Hope Dunner

Hesham Auda

CSC 221

Project 2

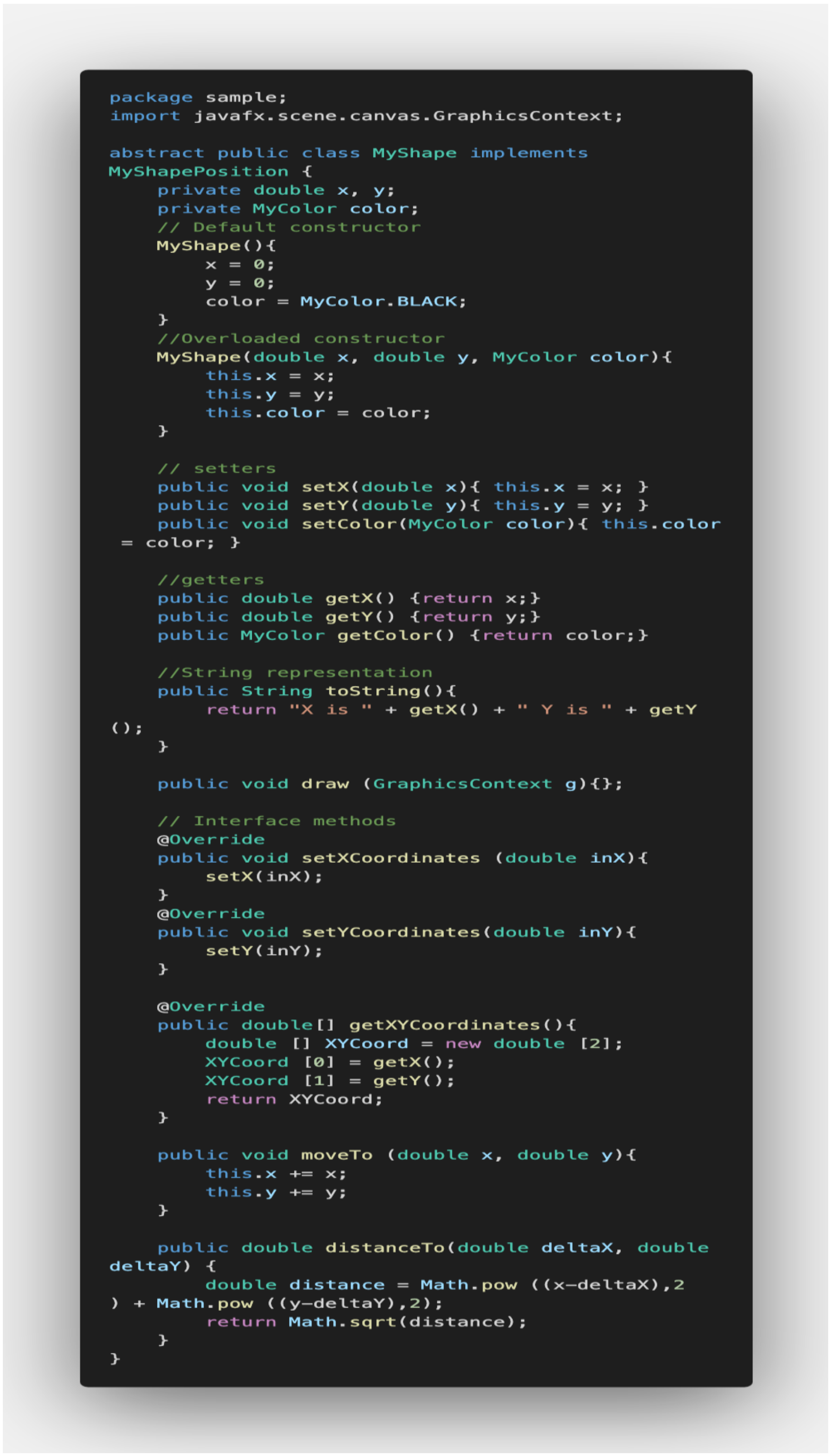
For the second project our class was instructed to implement JavaFx graphics and use a class hierarchy to draw several geometric sequences. The geometric sequences comprised of several ovals transcribed inside of several rectangles. And over the entire image was then transcribed diagonally with the line, requiring the dimensions of the line proportional to the entire geometric sequence. Finally the project required us to create a MyColor class and fill our geometric ovals and rectangles with the different colors of our choice.

First we had to create different classes that would inherit the Java class outputs. The first class that we had to implement was a MyShape class. Class MyShape is an abstract class; and serves as the hierarchy's superclass; and inherits Java class Object. The draw method in class MyShape is an abstract method and hence must be overridden in each subclass in the hierarchy. The class had several constructors and methods within the class; x, y, color. These constructors return the point(x, y) as well as the color for the MyShape object. I also implemented a ToString function that returns the objects description to the coordinates of the string. Also I created DistanceTo and moveTo which would later be implemented in the MyPoint interface. Lastly a draw function that draws the MyShape object was written. This method was overridden in the subclass hierarchy. Within this specific abstract class it painted the drawing canvas. Similar to the first project, I had to only focus on overriding each abstract method in each subclass hierarchy.

Hope Dunner

Hesham Auda

CSC 221



```java
package sample;
import javafx.scene.canvas.GraphicsContext;

abstract public class MyShape implements
MyShapePosition {
    private double x, y;
    private MyColor color;
    // Default constructor
    MyShape(){
        x = 0;
        y = 0;
        color = MyColor.BLACK;
    }
    //Overloaded constructor
    MyShape(double x, double y, MyColor color){
        this.x = x;
        this.y = y;
        this.color = color;
    }

    // setters
    public void setX(double x){ this.x = x; }
    public void setY(double y){ this.y = y; }
    public void setColor(MyColor color){ this.color
 = color; }

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

    //String representation
    public String toString(){
        return "X is " + getX() + " Y is " + getY
();
    }

    public void draw (GraphicsContext g){};

    // Interface methods
    @Override
    public void setXCoordinates (double inX){
        setX(inX);
    }
    @Override
    public void setYCoordinates(double inY){
        setY(inY);
    }

    @Override
    public double[] getXYCoordinates(){
        double [] XYCoord = new double [2];
        XYCoord [0] = getX();
        XYCoord [1] = getY();
        return XYCoord;
    }

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

    public double distanceTo(double deltaX, double
deltaY) {
        double distance = Math.pow ((x-deltaX),2
) + Math.pow ((y-deltaY),2);
        return Math.sqrt(distance);
    }
}
```

I imported a canvas package for this class. From this abstract class two different, class shapes where inherited. MyRectangle and MyOval are the next two classes that I implemented from the

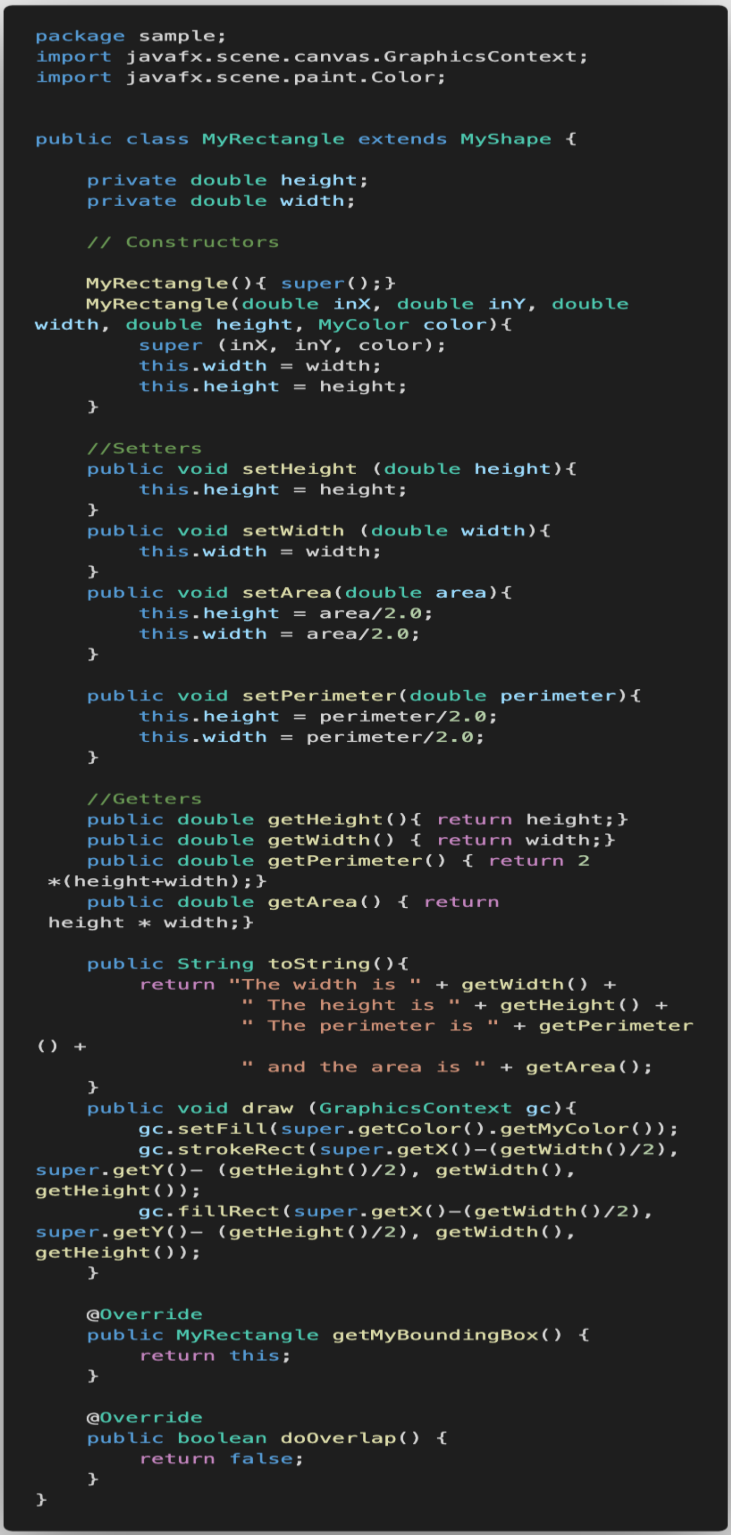
Hope Dunner

Hesham Auda

CSC 221

hierarchal MyShape class. I only used the canvas package for the sole purpose of displaying the shapes.

For the next class I implemented a MyRectangle class which was inherited from the MyShape class. Because the MyRectangle object is a rectangle of height h and width w, centered at a point (x, y), and filled with color, I had to make certain constructors and methods to allow me to produce these requirements. I made getter constructors which were used to return the width, height, perimeter, and area of the My Rectangle object. I also implemented setter constructors which set the width, height, perimeter, and area of the MyRectangle object. I used a ToString constructor to return a



```
package sample;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;


public class MyRectangle extends MyShape {

    private double height;
    private double width;

    // Constructors

    MyRectangle(){ super();}
    MyRectangle(double inX, double inY, double
width, double height, MyColor color){
        super (inX, inY, color);
        this.width = width;
        this.height = height;
    }

    //Setters
    public void setHeight (double height){
        this.height = height;
    }
    public void setWidth (double width){
        this.width = width;
    }
    public void setArea(double area){
        this.height = area/2.0;
        this.width = area/2.0;
    }

    public void setPerimeter(double perimeter){
        this.height = perimeter/2.0;
        this.width = perimeter/2.0;
    }

    //Getters
    public double getHeight(){ return height;}
    public double getWidth() { return width;}
    public double getPerimeter() { return 2
 *(height+width);}
    public double getArea() { return
 height * width;}

    public String toString(){
        return "The width is " + getWidth() +
                " The height is " + getHeight() +
                " The perimeter is " + getPerimeter
() +
                " and the area is " + getArea();
    }
    public void draw (GraphicsContext gc){
        gc.setFill(super.getColor().getMyColor());
        gc.strokeRect(super.getX()-(getWidth()/2),
super.getY()- (getHeight()/2), getWidth(),
getHeight());
        gc.fillRect(super.getX()-(getWidth()/2),
super.getY()- (getHeight()/2), getWidth(),
getHeight());
    }

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

    @Override
    public boolean doOverlap() {
        return false;
    }
}
```

Hope Dunner

Hesham Auda

CSC 221

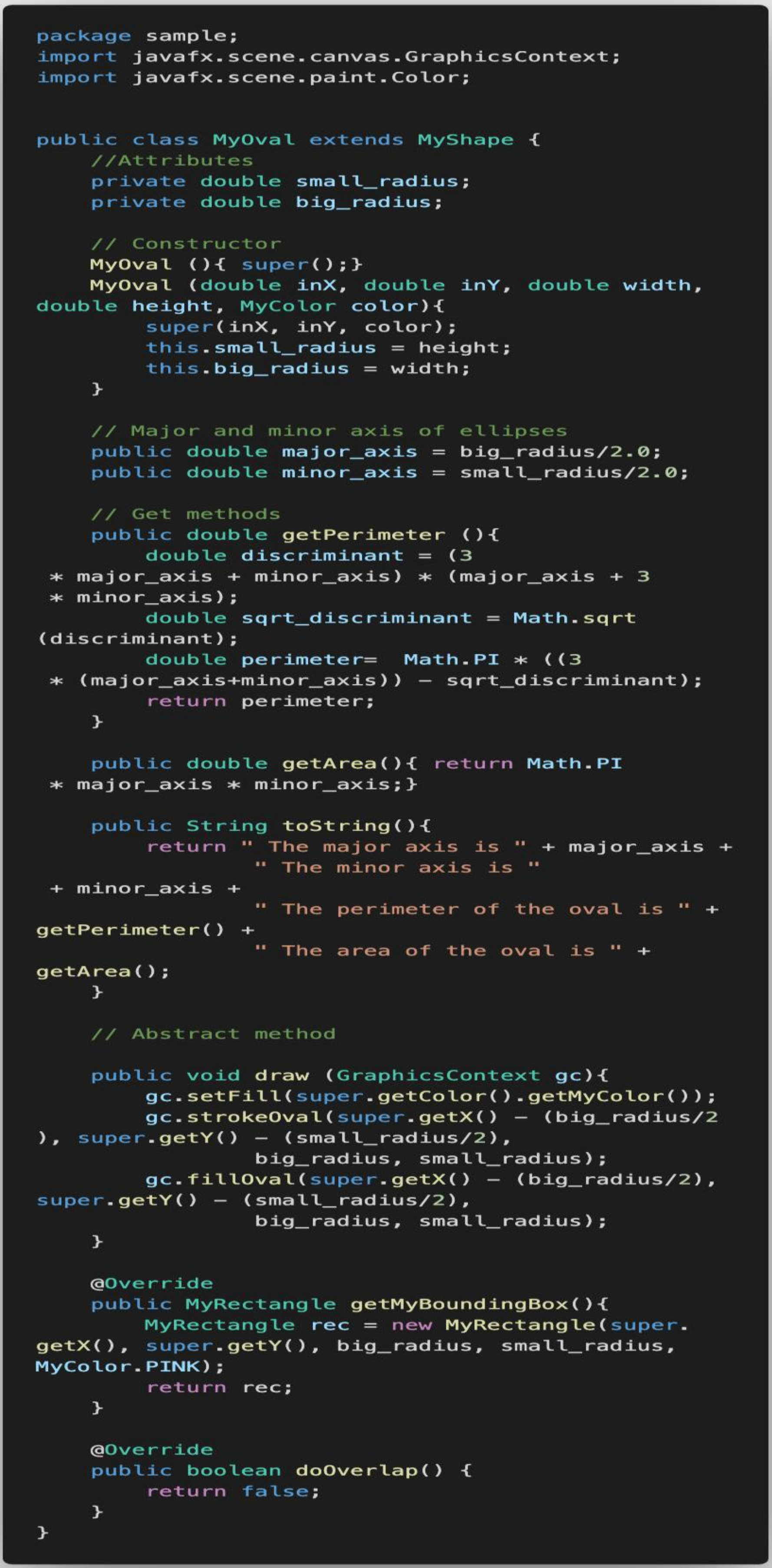
string representation of the MyRectangle object returning the fields; width, height, perimeter, and area. The ToString used the getter constructors implemented above concatenated with a string. Finally a draw method was implemented which drew the MyRectangle which with the help of the canvas package I implemented, I was able to draw the rectangle class with a specified height and width center at the point (x, y). The center of the rectangle is again defined in the class MyShape. The paint package was to see color on my shapes.

Next I implemented which was inherited from the MyOval class. Because the MyOval object is a rectangle of height h and width w, centered at a point (x, y), and filled with color, I had to make certain constructors and methods to allow me to produce these requirements. I made getter constructors which were used to return the perimeter. GetPerimeter got the discriminant by multiplying ((3* width/2)+ height/2) * ((width/2 +3)* height/2), taking the square root of that product, and using that product to subtract from the major and minor ellipses multiplied by 3 and finally multiplied by pi. I made the getArea by multiplying Pi to the major and minor axis of ellipses. I used a ToString constructor to return a string representation of the MyOval object returning the fields; axes lengths, perimeter, and area. The ToString used the getter constructors implemented above concatenated with a string. Finally a draw method was implemented which drew the MyOval which with the help of the canvas package I implemented, I was able to draw the oval class with a specified height and width center at the point (x, y). The center of the rectangle is again defined in the class MyShape. I used the canvas and paint packages so I could see my images when I ran my code.

Hope Dunner

Hesham Auda

CSC 221



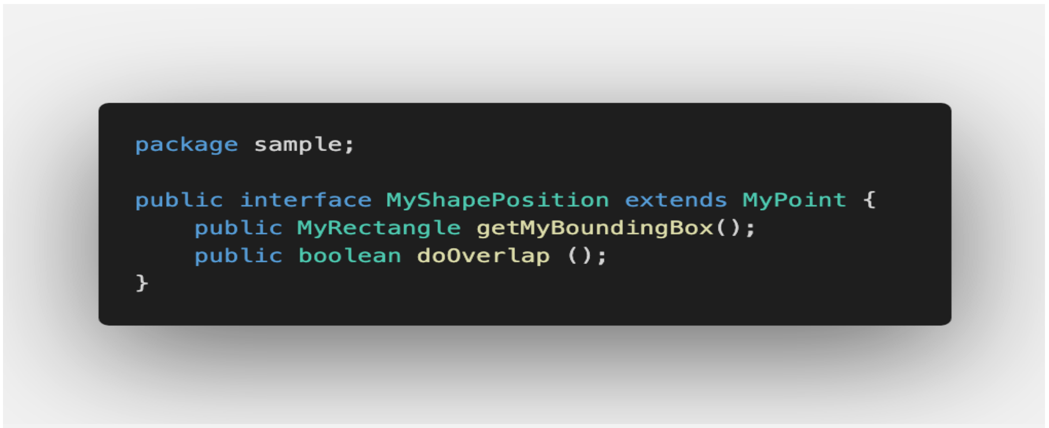
```
package sample;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;


public class MyOval extends MyShape {
    //Attributes
    private double small_radius;
    private double big_radius;

    // Constructor
    MyOval (){ super();}
    MyOval (double inX, double inY, double width,
double height, MyColor color){
        super(inX, inY, color);
        this.small_radius = height;
        this.big_radius = width;
    }

    // Major and minor axis of ellipses
    public double major_axis = big_radius/2.0;
    public double minor_axis = small_radius/2.0;

    // Get methods
    public double getPerimeter (){
        double discriminant = (3
 * major_axis + minor_axis) * (major_axis + 3
 * minor_axis);
        double sqrt_discriminant = Math.sqrt
(discriminant);
        double perimeter=  Math.PI * ((3
 * (major_axis+minor_axis)) - sqrt_discriminant);
        return perimeter;
    }

    public double getArea(){ return Math.PI
 * major_axis * minor_axis;}

    public String toString(){
        return " The major axis is " + major_axis +
                " The minor axis is "
 + minor_axis +
                " The perimeter of the oval is " +
getPerimeter() +
                " The area of the oval is " +
getArea();
    }

    // Abstract method

    public void draw (GraphicsContext gc){
        gc.setFill(super.getColor().getMyColor());
        gc.strokeOval(super.getX() - (big_radius/2
), super.getY() - (small_radius/2),
                big_radius, small_radius);
        gc.fillOval(super.getX() - (big_radius/2),
super.getY() - (small_radius/2),
                big_radius, small_radius);
    }

    @Override
    public MyRectangle getMyBoundingBox(){
        MyRectangle rec = new MyRectangle(super.
getX(), super.getY(), big_radius, small_radius,
MyColor.PINK);
        return rec;
    }

    @Override
    public boolean doOverlap() {
        return false;
    }
}
```

Hope Dunner

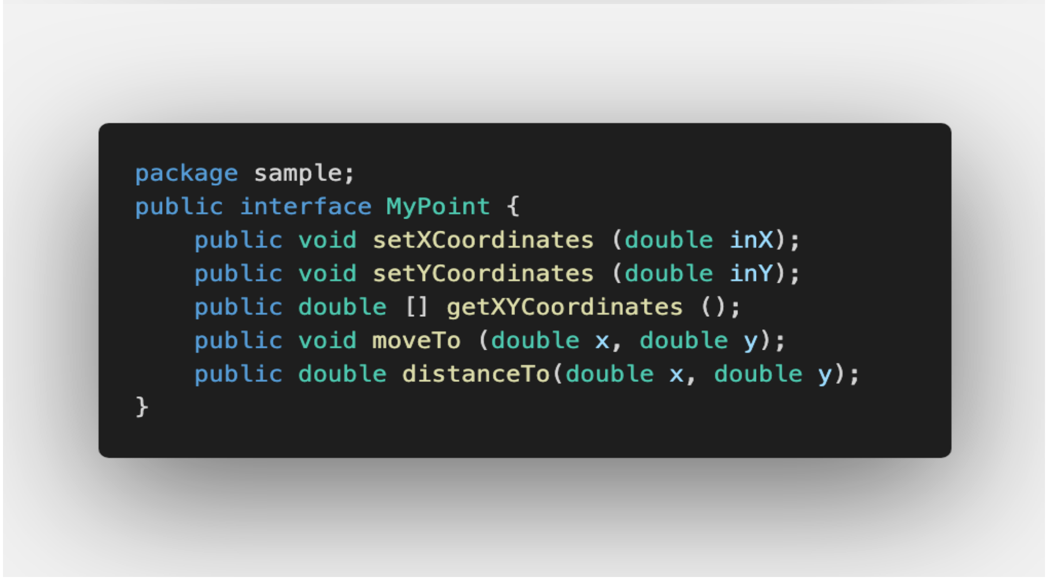
Hesham Auda

CSC 221

The next two factors that needed to be implemented were the interface files. Here the MyPoint and the MyShapePosition are connected to the class hierarchy of MyShape. The abstract class MyShape implements the interface MyShapePosition which extends the interface MyPoint. Within the MyPoint interface, getPoint, setPoint, moveTo, and DistanceTo are all called within the MyShape abstract class. The interface MyShapePosition calls on the functions of getMyBoundingBox and doOverlap



```java
package sample;

public interface MyShapePosition extends MyPoint {
    public MyRectangle getMyBoundingBox();
    public boolean doOverlap ();
}
```



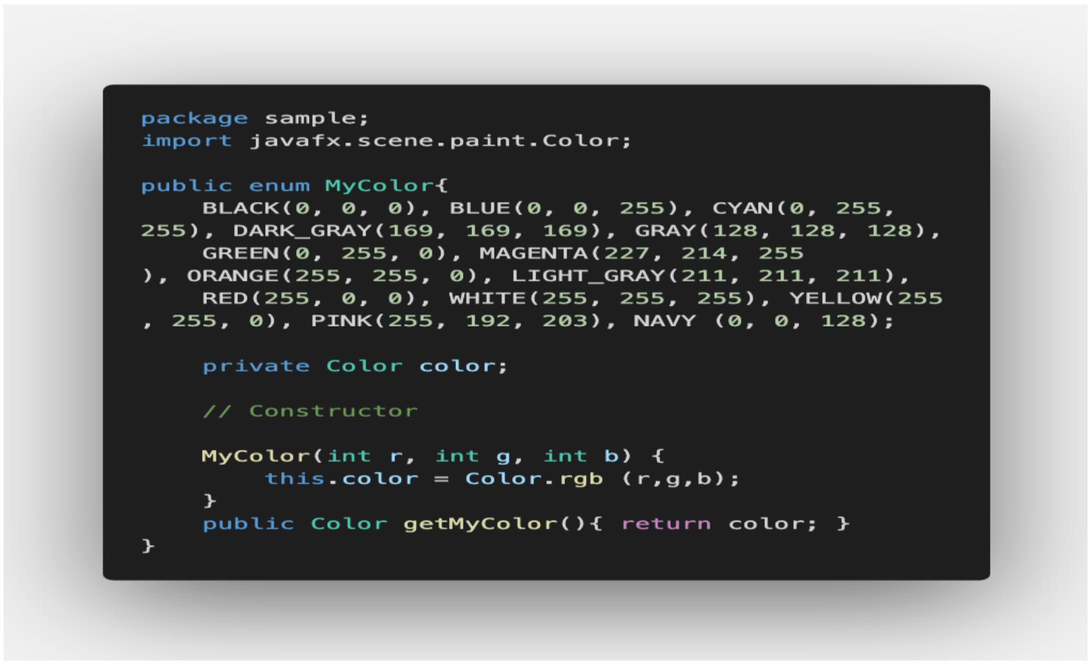
```java
package sample;
public interface MyPoint {
    public void setXCoordinates (double inX);
    public void setYCoordinates (double inY);
    public double [] getXYCoordinates ();
    public void moveTo (double x, double y);
    public double distanceTo(double x, double y);
}
```

Hope Dunner

Hesham Auda

CSC 221

Next I created the color class. I imported JavaFX color/ paint package for this class. The enum class and filled the shapes with colors and served as a hierarchy class to my main class.



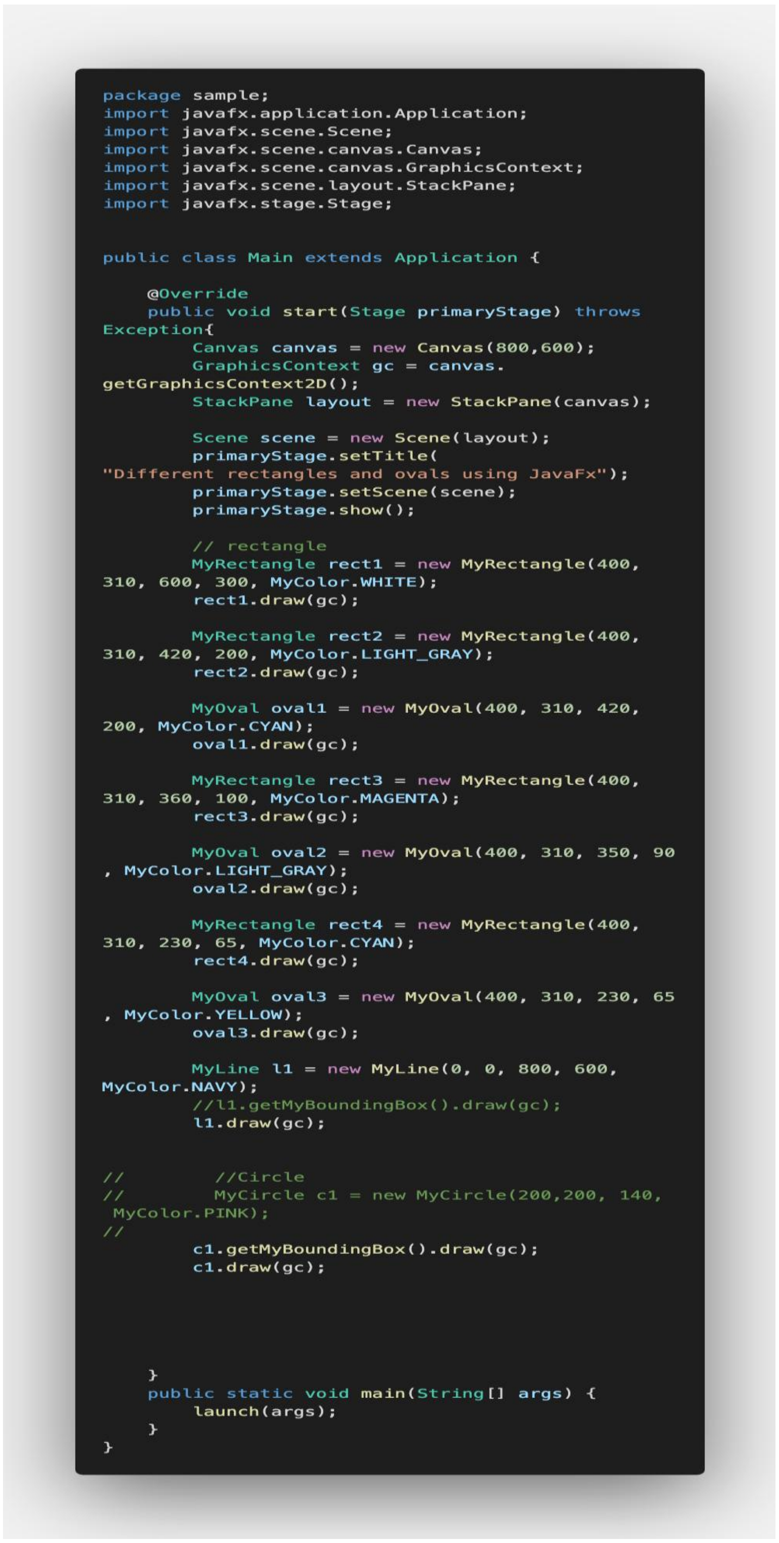
```
package sample;
import javafx.scene.paint.Color;

public enum MyColor{
    BLACK(0, 0, 0), BLUE(0, 0, 255), CYAN(0, 255,
255), DARK_GRAY(169, 169, 169), GRAY(128, 128, 128),
    GREEN(0, 255, 0), MAGENTA(227, 214, 255
), ORANGE(255, 255, 0), LIGHT_GRAY(211, 211, 211),
    RED(255, 0, 0), WHITE(255, 255, 255), YELLOW(255
, 255, 0), PINK(255, 192, 203), NAVY (0, 0, 128);

    private Color color;

    // Constructor

    MyColor(int r, int g, int b) {
        this.color = Color.rgb (r,g,b);
    }
    public Color getMyColor(){ return color; }
}
```

Finally I created the main function which calls on each shape and size using the previously implemented classes. I have 4 rectangles with 3 ovals inscribed similar to the one the professor provided in the write out. Finally I included my line which was diagonally drawn across the larger rectangle. I used the specific imports of scene to set the size of my screen, canvas to set the size of my canvas where I would draw the image, stage, stack pane, and graphics context. These were all necessary packages so I could see my image in real-time.

Hope Dunner

Hesham Auda

CSC 221



```
package sample;
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;


public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws
Exception{
        Canvas canvas = new Canvas(800,600);
        GraphicsContext gc = canvas.
getGraphicsContext2D();
        StackPane layout = new StackPane(canvas);

        Scene scene = new Scene(layout);
        primaryStage.setTitle(
"Different rectangles and ovals using JavaFx");
        primaryStage.setScene(scene);
        primaryStage.show();

        // rectangle
        MyRectangle rect1 = new MyRectangle(400,
310, 600, 300, MyColor.WHITE);
        rect1.draw(gc);

        MyRectangle rect2 = new MyRectangle(400,
310, 420, 200, MyColor.LIGHT_GRAY);
        rect2.draw(gc);

        MyOval oval1 = new MyOval(400, 310, 420,
200, MyColor.CYAN);
        oval1.draw(gc);

        MyRectangle rect3 = new MyRectangle(400,
310, 360, 100, MyColor.MAGENTA);
        rect3.draw(gc);

        MyOval oval2 = new MyOval(400, 310, 350, 90
, MyColor.LIGHT_GRAY);
        oval2.draw(gc);

        MyRectangle rect4 = new MyRectangle(400,
310, 230, 65, MyColor.CYAN);
        rect4.draw(gc);

        MyOval oval3 = new MyOval(400, 310, 230, 65
, MyColor.YELLOW);
        oval3.draw(gc);

        MyLine l1 = new MyLine(0, 0, 800, 600,
MyColor.NAVY);
        //l1.getMyBoundingBox().draw(gc);
        l1.draw(gc);


//        //Circle
//        MyCircle c1 = new MyCircle(200,200, 140,
 MyColor.PINK);
//
        c1.getMyBoundingBox().draw(gc);
        c1.draw(gc);




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

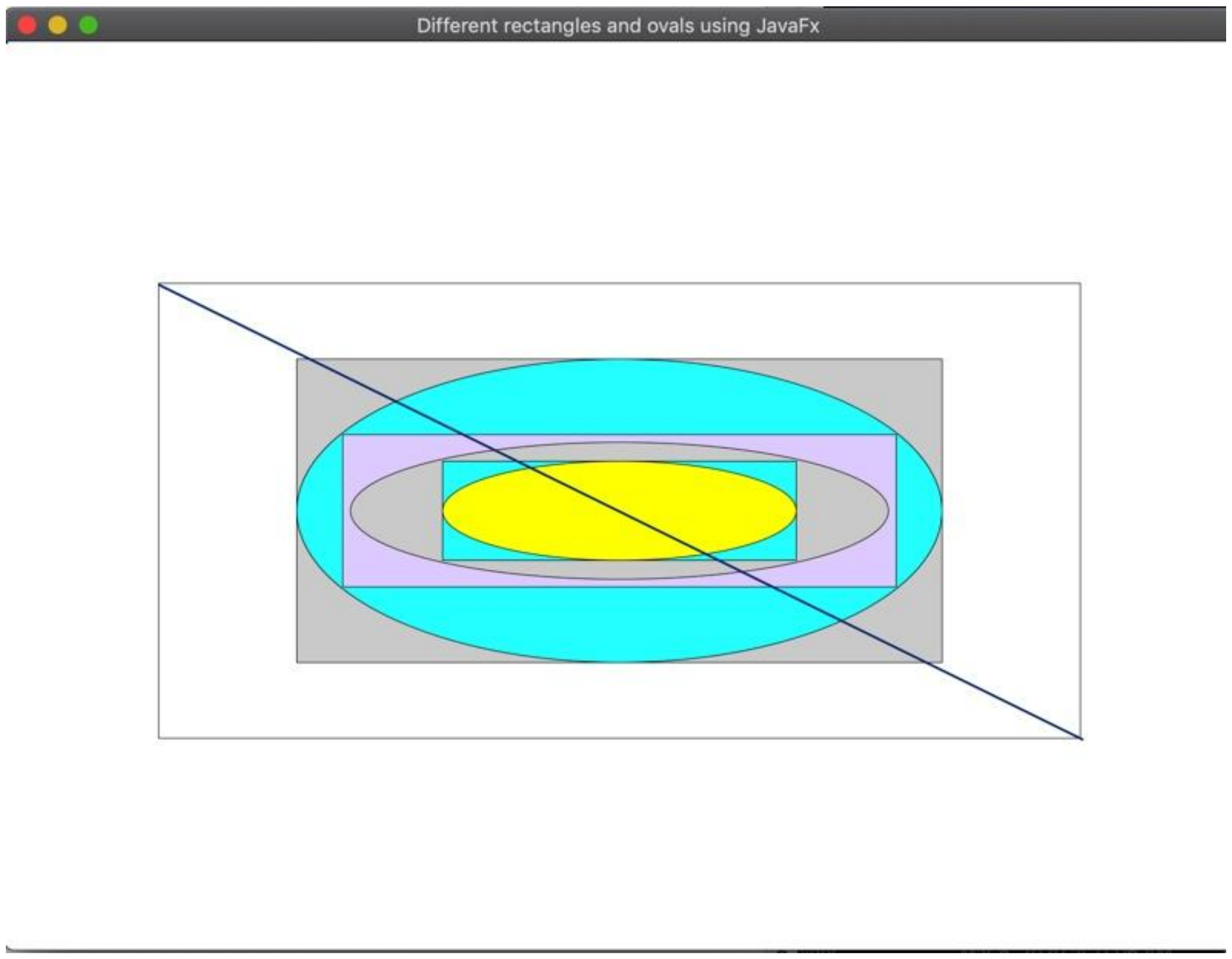
Hope Dunner

Hesham Auda

CSC 221

Finally here is the output

;

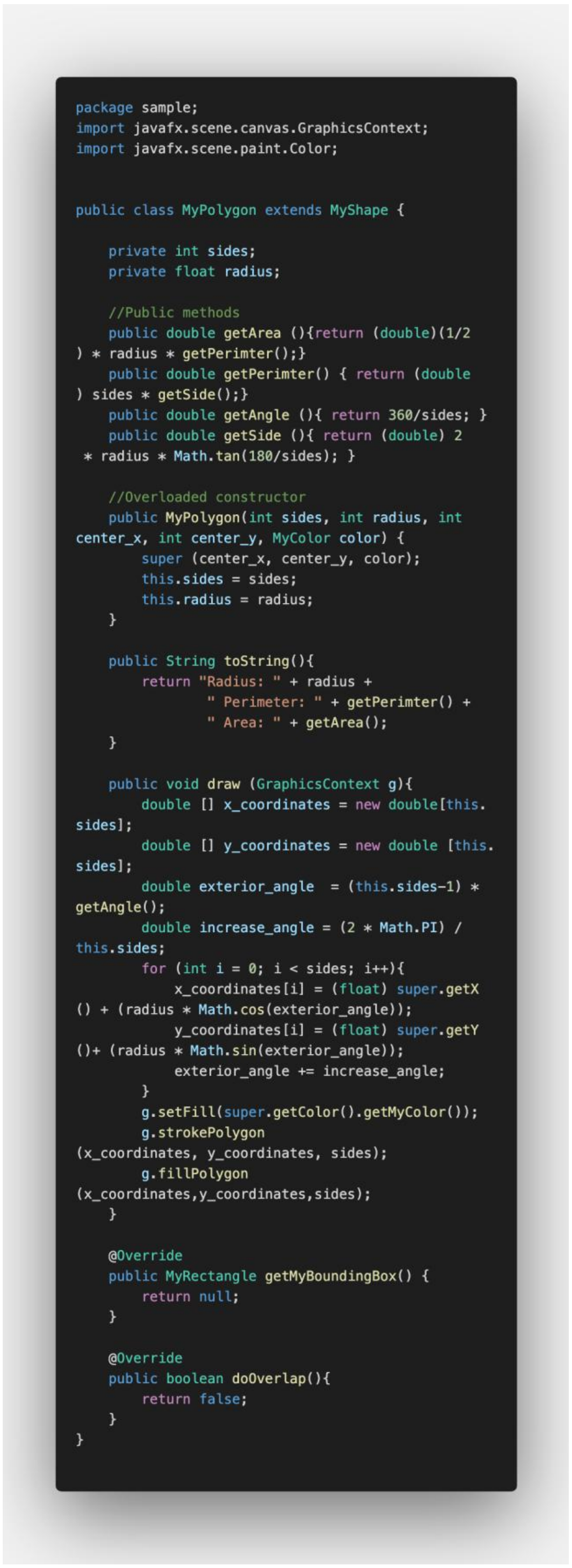


From the first project, there were some classes from exercise 1 that were defined and not utilized. Those classes were; MyPolygon, MyLine, and MyCircle. Similar to project1, I had them implemented so I decided to post the code snippets down below.

Hope Dunner

Hesham Auda

CSC 221

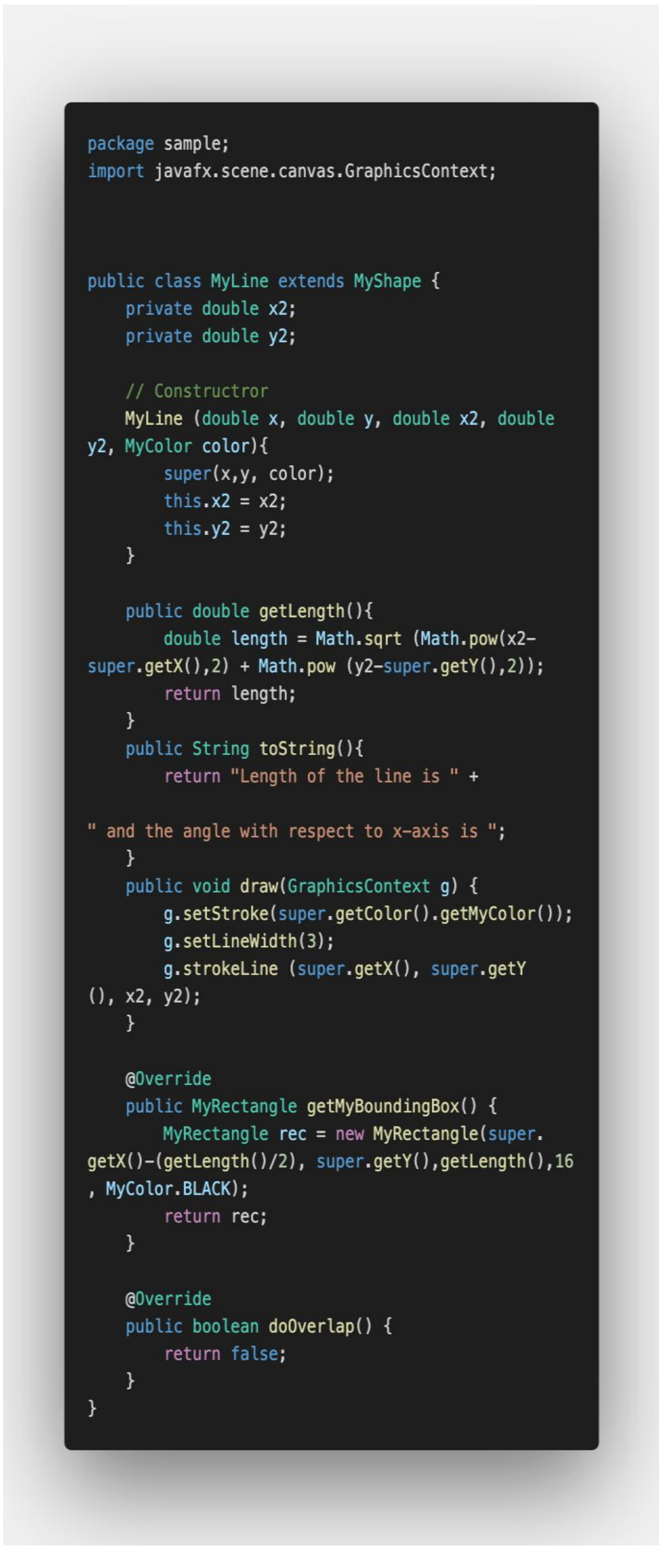


```
package sample;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;


public class MyPolygon extends MyShape {

    private int sides;
    private float radius;

    //Public methods
    public double getArea (){return (double)(1/2
) * radius * getPerimter();}
    public double getPerimter() { return (double
) sides * getSide();}
    public double getAngle (){ return 360/sides; }
    public double getSide (){ return (double) 2
 * radius * Math.tan(180/sides); }

    //Overloaded constructor
    public MyPolygon(int sides, int radius, int
center_x, int center_y, MyColor color) {
        super (center_x, center_y, color);
        this.sides = sides;
        this.radius = radius;
    }

    public String toString(){
        return "Radius: " + radius +
                " Perimeter: " + getPerimter() +
                " Area: " + getArea();
    }

    public void draw (GraphicsContext g){
        double [] x_coordinates = new double[this.
sides];
        double [] y_coordinates = new double [this.
sides];
        double exterior_angle  = (this.sides-1) *
getAngle();
        double increase_angle = (2 * Math.PI) /
this.sides;
        for (int i = 0; i < sides; i++){
            x_coordinates[i] = (float) super.getX
() + (radius * Math.cos(exterior_angle));
            y_coordinates[i] = (float) super.getY
()+ (radius * Math.sin(exterior_angle));
            exterior_angle += increase_angle;
        }
        g.setFill(super.getColor().getMyColor());
        g.strokePolygon
(x_coordinates, y_coordinates, sides);
        g.fillPolygon
(x_coordinates,y_coordinates,sides);
    }

    @Override
    public MyRectangle getMyBoundingBox() {
        return null;
    }

    @Override
    public boolean doOverlap(){
        return false;
    }
}
```

Hope Dunner

Hesham Auda

CSC 221

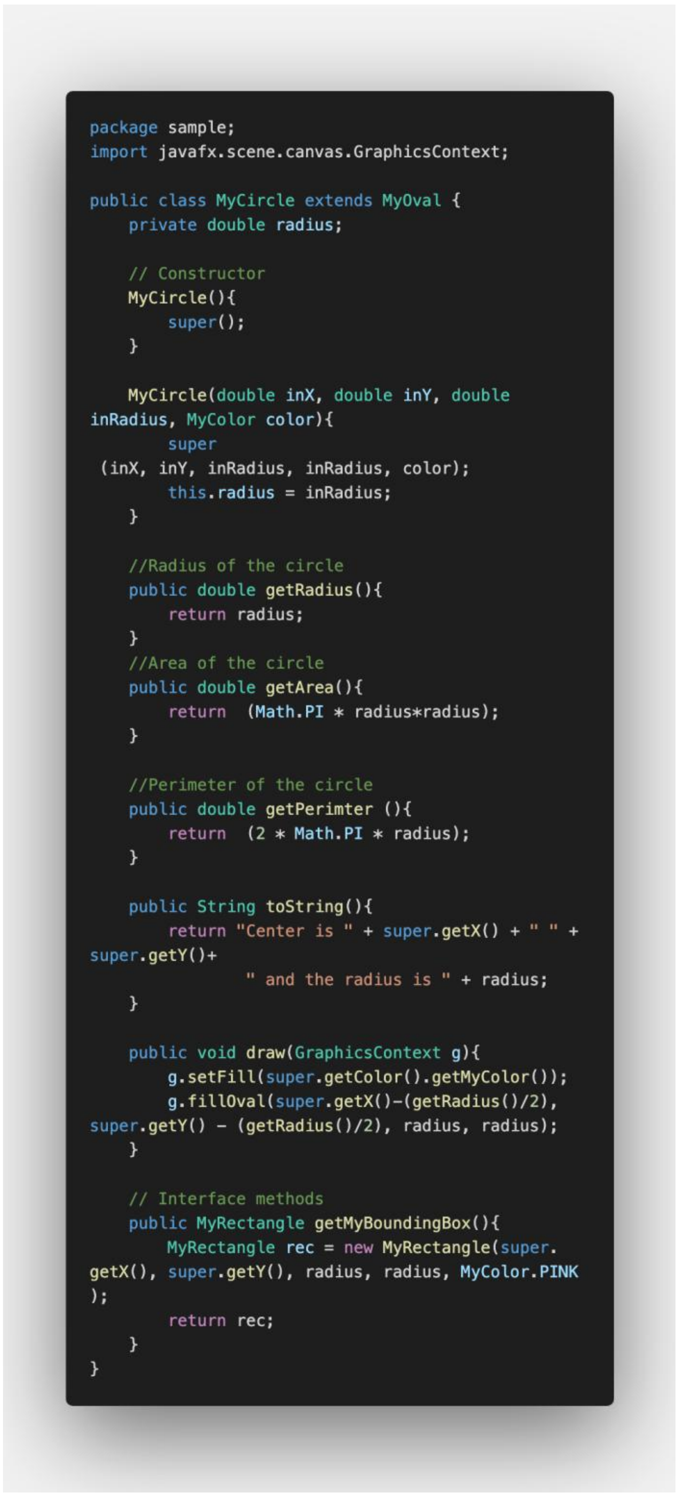


```
package sample;
import javafx.scene.canvas.GraphicsContext;



public class MyLine extends MyShape {
    private double x2;
    private double y2;

    // Constructror
    MyLine (double x, double y, double x2, double
y2, MyColor color){
        super(x,y, color);
        this.x2 = x2;
        this.y2 = y2;
    }

    public double getLength(){
        double length = Math.sqrt (Math.pow(x2-
super.getX(),2) + Math.pow (y2-super.getY(),2));
        return length;
    }
    public String toString(){
        return "Length of the line is " +

" and the angle with respect to x-axis is ";
    }
    public void draw(GraphicsContext g) {
        g.setStroke(super.getColor().getMyColor());
        g.setLineWidth(3);
        g.strokeLine (super.getX(), super.getY
(), x2, y2);
    }

    @Override
    public MyRectangle getMyBoundingBox() {
        MyRectangle rec = new MyRectangle(super.
getX()-(getLength()/2), super.getY(),getLength(),16
, MyColor.BLACK);
        return rec;
    }

    @Override
    public boolean doOverlap() {
        return false;
    }
}
```

Hope Dunner

Hesham Auda

CSC 221



```
package sample;
import javafx.scene.canvas.GraphicsContext;

public class MyCircle extends MyOval {
    private double radius;

    // Constructor
    MyCircle(){
        super();
    }

    MyCircle(double inX, double inY, double
inRadius, MyColor color){
        super
 (inX, inY, inRadius, inRadius, color);
        this.radius = inRadius;
    }

    //Radius of the circle
    public double getRadius(){
        return radius;
    }
    //Area of the circle
    public double getArea(){
        return  (Math.PI * radius*radius);
    }

    //Perimeter of the circle
    public double getPerimter (){
        return  (2 * Math.PI * radius);
    }

    public String toString(){
        return "Center is " + super.getX() + " " +
super.getY()+
                " and the radius is " + radius;
    }

    public void draw(GraphicsContext g){
        g.setFill(super.getColor().getMyColor());
        g.fillOval(super.getX()-(getRadius()/2),
super.getY() - (getRadius()/2), radius, radius);
    }

    // Interface methods
    public MyRectangle getMyBoundingBox(){
        MyRectangle rec = new MyRectangle(super.
getX(), super.getY(), radius, radius, MyColor.PINK
);
        return rec;
    }
}
```

Hope Dunner

Hesham Auda

CSC 221

All 3 classes follow the same format as the other shape classes, but for the sake of readability, and not having this project continue for another 3 pages, I have just attached the code snippets. I explained how those classes were built in my first exercise. For this project in particular they were not used, only defined. All the instruction for these 3 classes, stemmed from the first project.

Overall, I put a ton more effort into this project and I hope it shows! I had a fun time with implementing my shapes and seeing my colors, I hope you enjoy reading through my work, as it took many hours to complete.