterface {
    private double x, y;
    private Color color;
    public MyShape(double x, double y, Color color){
        this.x = x;
        this.y = y;
        this.color = color;
    }
    public abstract String getArea();
    public abstract String getPerimeter();
    public abstract MyRectangle getBoundingBox();
    public abstract boolean doOverlap(MyShape other);
    public abstract void draw(GraphicsContext gc);
    public double getX(){ return this.x; }
    public double getY(){ return this.y; }
    public Color getColor(){ return this.color; }
    public void setX(double x){ this.x = x; }
    public void setY(double y){ this.y = y; }
    public void setColor(Color color){ this.color = color; }
    public double[][] getPoint() {
        double[][] xy = new double[2][1];
        xy[0][0] = this.getX();
        xy[1][0] = this.getY();
        return xy;
    }
    public void moveTo(double x, double y){
        this.x += x;
        this.y += y;
    }
    public double distanceTo(double x2, double y2) { return Math.hypot(getX() - x2, getX() - y2); }
    public String toString(){ return "¥nThe coordinate of X and Y is (" + getX() + "," + getY() + ") ¥nThe Color is" + getColor(); }
}
```

Here we can see that most of the methods are like exercise 1 but since MyShape is an abstract class so it can’t instantiate objects, so we must declare methods such as draw, getBoundingBox, doOverlap, getArea, and getPerimeter as abstract methods which to be override later in the class hierarchy.

Then we will modify the class MyPolygon from exercise 1 and we will be overriding abstract methods from the interfaces.

Code for MyPolygon class

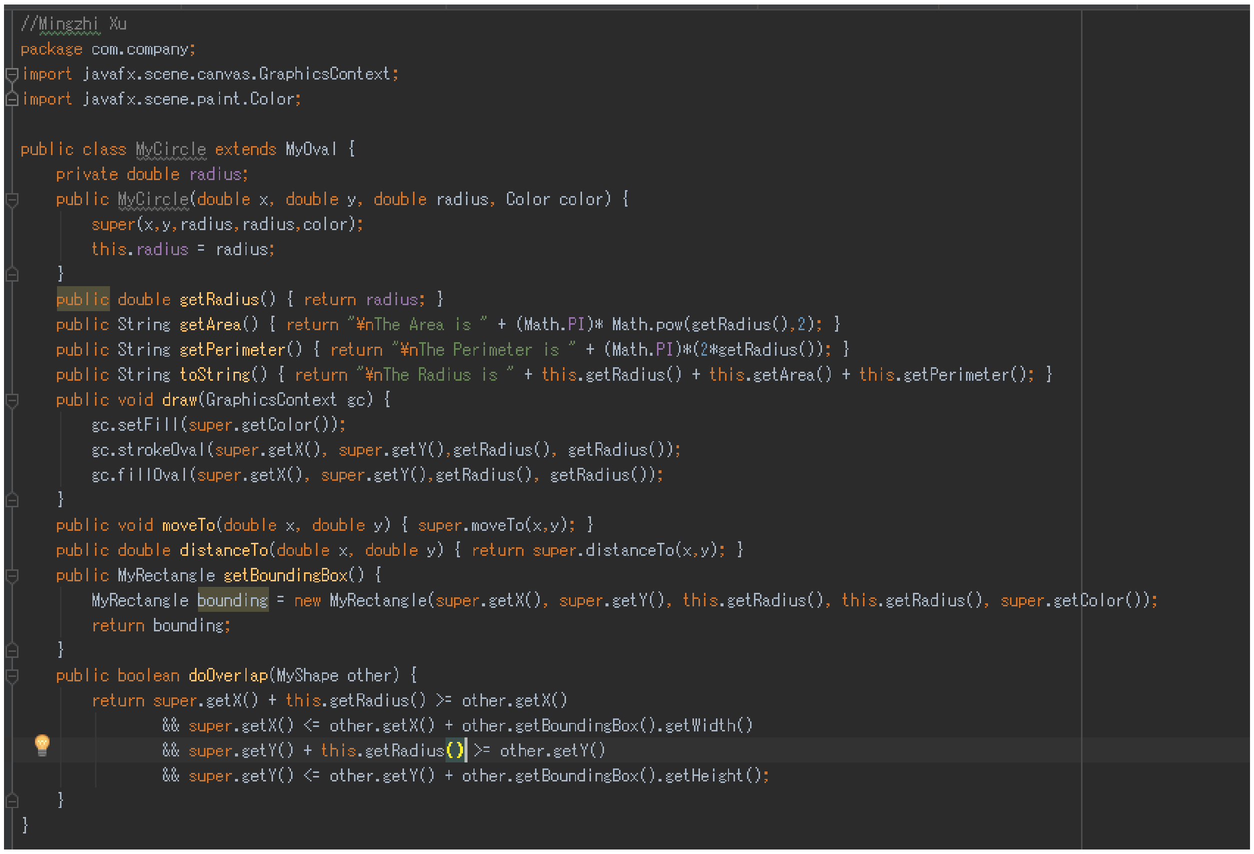


```java
//Mingzhi Xu
package com.company;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class MyPolygon extends MyShape{
    private int n;
    private double sideLength;
    public MyPolygon(double x, double y, int n, double sideLength, Color color) {
        super(x,y, color);
        this.n = n;
        this.sideLength = sideLength;
    }
    public String getArea() {
        double getRadians = Math.toRadians(180/this.n);
        return "¥nThe Area is " + Math.pow(getSide(),2) * this.n/4 * (Math.tan(getRadians));
    }
    public String getPerimeter() { return "¥nThe Perimeter is " + this.n * getSide(); }
    public double getAngle() { return 180*(this.n - 2) / this.n; }
    public double getSide() { return sideLength; }
    public String toString() { return "¥nThe Side Length is " + getSide() + "¥nThe Interior Angle is " + getAngle() + getArea() + getPerimeter(); }
    public void draw(GraphicsContext gc){
        gc.setFill(super.getColor());
        double[] x_vertices = new double[this.n];
        double[] y_vertices = new double[this.n];
        double angle = (this.n - 1) * getAngle();
        double angle_increment = (2*Math.PI)/this.n;
        int i;
        for (i = 0; i < this.n; i++) {
            x_vertices[i] =  (int)  ((getSide()*Math.cos(angle)) + super.getX());
            y_vertices[i] = (int) ((getSide()*Math.sin(angle)) + super.getY());
            angle += angle_increment;
        }
        gc.strokePolygon(x_vertices, y_vertices, this.n);
        gc.fillPolygon(x_vertices, y_vertices, this.n);
    }
    public void moveTo(double x, double y) { super.moveTo(x,y); }
    public double distanceTo(double x, double y) { return super.distanceTo(x,y); }
    public MyRectangle getBoundingBox() {
        MyRectangle bounding = new MyRectangle(super.getX(), super.getY(), this.getSide(), this.getSide(), super.getColor());
        return bounding;
    }
    public boolean doOverlap(MyShape other) {
        return super.getX() + this.getSide() >= other.getX()
                && super.getX() <= other.getX() + other.getBoundingBox().getWidth()
                && super.getY() + this.getSide() >= other.getY()
                && super.getY() <= other.getY() + other.getBoundingBox().getHeight();
    }
}
```

Here we can see that most of the methods are unchanged from exercise 1 but the new methods that are needed to override requires the MyRectangle class which is used to create the bounding box and to be used in the overlap method to demonstrate polymorphism.

Next, we will look at the class MyCircle which was used in exercise 1 but this time we will change up the class hierarchy a bit and instead of MyCircle extending MyShape, MyCircle will be extending MyOval which will be a new class that extends MyShape.

Code for MyCircle class

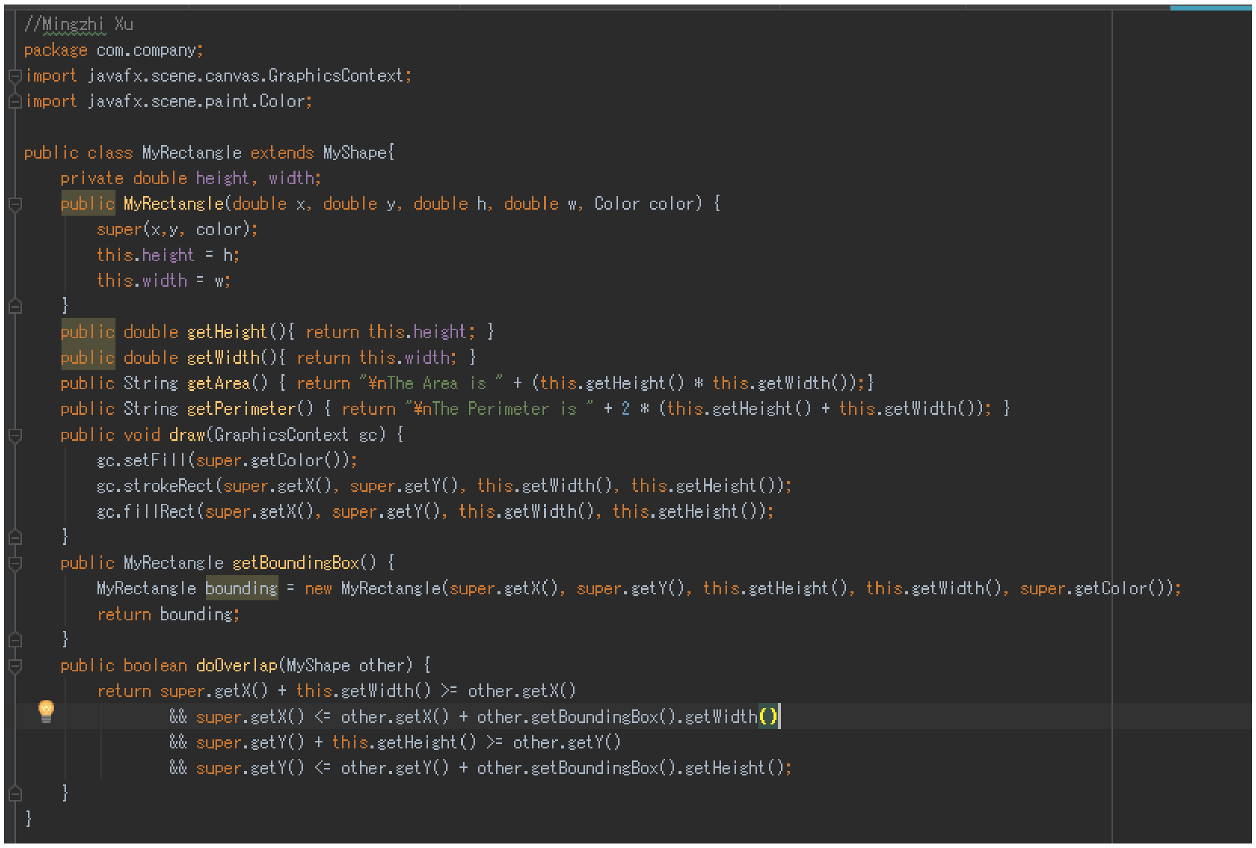


```java
//Mingzhi Xu
package com.company;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class MyCircle extends MyOval {
    private double radius;
    public MyCircle(double x, double y, double radius, Color color) {
        super(x,y,radius,radius,color);
        this.radius = radius;
    }
    public double getRadius() { return radius; }
    public String getArea() { return "¥nThe Area is " + (Math.PI)* Math.pow(getRadius(),2); }
    public String getPerimeter() { return "¥nThe Perimeter is " + (Math.PI)*(2*getRadius()); }
    public String toString() { return "¥nThe Radius is " + this.getRadius() + this.getArea() + this.getPerimeter(); }
    public void draw(GraphicsContext gc) {
        gc.setFill(super.getColor());
        gc.strokeOval(super.getX(), super.getY(),getRadius(), getRadius());
        gc.fillOval(super.getX(), super.getY(),getRadius(), getRadius());
    }
    public void moveTo(double x, double y) { super.moveTo(x,y); }
    public double distanceTo(double x, double y) { return super.distanceTo(x,y); }
    public MyRectangle getBoundingBox() {
        MyRectangle bounding = new MyRectangle(super.getX(), super.getY(), this.getRadius(), this.getRadius(), super.getColor());
        return bounding;
    }
    public boolean doOverlap(MyShape other) {
        return super.getX() + this.getRadius() >= other.getX()
                && super.getX() <= other.getX() + other.getBoundingBox().getWidth()
                && super.getY() + this.getRadius() >= other.getY()
                && super.getY() <= other.getY() + other.getBoundingBox().getHeight();
    }
}
```

Here we can see that most of the methods are unchanged from exercise one and similarly to MyPolygon, we will be overriding abstract methods for MyCircle.

Now we will be creating the new class MyRectangle that will be added to class hierarchy which extends MyShape. The class MyRectangle is like MyPolygon but instead of having equal length on each side, a MyRectangle object will have its own width and height and only have four sides.

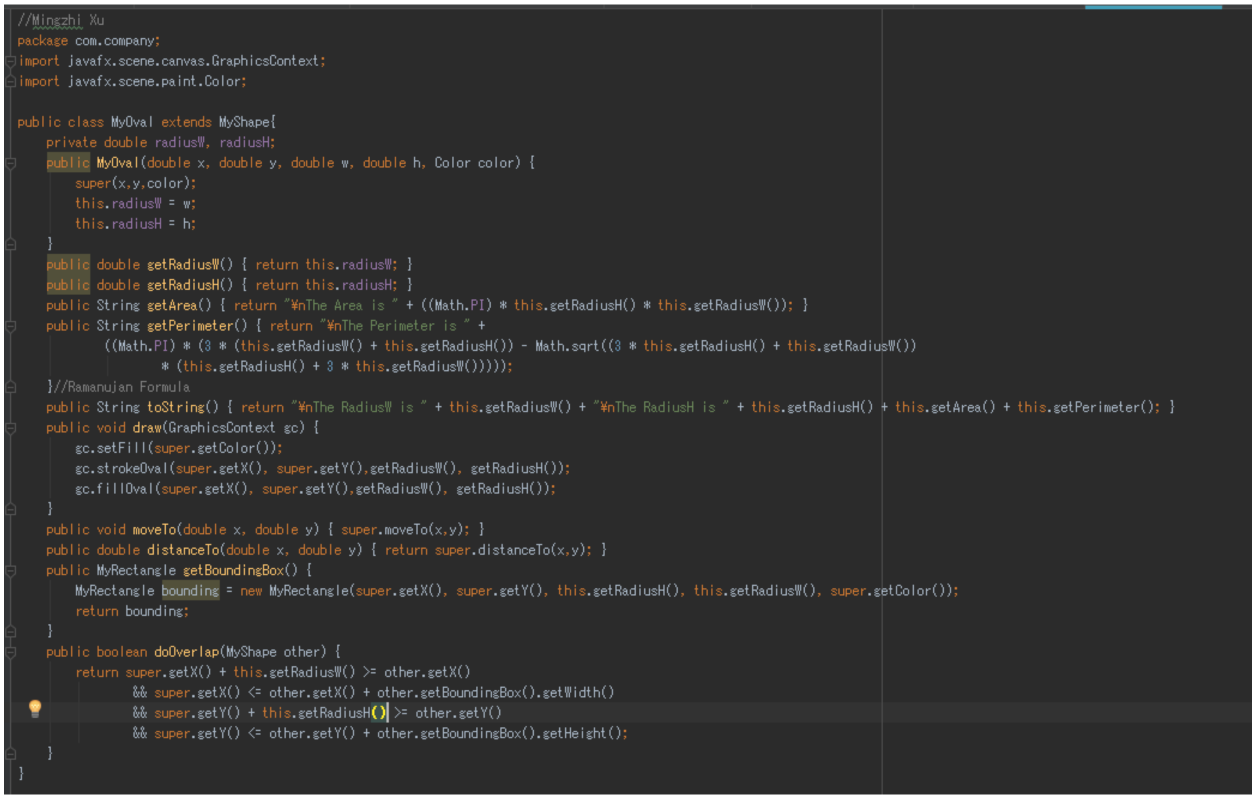
Code for MyRectangle class



```java
//Mingzhi Xu
package com.company;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class MyRectangle extends MyShape{
    private double height, width;
    public MyRectangle(double x, double y, double h, double w, Color color) {
        super(x,y, color);
        this.height = h;
        this.width = w;
    }
    public double getHeight(){ return this.height; }
    public double getWidth(){ return this.width; }
    public String getArea() { return "¥nThe Area is " + (this.getHeight() * this.getWidth());}
    public String getPerimeter() { return "¥nThe Perimeter is " + 2 * (this.getHeight() + this.getWidth()); }
    public void draw(GraphicsContext gc) {
        gc.setFill(super.getColor());
        gc.strokeRect(super.getX(), super.getY(), this.getWidth(), this.getHeight());
        gc.fillRect(super.getX(), super.getY(), this.getWidth(), this.getHeight());
    }
    public MyRectangle getBoundingBox() {
        MyRectangle bounding = new MyRectangle(super.getX(), super.getY(), this.getHeight(), this.getWidth(), super.getColor());
        return bounding;
    }
    public boolean doOverlap(MyShape other) {
        return super.getX() + this.getWidth() >= other.getX()
                && super.getX() <= other.getX() + other.getBoundingBox().getWidth()
                && super.getY() + this.getHeight() >= other.getY()
                && super.getY() <= other.getY() + other.getBoundingBox().getHeight();
    }
}
```

Lastly, we will be creating the new class MyOval that will be added to class hierarchy which extends MyShape. The class MyOval is like MyCircle but instead of having equal height and width radius, MyOval object have its own width radius and height radius.

Code for MyOval class

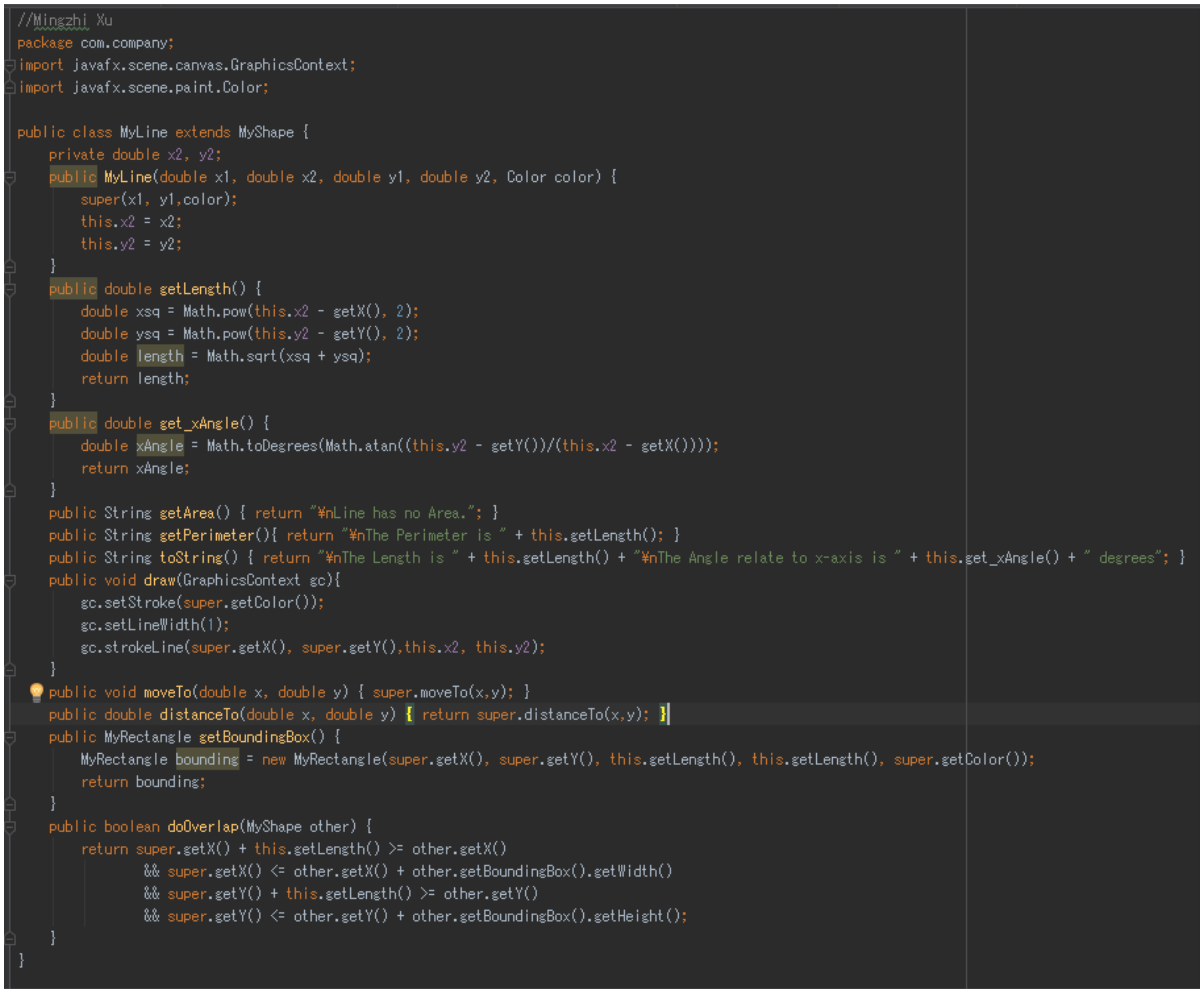


```java
//Mingzhi Xu
package com.company;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class MyOval extends MyShape{
    private double radiusW, radiusH;
    public MyOval(double x, double y, double w, double h, Color color) {
        super(x,y,color);
        this.radiusW = w;
        this.radiusH = h;
    }
    public double getRadiusW() { return this.radiusW; }
    public double getRadiusH() { return this.radiusH; }
    public String getArea() { return "¥nThe Area is " + ((Math.PI) * this.getRadiusH() * this.getRadiusW()); }
    public String getPerimeter() { return "¥nThe Perimeter is " +
            ((Math.PI) * (3 * (this.getRadiusW() + this.getRadiusH()) - Math.sqrt((3 * this.getRadiusH() + this.getRadiusW())
                    * (this.getRadiusH() + 3 * this.getRadiusW())))));
    }//Ramanujan Formula
    public String toString() { return "¥nThe RadiusW is " + this.getRadiusW() + "¥nThe RadiusH is " + this.getRadiusH() + this.getArea() + this.getPerimeter(); }
    public void draw(GraphicsContext gc) {
        gc.setFill(super.getColor());
        gc.strokeOval(super.getX(), super.getY(),getRadiusW(), getRadiusH());
        gc.fillOval(super.getX(), super.getY(),getRadiusW(), getRadiusH());
    }
    public void moveTo(double x, double y) { super.moveTo(x,y); }
    public double distanceTo(double x, double y) { return super.distanceTo(x,y); }
    public MyRectangle getBoundingBox() {
        MyRectangle bounding = new MyRectangle(super.getX(), super.getY(), this.getRadiusH(), this.getRadiusW(), super.getColor());
        return bounding;
    }
    public boolean doOverlap(MyShape other) {
        return super.getX() + this.getRadiusW() >= other.getX()
                && super.getX() <= other.getX() + other.getBoundingBox().getWidth()
                && super.getY() + this.getRadiusH() >= other.getY()
                && super.getY() <= other.getY() + other.getBoundingBox().getHeight();
    }
}
```

Here we can see that most of the method are like MyCircle class but since MyOval has a different width radius than height radius, we can use the Ramanujan formula to determine the perimeter.

We will also have the class MyLine but nothing much have changed from exercise 1.

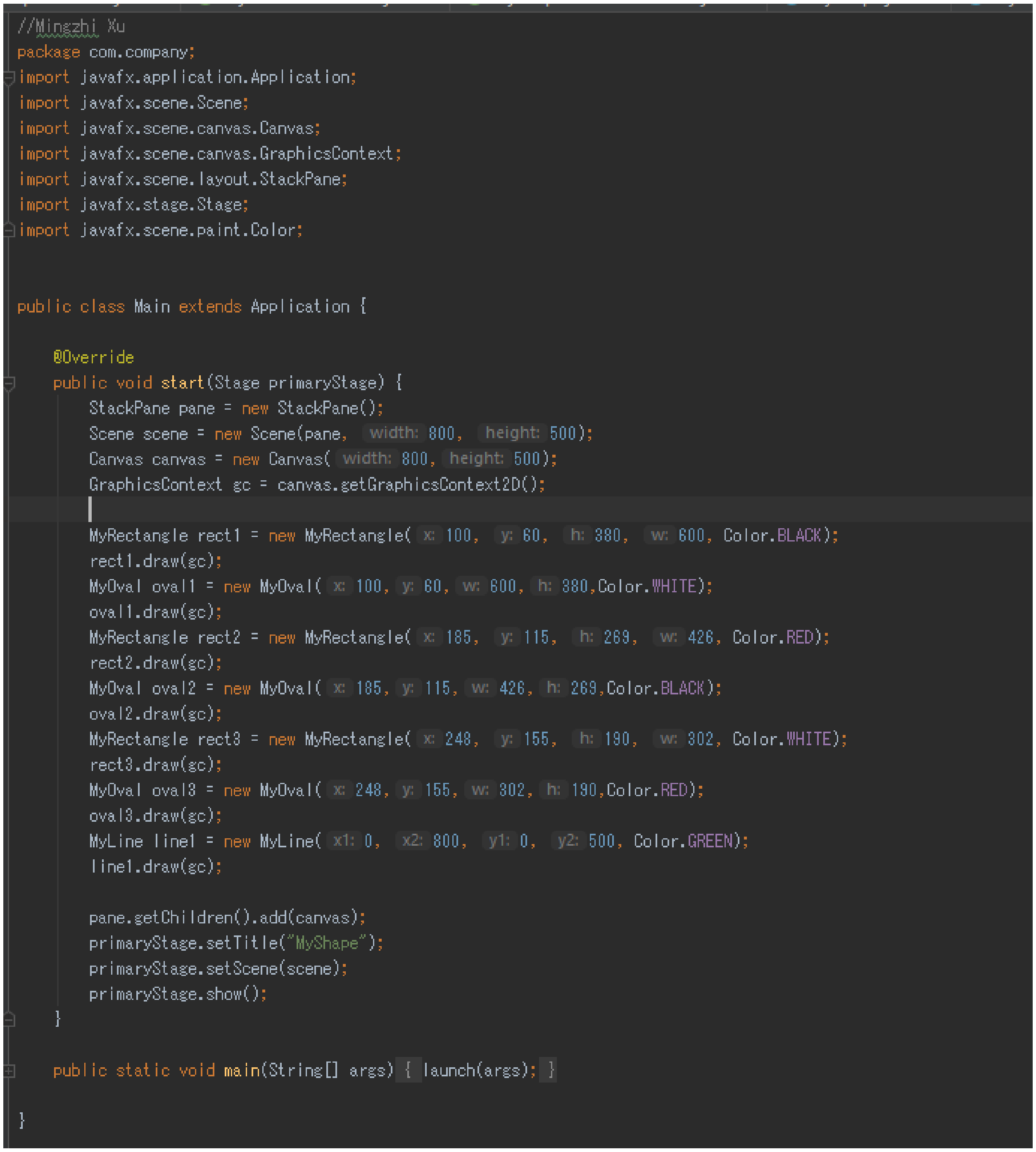
Code for MyLine class



```java
//Mingzhi Xu
package com.company;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class MyLine extends MyShape {
    private double x2, y2;
    public MyLine(double x1, double x2, double y1, double y2, Color color) {
        super(x1, y1,color);
        this.x2 = x2;
        this.y2 = y2;
    }
    public double getLength() {
        double xsq = Math.pow(this.x2 - getX(), 2);
        double ysq = Math.pow(this.y2 - getY(), 2);
        double length = Math.sqrt(xsq + ysq);
        return length;
    }
    public double get_xAngle() {
        double xAngle = Math.toDegrees(Math.atan((this.y2 - getY())/(this.x2 - getX())));
        return xAngle;
    }
    public String getArea() { return "¥nLine has no Area."; }
    public String getPerimeter(){ return "¥nThe Perimeter is " + this.getLength(); }
    public String toString() { return "¥nThe Length is " + this.getLength() + "¥nThe Angle relate to x-axis is " + this.get_xAngle() + " degrees"; }
    public void draw(GraphicsContext gc){
        gc.setStroke(super.getColor());
        gc.setLineWidth(1);
        gc.strokeLine(super.getX(), super.getY(),this.x2, this.y2);
    }
    public void moveTo(double x, double y) { super.moveTo(x,y); }
    public double distanceTo(double x, double y) { return super.distanceTo(x,y); }
    public MyRectangle getBoundingBox() {
        MyRectangle bounding = new MyRectangle(super.getX(), super.getY(), this.getLength(), this.getLength(), super.getColor());
        return bounding;
    }
    public boolean doOverlap(MyShape other) {
        return super.getX() + this.getLength() >= other.getX()
                && super.getX() <= other.getX() + other.getBoundingBox().getWidth()
                && super.getY() + this.getLength() >= other.getY()
                && super.getY() <= other.getY() + other.getBoundingBox().getHeight();
    }
}
```

Finally, we will draw rectangles and ovals in the main class. We can determine the width and height of the inner rectangle using a ratio of half of the width of the outer rectangle over square root 2 and half of the height of the out outer rectangle over square root 2. There won’t be much change on the main class since we are just drawing different shapes compared to exercise 1.

Code for main class



```
//Mingzhi Xu
package com.company;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
import javafx.scene.paint.Color;


public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Scene scene = new Scene(pane,  width: 800,  height: 500);
        Canvas canvas = new Canvas( width: 800,  height: 500);
        GraphicsContext gc = canvas.getGraphicsContext2D();

        MyRectangle rect1 = new MyRectangle( x: 100,  y: 60,  h: 380,  w: 600, Color.BLACK);
        rect1.draw(gc);
        MyOval oval1 = new MyOval( x: 100,  y: 60,  w: 600,  h: 380,Color.WHITE);
        oval1.draw(gc);
        MyRectangle rect2 = new MyRectangle( x: 185,  y: 115,  h: 269,  w: 426, Color.RED);
        rect2.draw(gc);
        MyOval oval2 = new MyOval( x: 185,  y: 115,  w: 426,  h: 269,Color.BLACK);
        oval2.draw(gc);
        MyRectangle rect3 = new MyRectangle( x: 248,  y: 155,  h: 190,  w: 302, Color.WHITE);
        rect3.draw(gc);
        MyOval oval3 = new MyOval( x: 248,  y: 155,  w: 302,  h: 190,Color.RED);
        oval3.draw(gc);
        MyLine line1 = new MyLine( x1: 0,  x2: 800,  y1: 0,  y2: 500, Color.GREEN);
        line1.draw(gc);

        pane.getChildren().add(canvas);
        primaryStage.setTitle("MyShape");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Here is the result of drawing those rectangles and ovals.