SOFTWARE DESIGN LABORATORY

Josh Miranda

Hesham Auda

Section P

09 March 2021

1. Problem

Create a hierarchy of Java classes as follows:

MyLine is_a MyShape;

MyRectangle is_a MyShape;

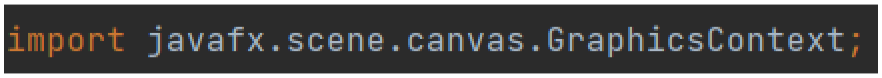
MyOval is_a MyShape.

The problem that we need to solve in this project is to use the JavaFX package to create a hierarchy of Java classes to illustrate different shapes. These shapes are subclasses of the main class MyShape that include: MyLine, MyRectangle and MyOval. There is also another subclass called MyColor that allows us to color the shapes and differentiate them easily when we create our illustration. This report will explain the solution methods to get to the output.

2. Solution Methods

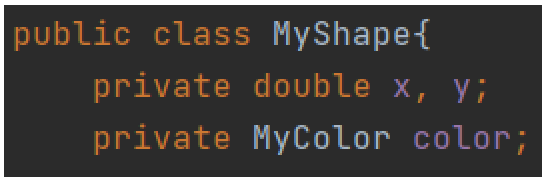
Class MyShape

The first method that will be talked about is the class MyShape. MyShape is the hierarchy's superclass and allows other classes to inherit from it. This class will have the reference points (x, y) and the color of the shapes.



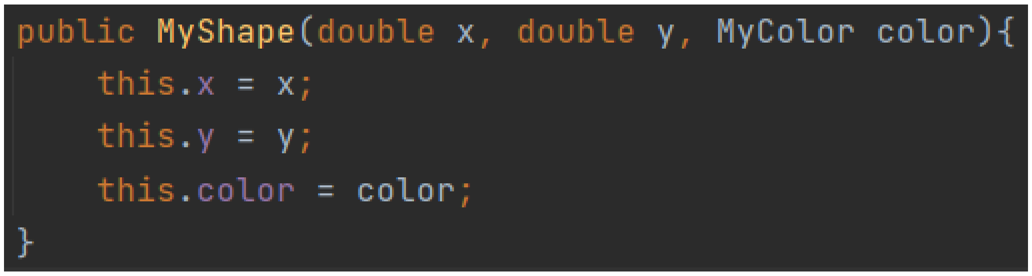
```
import javafx.scene.canvas.GraphicsContext;
```

To draw the method on the screen we need to use the **JavaFX** package. This was done by **import javafx.scene.canvas.GraphicsContext;**.



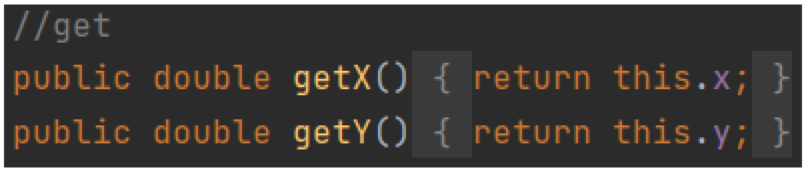
```
public class MyShape{
    private double x, y;
    private MyColor color;
```

The public class MyShape is where we will declare the reference points (x, y) and the color for our shapes. Both are private so that it is only accessible within the class.



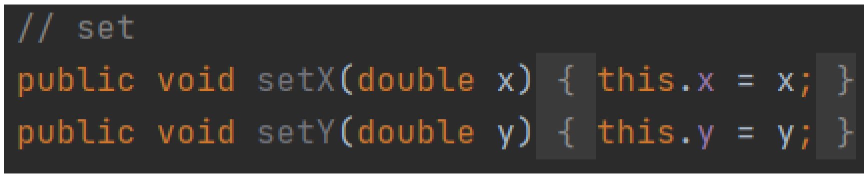
```java
public MyShape(double x, double y, MyColor color){
    this.x = x;
    this.y = y;
    this.color = color;
}
```

This is the **public MyShape** constructor that holds the parameters **x**, **y** and **MyColor**. Inside we are assigning variables to **this** to eliminate confusion and assign them to the current instance. Assigning x to **this.x** and the same is done with y and color.



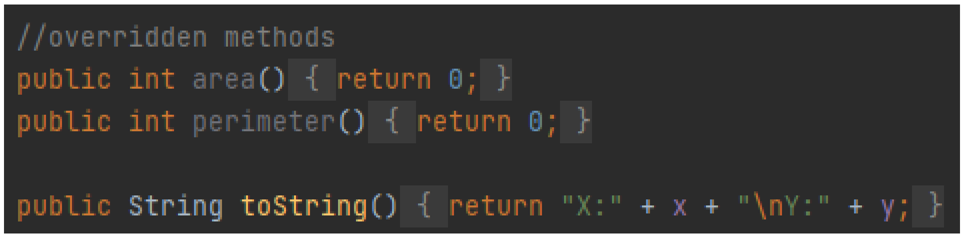
```java
//get
public double getX() { return this.x; }
public double getY() { return this.y; }
```

In the beginning of our code we declared private variables **x**, **y**, and **MyColor**. In order to access them we need to use the **get** method, **getX()** to **return this.x**. This is also done with y, **getY()** to return **this.y**.



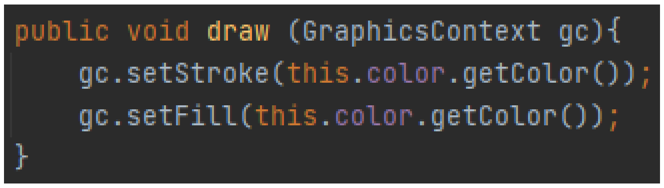
```java
// set
public void setX(double x) { this.x = x; }
public void setY(double y) { this.y = y; }
```

Since we have the **get** method we must also use **set**. The **set** method takes the parameter and sets it to a new value. The method **setX()** takes in the parameter **double x** and assigns it to the **x** variable, this is the same for **setY()**.



```java
//overridden methods
public int area() { return 0; }
public int perimeter() { return 0; }

public String toString() { return "X:" + x + "\nY:" + y; }
```

Here we have overridden methods **area** and **perimeter** that **return 0;**. Underneath is the **toString** method that returns the objects description as a string. This method returns the values of **x** and **y**. In each subclass in the hierarchy they are both overridden.

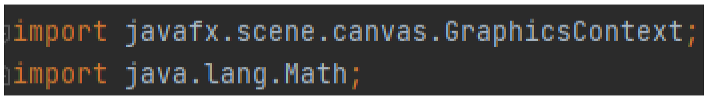


```java
public void draw (GraphicsContext gc){
    gc.setStroke(this.color.getColor());
    gc.setFill(this.color.getColor());
}
```

Lastly we need a draw method which uses the **JavaFX** package we imported earlier called **GraphicsContext**. With this package we can **setStroke** and **setFill** with a color for our shape.

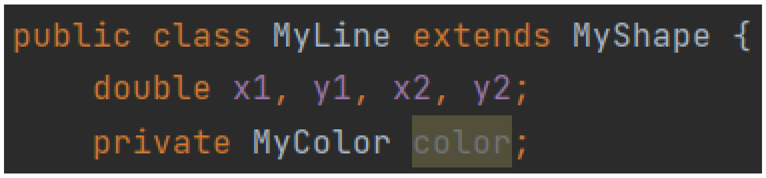
Class MyLine

The **Class MyLine** will be used to create a straight line by using the endpoints (x1, y1) and (x2, y2).



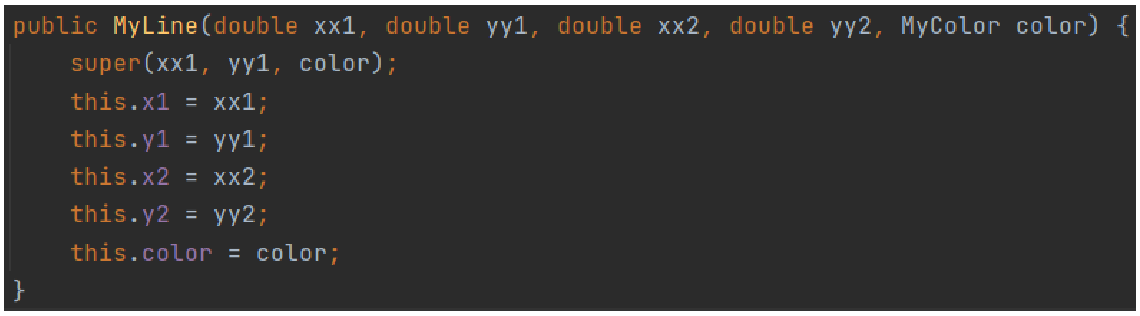
```java
import javafx.scene.canvas.GraphicsContext;
import java.lang.Math;
```

Similarly for the **Class MyShape** we will be importing **GraphicsContext** again and also **import java.lang.Math**. **GraphicsContext** for drawing the line and **Math** since we will be using math calculations.



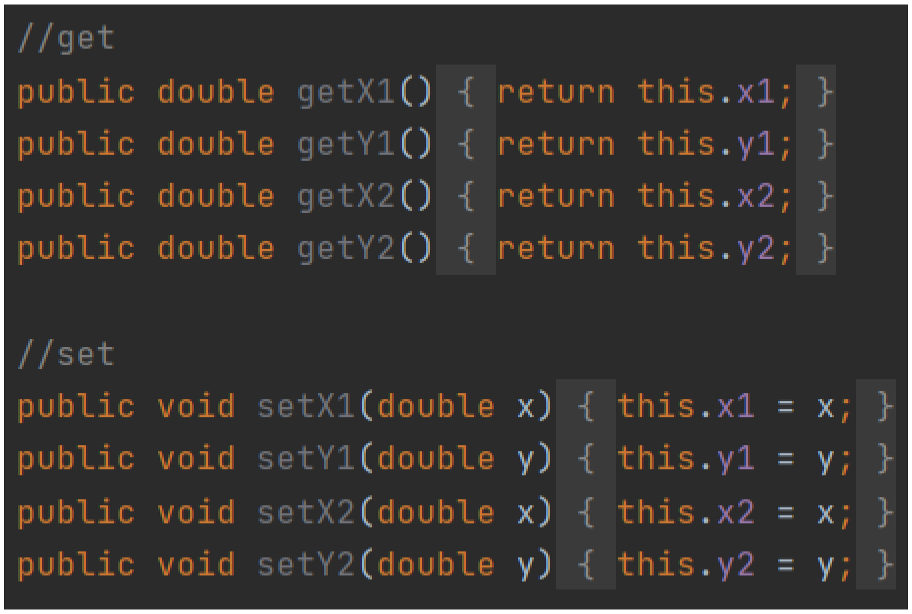
```java
public class MyLine extends MyShape {
    double x1, y1, x2, y2;
    private MyColor color;
```

We have **public class Myline** but now we are extending it to **MyShape**. This allows **MyLine** to inherit from **MyShape** since **MyShape** is the hierarchy’s superclass. In this public class we are declaring our variables: **x1**, **y1**, **x2**, **y2**; and **color**.



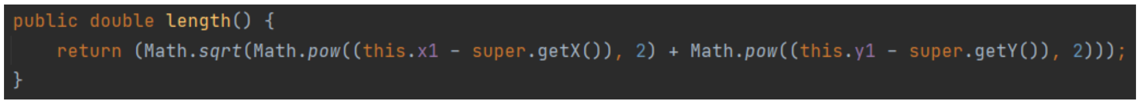
```java
public MyLine(double xx1, double yy1, double xx2, double yy2, MyColor color) {
    super(xx1, yy1, color);
    this.x1 = xx1;
    this.y1 = yy1;
    this.x2 = xx2;
    this.y2 = yy2;
    this.color = color;
}
```

This is the **MyLine** constructor that takes in parameters: **xx1**, **yy1**, **xx2**, **yy2** and **color**. Below we are using **super** to refer to the superclass **MyShape**. It is to eliminate confusion for similar named methods and access methods in the super class. Beneath we doing the same technique we used in **MyShape** by using **this**. We are assigning variables to **this** to eliminate confusion and assign them to the current instance. Assigning **xx1** to **this.x1** and the same is done below for the rest.



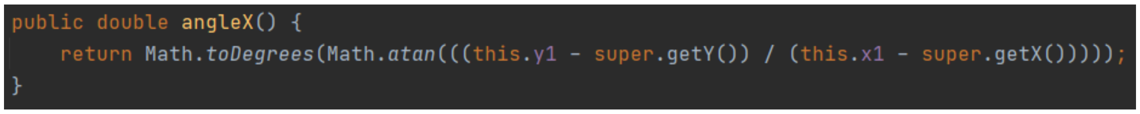
```
//get
public double getX1() { return this.x1; }
public double getY1() { return this.y1; }
public double getX2() { return this.x2; }
public double getY2() { return this.y2; }

//set
public void setX1(double x) { this.x1 = x; }
public void setY1(double y) { this.y1 = y; }
public void setX2(double x) { this.x2 = x; }
public void setY2(double y) { this.y2 = y; }
```

The start of the **class MyLine** we declared private variables so now we need to use the **get** and **set** method pair. Using **get** to access the private variables and **set** to take the parameter and set it to a new value similarly to what we have done in the **class MyShape**.



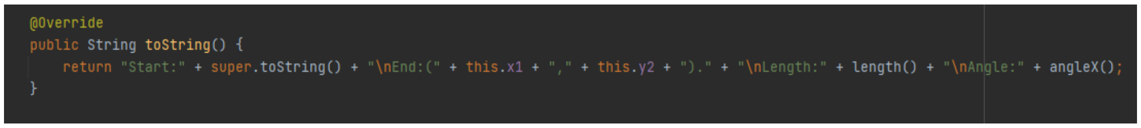
```
public double length() {
    return (Math.sqrt(Math.pow((this.x1 - super.getX()), 2) + Math.pow((this.y1 - super.getY()), 2)));
}
```

To get the length of the line we must use the distance formula $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. This is where importing **java.lang.Math** comes in, allowing the use of square rooting with **Math.sqrt** and power with **Math.pow**. Inside the parenthesis we use **this.x1** to be specific to the **x1** variable within the class and **super.getX()** to access the **x** values from **class myShape**. This is likewise for **this.y1** and **super.getY()**. In the end the distance formula is being returned.



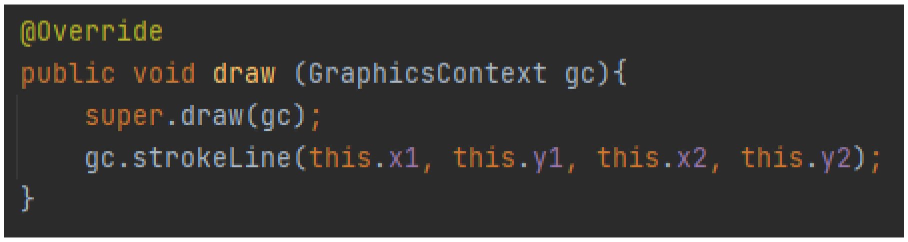
```java
public double angleX() {
    return Math.toDegrees(Math.atan(((this.y1 - super.getY()) / (this.x1 - super.getX()))));
}
```

To get the angle of the line we use $\theta = tan^{-1}(\frac{\Delta y}{\Delta x})$. Through importing **java.lang.Math** we have access to **Math.toDegrees** converting our answer to degrees and **Math.atan** which is arctan. Inside the parenthesis we use **this.y1** to be specific to the **y1** variable within the class and **super.getY()** to access the **y** values from **class myShape**. This is likewise for **this.x1** and **super.getX()**. All this returns the angle of the line.



```java
@Override
public String toString() {
    return "Start:" + super.toString() + "\nEnd:(" + this.x1 + "," + this.y2 + ")." + "\nLength:" + length() + "\nAngle:" + angleX();
}
```

Here is the **toString()** method that returns the start, ends points, length() and angleX(). This method is overridden because it is following instructions separate from the superclass of **MyShape**. We used **super** on **toString()** to access the values from the superclass **MyShape**. Once again we use **this.x1** and **this.x2** to be specific to the values within the class.

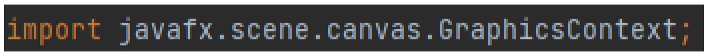


```java
@Override
public void draw (GraphicsContext gc){
    super.draw(gc);
    gc.strokeLine(this.x1, this.y1, this.x2, this.y2);
}
```

Similar to the **toString()** method, the **draw** method is overridden due to it following its own instructions separate from superclass **MyShape**. Since the method is being overridden it allows the use of **super.draw(gc)**. Below is the function **strokeLine** from **Graphicscontext** containing the parameters: **this.x1**, **this.y1**, **this.x2** and **this.y2**. This method displays the MyLine object.

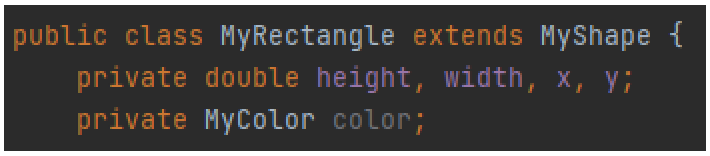
Class MyRectangle

The **Class MyRectangle** creates a rectangle using height h and width w, and top left corner point (x. y).



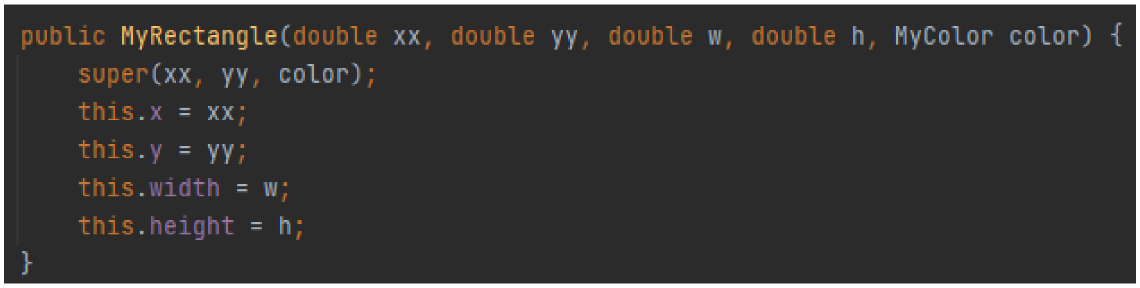
```java
import javafx.scene.canvas.GraphicsContext;
```

Imports will be similar to **MyShape** and **MyLin**e using **javafxscene.canvas.GraphicsContext.**



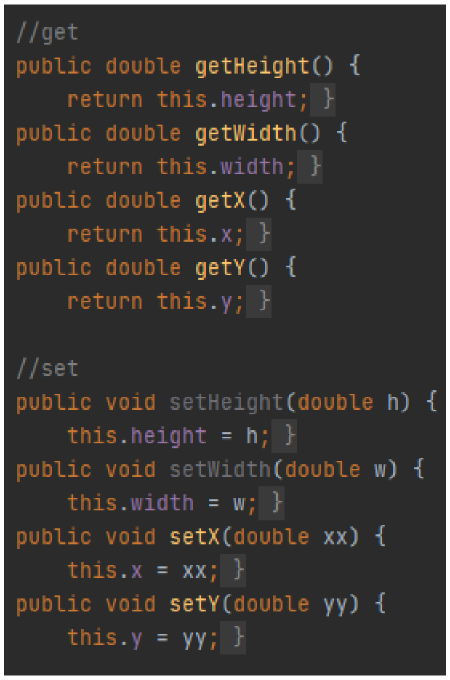
```
public class MyRectangle extends MyShape {
    private double height, width, x, y;
    private MyColor color;
```

The **public class MyRectangle** is extended to **MyShape** allowing **MyRectangle** to inherit from **MyShape**. Declared variables are **height**, **width**, **x**, **y** and **color**.



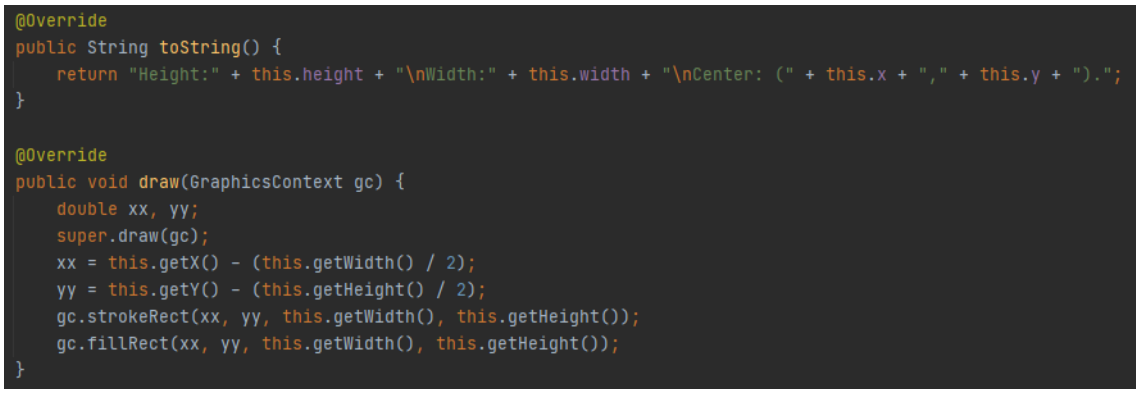
```
public MyRectangle(double xx, double yy, double w, double h, MyColor color) {
    super(xx, yy, color);
    this.x = xx;
    this.y = yy;
    this.width = w;
    this.height = h;
}
```

This **MyRectangle** constructor takes in parameters: **xx**, **yy**, **w**, **h** and **color**. They are all taken in as type **doubles** except for **color**. Below we are using **super** to refer to the superclass **MyShape**. It is to eliminate confusion for similar named methods and access methods in the super class. Beneath we doing the same technique we used in **MyShape** and **MyLine** by using **this**. We are assigning variables to **this** to eliminate confusion and assign them to the current instance. Assigning **w** to **this.width** and the same is done below for the rest.



```java
//get
public double getHeight() {
    return this.height; }
public double getWidth() {
    return this.width; }
public double getX() {
    return this.x; }
public double getY() {
    return this.y; }

//set
public void setHeight(double h) {
    this.height = h; }
public void setWidth(double w) {
    this.width = w; }
public void setX(double xx) {
    this.x = xx; }
public void setY(double yy) {
    this.y = yy; }
```

The start of the code we declared our private variables for **MyRectangle** so this means we need to use the **get** and **set** method pair. This allows access to private variables from **get**, and **set** gives the ability to take the parameter and set it to a new variable just like what we did in **class MyShape** and **class MyLine**.

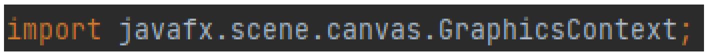


```java
@Override
public String toString() {
    return "Height:" + this.height + "\nWidth:" + this.width + "\nCenter: (" + this.x + "," + this.y + ").";
}

@Override
public void draw(GraphicsContext gc) {
    double xx, yy;
    super.draw(gc);
    xx = this.getX() - (this.getWidth() / 2);
    yy = this.getY() - (this.getHeight() / 2);
    gc.strokeRect(xx, yy, this.getWidth(), this.getHeight());
    gc.fillRect(xx, yy, this.getWidth(), this.getHeight());
}
```

The two overridden methods are **toString()** and **draw**. The **toString()** method returns the **height**, **width** and coordinates (x,y). For all **toString()** methods they will have similar formats except for the **class MyShape** since that is the superclass. The **draw** method uses the **GraphicsContext** that allows us to draw the rectangle. In the method we declare the variables **xx** and **yy** as type **double**. The **super.draw(gc)** overrides the **draw** method in **MyShape**. Then we calculate the center of the rectangle with **xx = this.getX() - (this.getWidth() / 2);** and **yy = this.getY() - (this.getHeight() / 2);**. After calculating the center we will be able to use it as a parameter for **strokeRect()** and **fillRect()** with **GraphicsContext**.

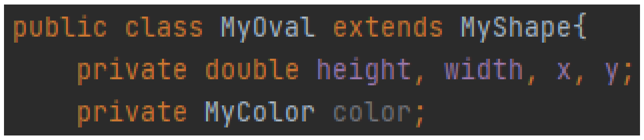
Class MyOval

The code for the **Class MyOval** is identical to the **MyRectangle** class but instead creates an oval using height h and width w, and top left corner point (x. y).



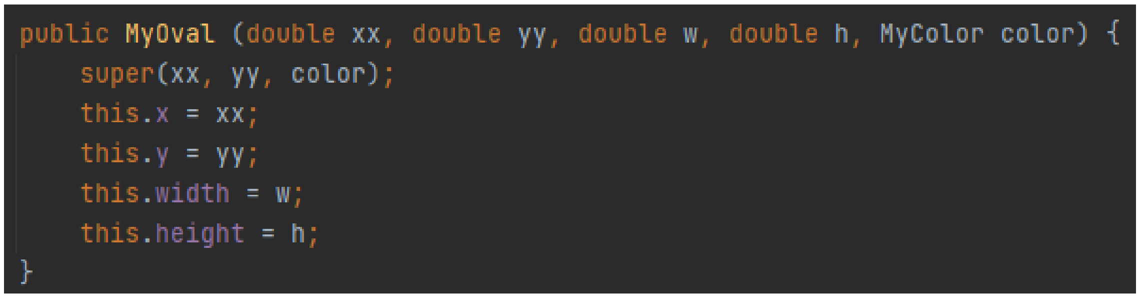
```
import javafx.scene.canvas.GraphicsContext;
```

Imports will be similar to **MyShape**, **MyLin**e and **MyRectangle** using **javafxscene.canvas.GraphicsContext**.



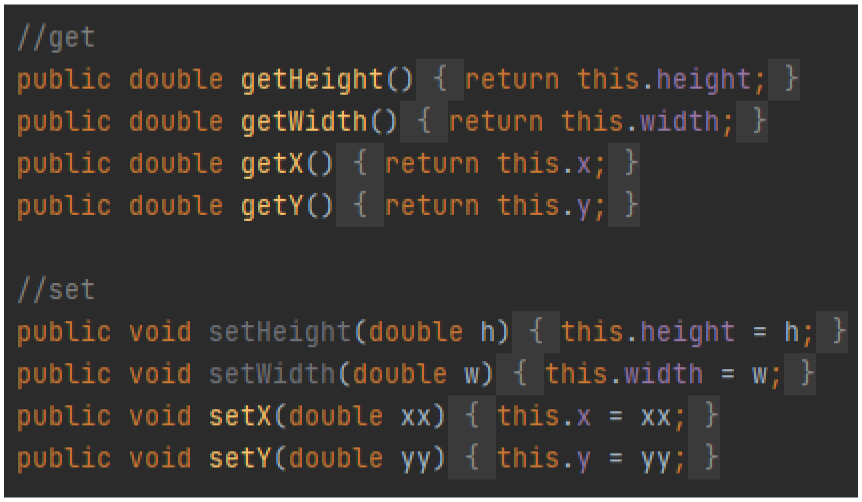
```
public class MyOval extends MyShape{
    private double height, width, x, y;
    private MyColor color;
```

The **public class MyOval** is extended to **MyShape** allowing **MyOval** to inherit from **MyShape**. Declared variables are **height**, **width**, **x**, **y** and **color**.



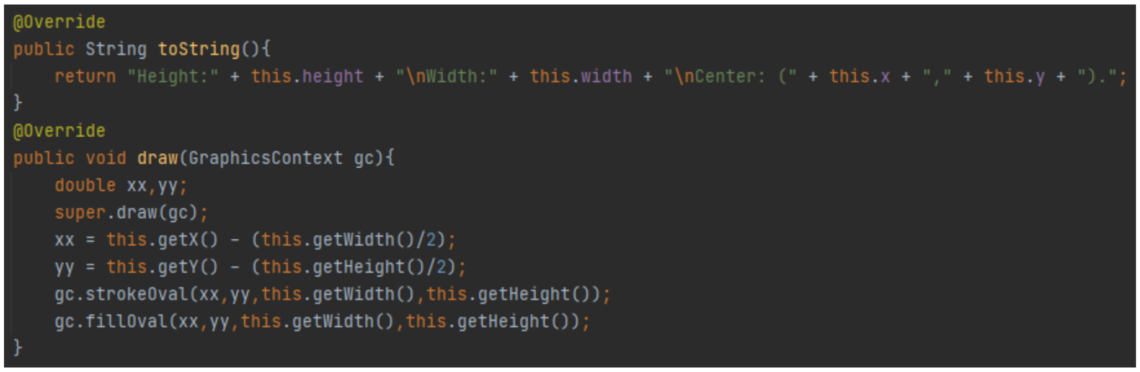
```
public MyOval (double xx, double yy, double w, double h, MyColor color) {
    super(xx, yy, color);
    this.x = xx;
    this.y = yy;
    this.width = w;
    this.height = h;
}
```

Similar to **MyRectangle**, the **MyOval** constructor takes in parameters: **xx**, **yy**, **w**, **h** and **color**. They are all taken in as type **doubles** except for **color**. Below we are using **super** to refer to the superclass **MyShape**. It is to eliminate confusion for similar named methods and access methods in the super class. Beneath we are doing the same technique we used in **MyShape**, **MyLine** and **MyRectangle** by using **this**. We are assigning variables to **this** to eliminate confusion and assign them to the current instance. Assigning **w** to **this.width** and the same is done below for the rest.



```java
//get
public double getHeight() { return this.height; }
public double getWidth() { return this.width; }
public double getX() { return this.x; }
public double getY() { return this.y; }

//set
public void setHeight(double h) { this.height = h; }
public void setWidth(double w) { this.width = w; }
public void setX(double xx) { this.x = xx; }
public void setY(double yy) { this.y = yy; }
```

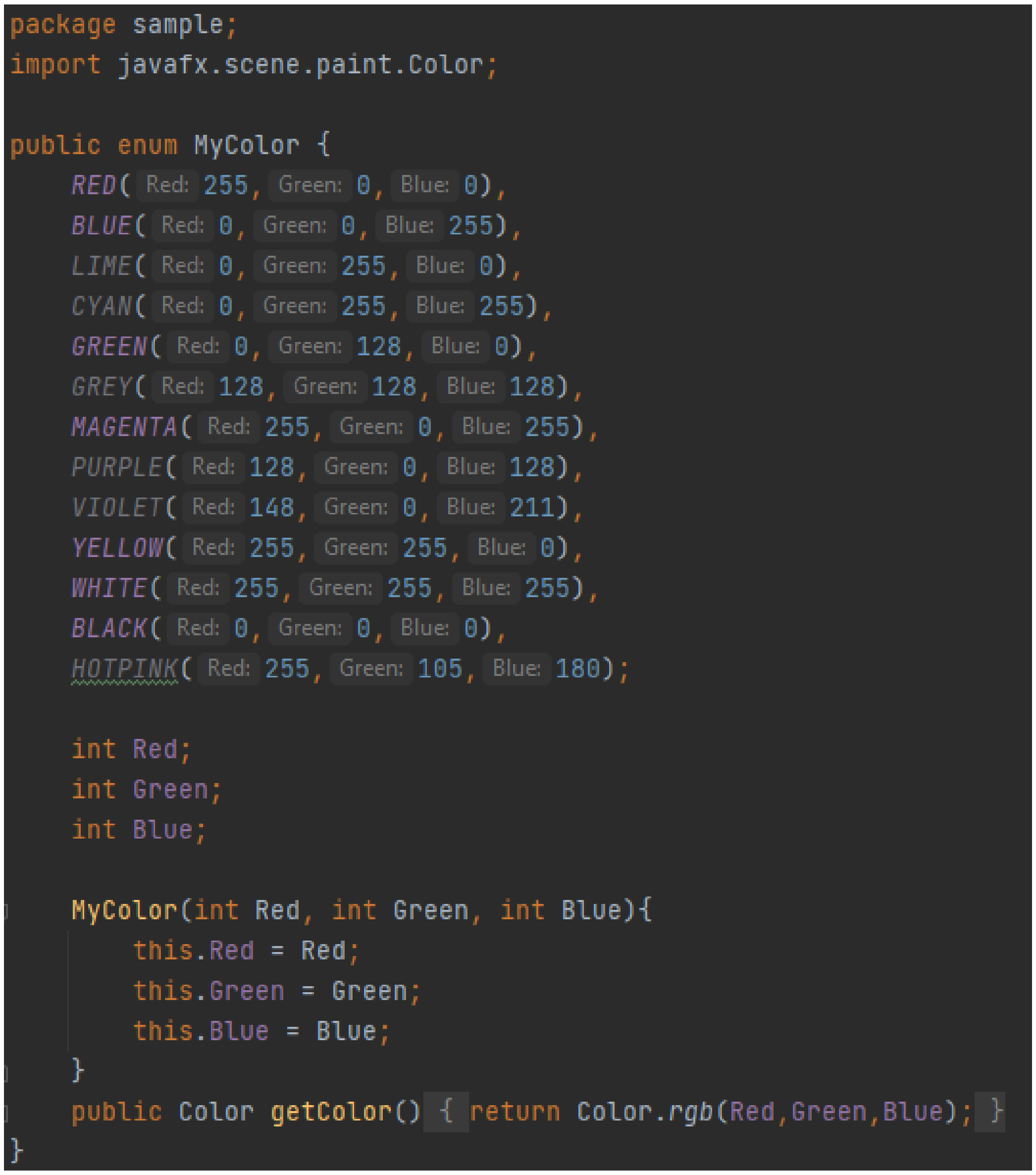
Declared above are our private variables for **MyOval** so this means we need to use the **get** and **set** method pair. This allows access to private variables from **get**, and **set** gives the ability to take the parameter and set it to a new variable just like what we did in **class MyShape**, **class MyLine** and **class MyRectangle**.



```java
@Override
public String toString(){
    return "Height:" + this.height + "\nWidth:" + this.width + "\nCenter: (" + this.x + "," + this.y + ").";
}
@Override
public void draw(GraphicsContext gc){
    double xx,yy;
    super.draw(gc);
    xx = this.getX() - (this.getWidth()/2);
    yy = this.getY() - (this.getHeight()/2);
    gc.strokeOval(xx,yy,this.getWidth(),this.getHeight());
    gc.fillOval(xx,yy,this.getWidth(),this.getHeight());
}
```

The two overridden methods are **toString()** and **draw**. The **toString()** method returns the **height**, **width** and coordinates (x,y). For all **toString()** methods they will have similar formats except for the **class MyShape** since that is the superclass. The **draw** method uses the **GraphicsContext** that allows us to draw the rectangle. In the method we declare the variables **xx** and **yy** as type **double**. The **super.draw(gc)** overrides the **draw** method in **MyShape**. Then we calculate the center of the rectangle with **xx = this.getX() - (this.getWidth() / 2);** and **yy = this.getY() - (this.getHeight() / 2);**. After calculating the center we will be able to use it as a parameter for **strokeRect()** and **fillRect()** with **GraphicsContext**.

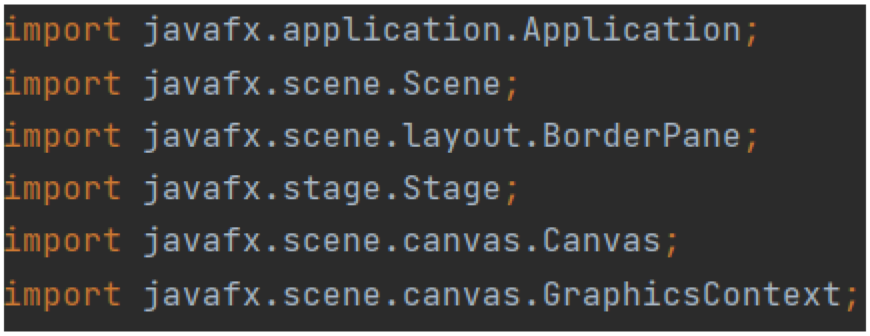
Class MyColor



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

public enum MyColor {
    RED( Red: 255, Green: 0, Blue: 0),
    BLUE( Red: 0, Green: 0, Blue: 255),
    LIME( Red: 0, Green: 255, Blue: 0),
    CYAN( Red: 0, Green: 255, Blue: 255),
    GREEN( Red: 0, Green: 128, Blue: 0),
    GREY( Red: 128, Green: 128, Blue: 128),
    MAGENTA( Red: 255, Green: 0, Blue: 255),
    PURPLE( Red: 128, Green: 0, Blue: 128),
    VIOLET( Red: 148, Green: 0, Blue: 211),
    YELLOW( Red: 255, Green: 255, Blue: 0),
    WHITE( Red: 255, Green: 255, Blue: 255),
    BLACK( Red: 0, Green: 0, Blue: 0),
    HOTPINK( Red: 255, Green: 105, Blue: 180);

    int Red;
    int Green;
    int Blue;

    MyColor(int Red, int Green, int Blue){
        this.Red = Red;
        this.Green = Green;
        this.Blue = Blue;
    }
    public Color getColor() { return Color.rgb(Red,Green,Blue); }
}
```

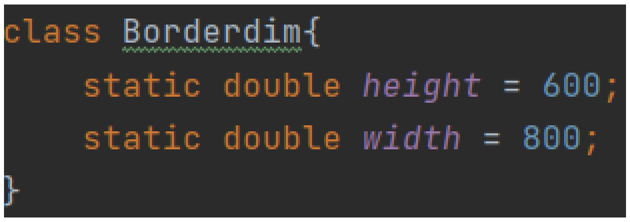
The **Class MyColor** allows us to color our illustrated shapes using RGB **int** values. Within the **enum MyColor** is a bank of colors already with their RGB **int** values. If there is a desired color then the RGB values must be added to the bank.

Class Main



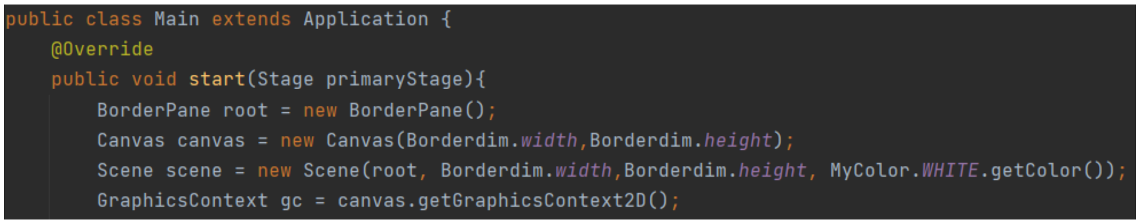
```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
```

The main class has multiple libraries imported compared to the rest of the other classes. Starting off we have the **javafx.application.Application** that allows the launch of the Javafx application. The next library **javafx.scene.Scene** sets up the scene and **javafx.scene.layout.BorderPane** keeps track of the scene such as top, bottom, left, right and center. For **javafx.stage.Stage** allows the use of constructing objects onto the stages like for this project shape. The **javafx.scene.canvas.Canvas** is an image that can be drawn on and works alongside **javafx.scene.canvas.GraphicsContext** which displays the drawn images.



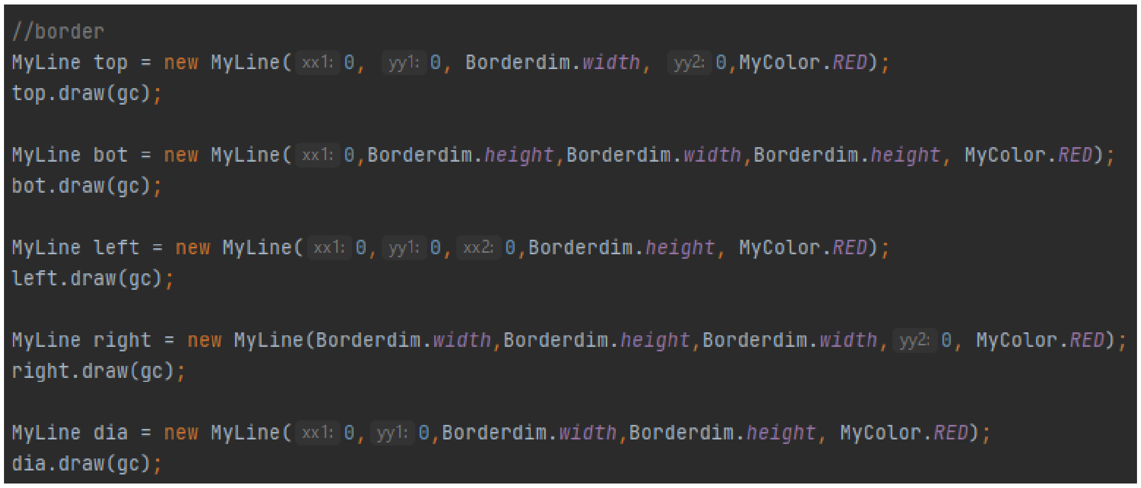
```java
class Borderdim{
    static double height = 600;
    static double width = 800;
}
```

This **class Borderdim** sets the dimensions of the application. By doing this it will allow me to scale my shapes without manually changing the border throughout my code.



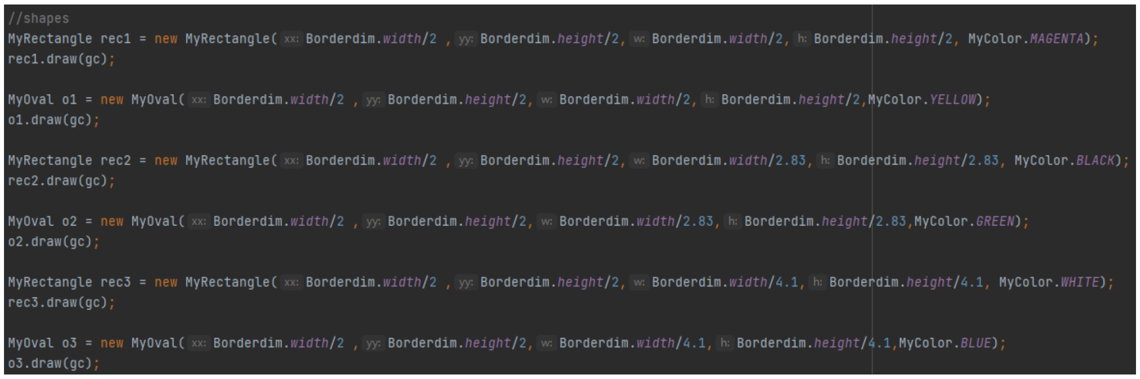
```java
public class Main extends Application {
    @Override
    public void start(Stage primaryStage){
        BorderPane root = new BorderPane();
        Canvas canvas = new Canvas(Borderdim.width,Borderdim.height);
        Scene scene = new Scene(root, Borderdim.width,Borderdim.height, MyColor.WHITE.getColor());
        GraphicsContext gc = canvas.getGraphicsContext2D();
```

This is the foundation of the scene using the **class Boderdim** for the dimensions of the scene. By calling **Boderdim** in **Canvas** and **Scene** the background screen would be scaled if the dimensions were ever to change. **GraphicsContext** is declared as **gc** so that it could be called in other classes.



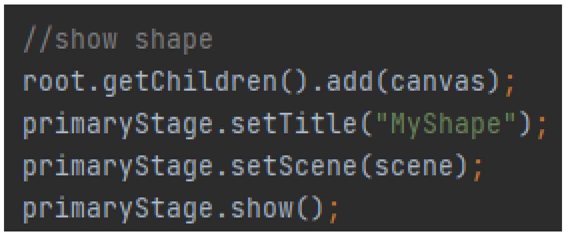
```java
//border
MyLine top = new MyLine( xx1: 0,  yy1: 0, Borderdim.width,  yy2: 0,MyColor.RED);
top.draw(gc);

MyLine bot = new MyLine( xx1: 0,Borderdim.height,Borderdim.width,Borderdim.height, MyColor.RED);
bot.draw(gc);

MyLine left = new MyLine( xx1: 0, yy1: 0, xx2: 0,Borderdim.height, MyColor.RED);
left.draw(gc);

MyLine right = new MyLine(Borderdim.width,Borderdim.height,Borderdim.width, yy2: 0, MyColor.RED);
right.draw(gc);

MyLine dia = new MyLine( xx1: 0, yy1: 0,Borderdim.width,Borderdim.height, MyColor.RED);
dia.draw(gc);
```

In this section we use all our classes combined to create our illustration. First I created a border using the **class MyLine**. I first start with the top border creating a **new Myline** with the parameters containing the dimensions.



```java
//shapes
MyRectangle rec1 = new MyRectangle( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2, h: Borderdim.height/2, MyColor.MAGENTA);
rec1.draw(gc);

MyOval o1 = new MyOval( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2, h: Borderdim.height/2,MyColor.YELLOW);
o1.draw(gc);

MyRectangle rec2 = new MyRectangle( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2.83, h: Borderdim.height/2.83, MyColor.BLACK);
rec2.draw(gc);

MyOval o2 = new MyOval( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2.83, h: Borderdim.height/2.83,MyColor.GREEN);
o2.draw(gc);

MyRectangle rec3 = new MyRectangle( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/4.1, h: Borderdim.height/4.1, MyColor.WHITE);
rec3.draw(gc);

MyOval o3 = new MyOval( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/4.1, h: Borderdim.height/4.1,MyColor.BLUE);
o3.draw(gc);
```

Once the border was formed I drew the first rectangle but wanted to align with the diagonal line so I divided each dimension by two. Then I followed the same with the first oval. I noticed that the dimensions for both the rectangle and oval were close to 1:1. For the next pair I only wanted to scale the shape down and align it with the diagonal so I changed the last two dimension values.

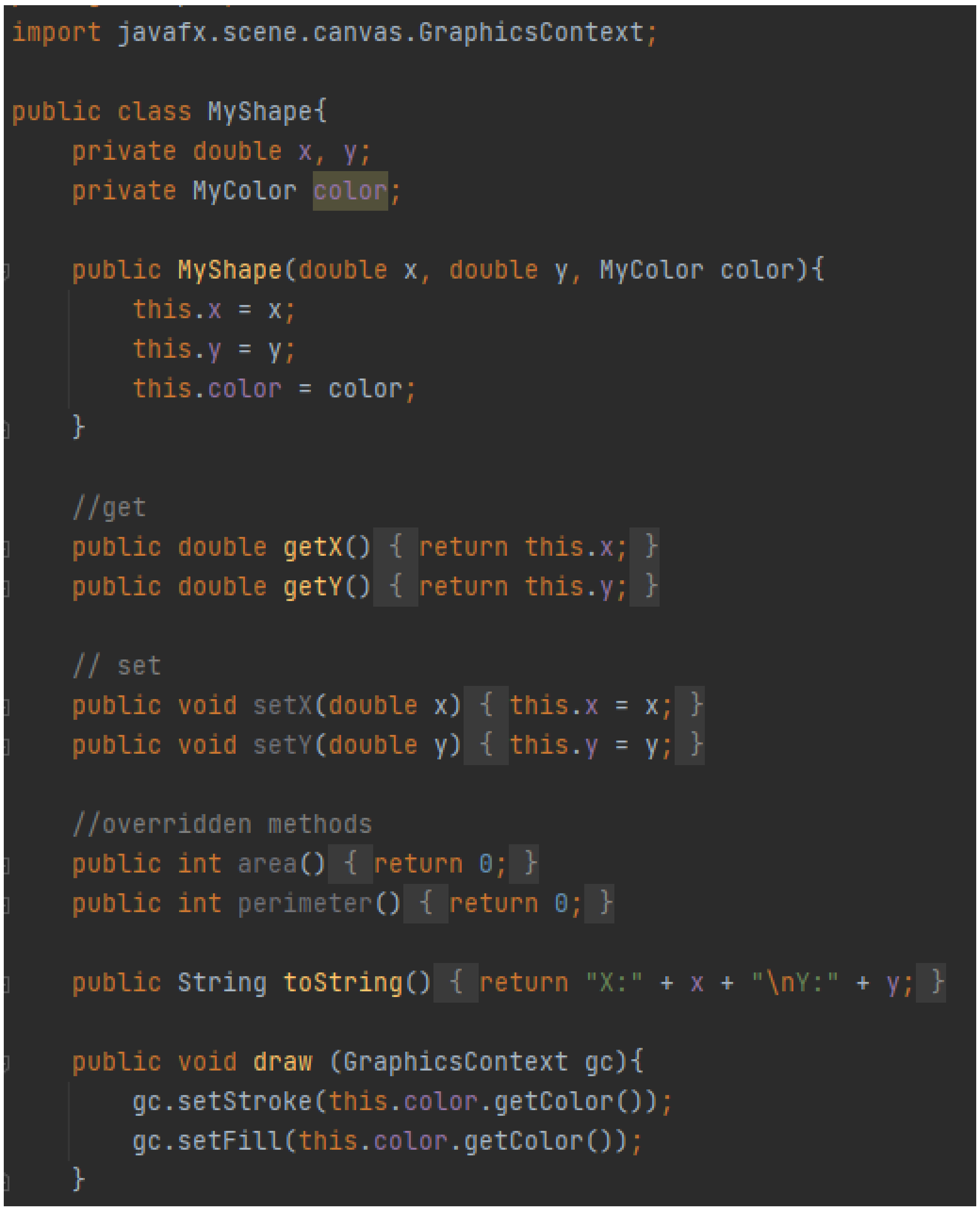


```
//show shape
root.getChildren().add(canvas);
primaryStage.setTitle("MyShape");
primaryStage.setScene(scene);
primaryStage.show();
```

For **root.getChildren().add(canvas)** adds the drawing into the canvas. The **primaryStage.setTitle("MyShape")** sets the title and **primaryStage.setScene(scene)** sets the primary stage to scene. To end it off **primaryStage.show()**, displays the illustration.

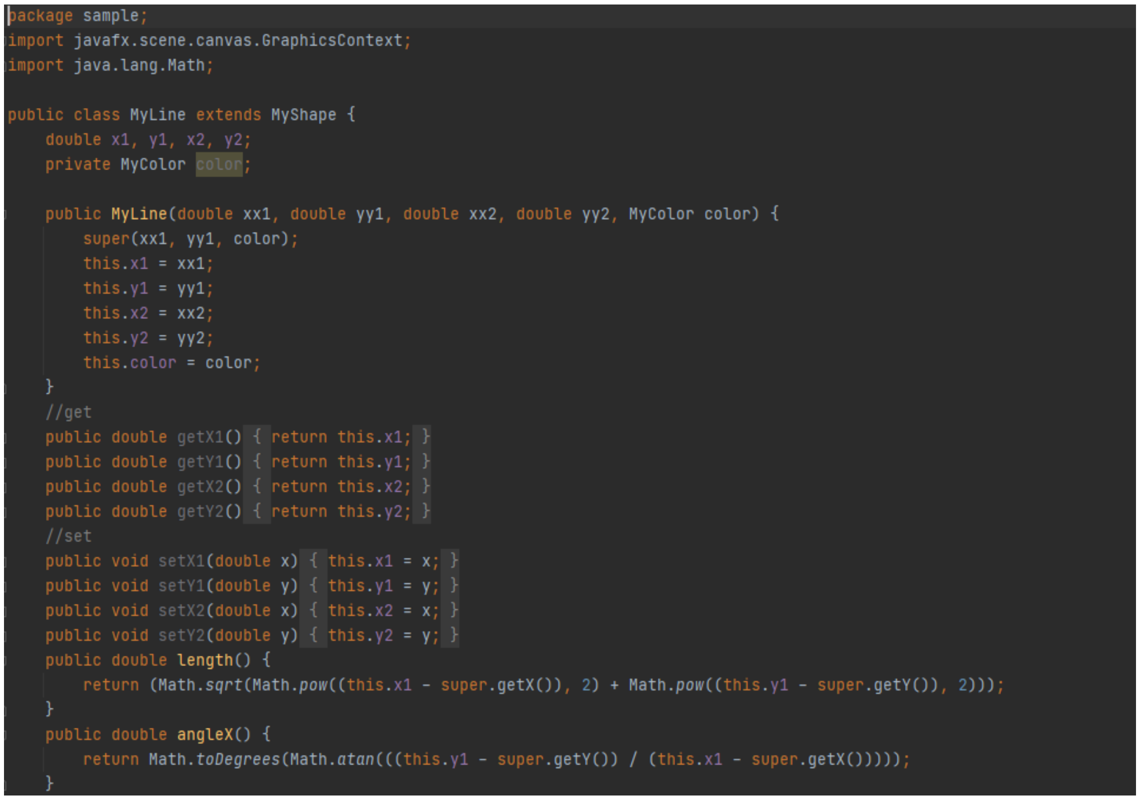
3. Code Developed

Class MyShape

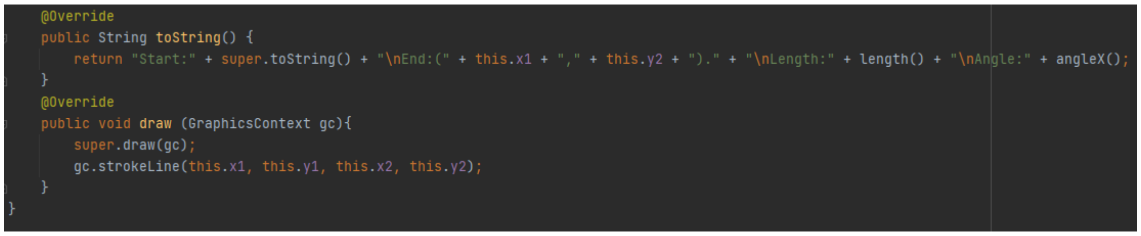


```java
import javafx.scene.canvas.GraphicsContext;

public class MyShape{
    private double x, y;
    private MyColor color;

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

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

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

    //overridden methods
    public int area() { return 0; }
    public int perimeter() { return 0; }

    public String toString() { return "X:" + x + "\nY:" + y; }

    public void draw (GraphicsContext gc){
        gc.setStroke(this.color.getColor());
        gc.setFill(this.color.getColor());
    }
```

Class MyLine

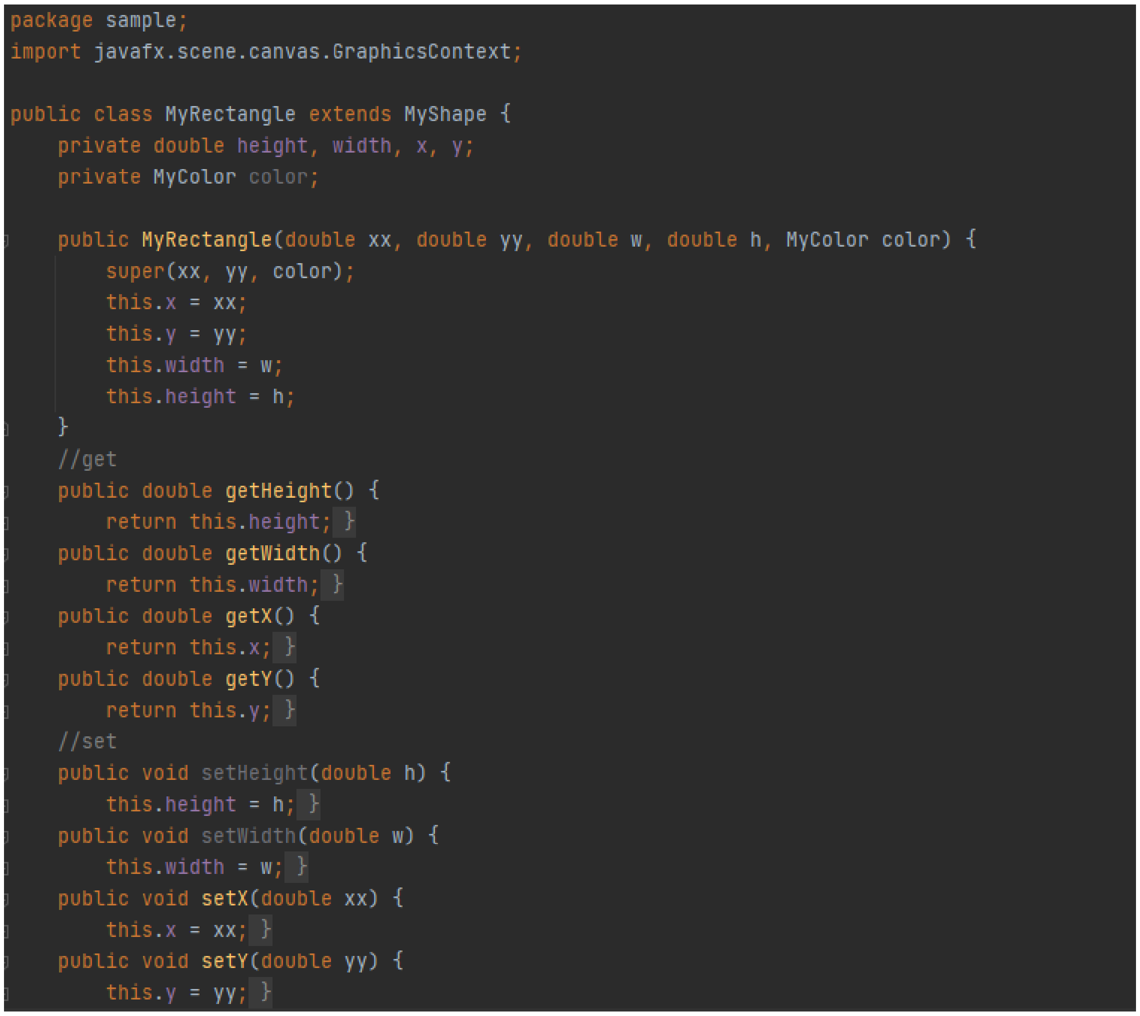


```java
package sample;
import javafx.scene.canvas.GraphicsContext;
import java.lang.Math;

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

    public MyLine(double xx1, double yy1, double xx2, double yy2, MyColor color) {
        super(xx1, yy1, color);
        this.x1 = xx1;
        this.y1 = yy1;
        this.x2 = xx2;
        this.y2 = yy2;
        this.color = color;
    }
    //get
    public double getX1() { return this.x1; }
    public double getY1() { return this.y1; }
    public double getX2() { return this.x2; }
    public double getY2() { return this.y2; }
    //set
    public void setX1(double x) { this.x1 = x; }
    public void setY1(double y) { this.y1 = y; }
    public void setX2(double x) { this.x2 = x; }
    public void setY2(double y) { this.y2 = y; }
    public double length() {
        return (Math.sqrt(Math.pow((this.x1 - super.getX()), 2) + Math.pow((this.y1 - super.getY()), 2)));
    }
    public double angleX() {
        return Math.toDegrees(Math.atan(((this.y1 - super.getY()) / (this.x1 - super.getX()))));
    }
```

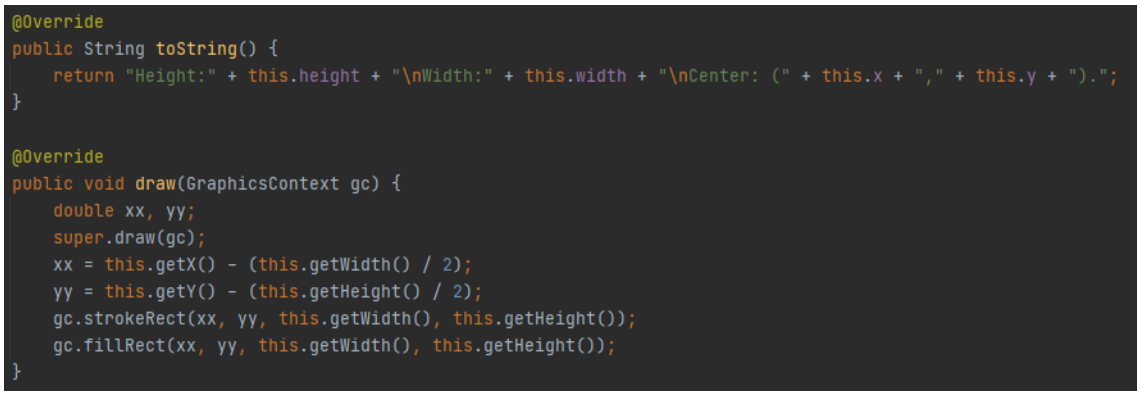


```java
    @Override
    public String toString() {
        return "Start:" + super.toString() + "\nEnd:(" + this.x1 + "," + this.y2 + ")." + "\nLength:" + length() + "\nAngle:" + angleX();
    }
    @Override
    public void draw (GraphicsContext gc){
        super.draw(gc);
        gc.strokeLine(this.x1, this.y1, this.x2, this.y2);
    }
}
```

Class MyRectangle

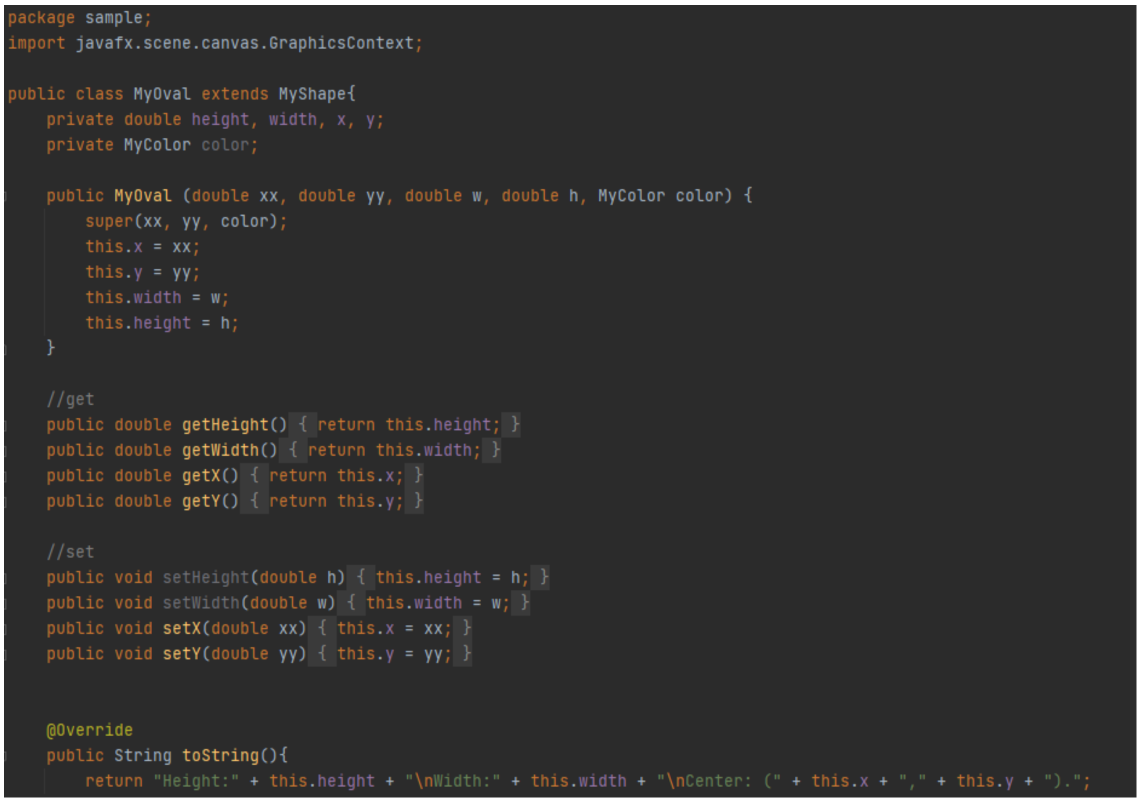


```java
package sample;
import javafx.scene.canvas.GraphicsContext;

public class MyRectangle extends MyShape {
    private double height, width, x, y;
    private MyColor color;

    public MyRectangle(double xx, double yy, double w, double h, MyColor color) {
        super(xx, yy, color);
        this.x = xx;
        this.y = yy;
        this.width = w;
        this.height = h;
    }
    //get
    public double getHeight() {
        return this.height; }
    public double getWidth() {
        return this.width; }
    public double getX() {
        return this.x; }
    public double getY() {
        return this.y; }
    //set
    public void setHeight(double h) {
        this.height = h; }
    public void setWidth(double w) {
        this.width = w; }
    public void setX(double xx) {
        this.x = xx; }
    public void setY(double yy) {
        this.y = yy; }
```

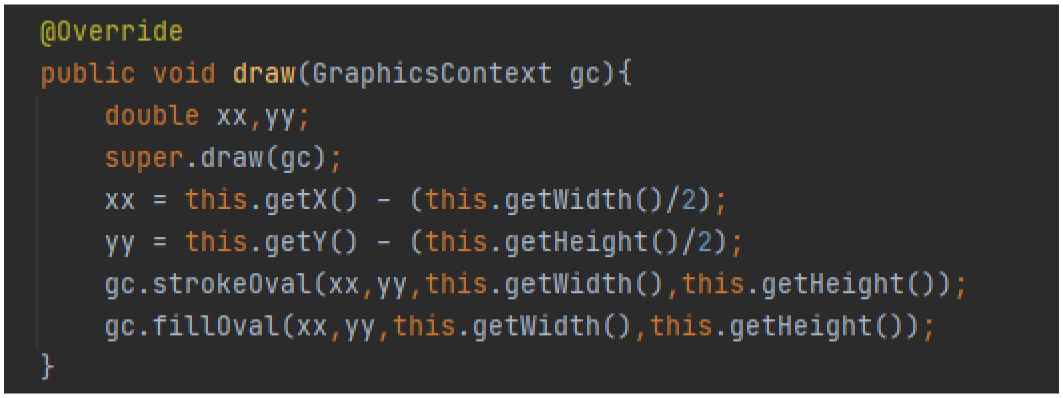


```java
@Override
public String toString() {
    return "Height:" + this.height + "\nWidth:" + this.width + "\nCenter: (" + this.x + "," + this.y + ").";
}

@Override
public void draw(GraphicsContext gc) {
    double xx, yy;
    super.draw(gc);
    xx = this.getX() - (this.getWidth() / 2);
    yy = this.getY() - (this.getHeight() / 2);
    gc.strokeRect(xx, yy, this.getWidth(), this.getHeight());
    gc.fillRect(xx, yy, this.getWidth(), this.getHeight());
}
```

Class MyOval

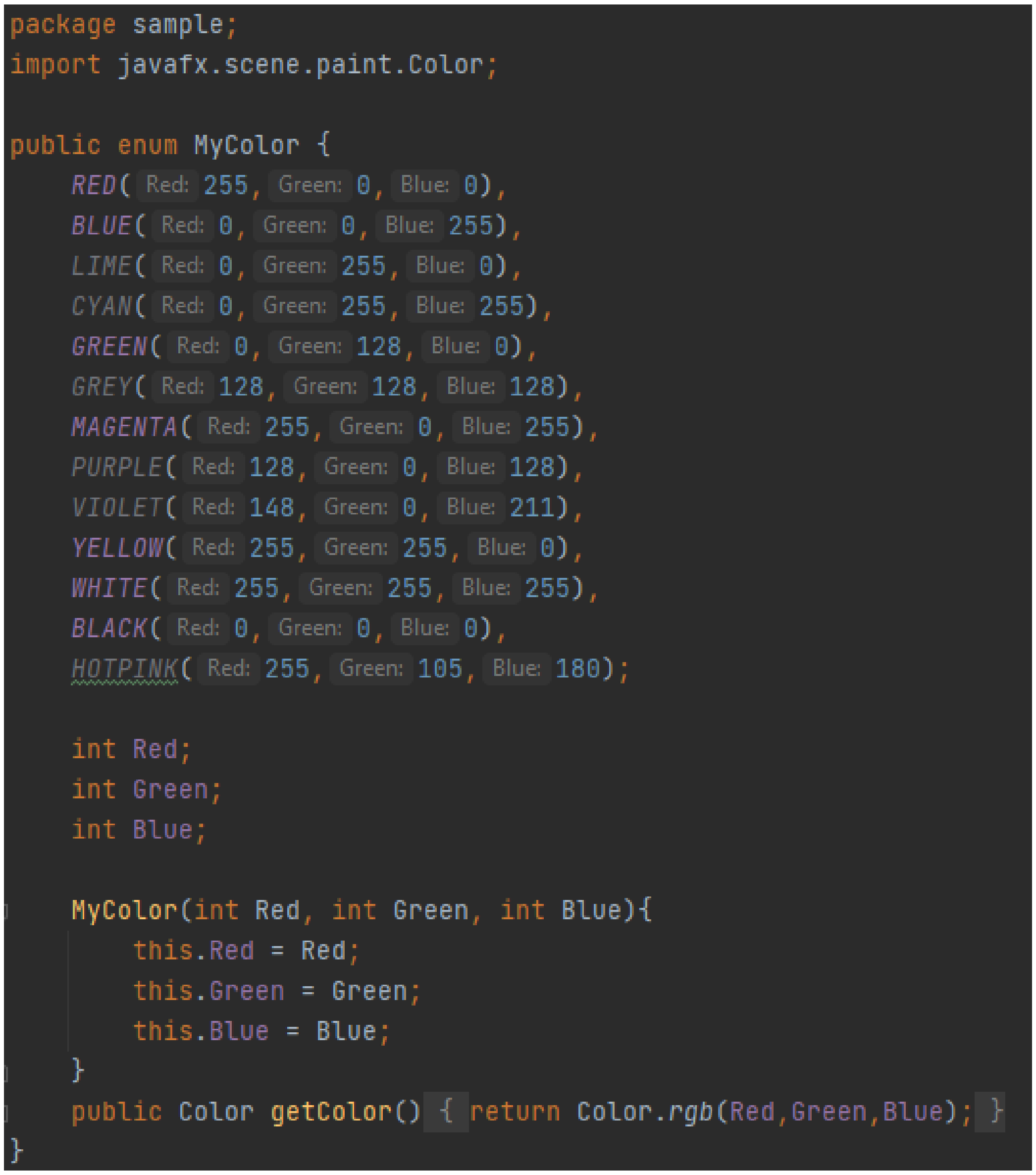


```java
package sample;
import javafx.scene.canvas.GraphicsContext;

public class MyOval extends MyShape{
    private double height, width, x, y;
    private MyColor color;

    public MyOval (double xx, double yy, double w, double h, MyColor color) {
        super(xx, yy, color);
        this.x = xx;
        this.y = yy;
        this.width = w;
        this.height = h;
    }

    //get
    public double getHeight() { return this.height; }
    public double getWidth() { return this.width; }
    public double getX() { return this.x; }
    public double getY() { return this.y; }

    //set
    public void setHeight(double h) { this.height = h; }
    public void setWidth(double w) { this.width = w; }
    public void setX(double xx) { this.x = xx; }
    public void setY(double yy) { this.y = yy; }


    @Override
    public String toString(){
        return "Height:" + this.height + "\nWidth:" + this.width + "\nCenter: (" + this.x + "," + this.y + ").";
```



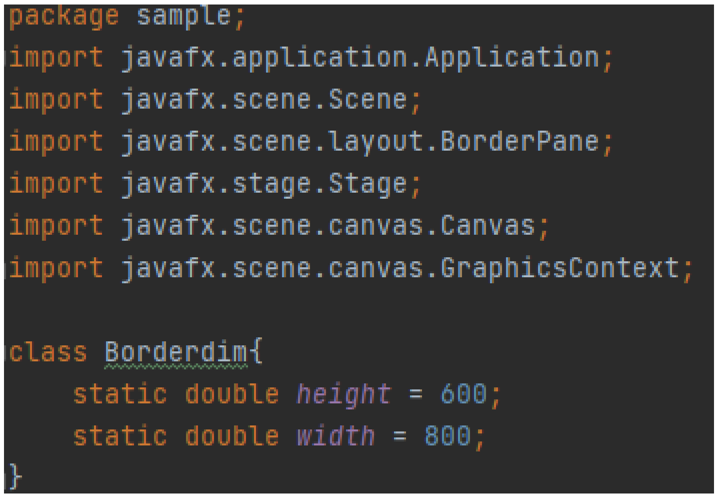
```java
@Override
public void draw(GraphicsContext gc){
    double xx,yy;
    super.draw(gc);
    xx = this.getX() - (this.getWidth()/2);
    yy = this.getY() - (this.getHeight()/2);
    gc.strokeOval(xx,yy,this.getWidth(),this.getHeight());
    gc.fillOval(xx,yy,this.getWidth(),this.getHeight());
}
```

Class MyColor

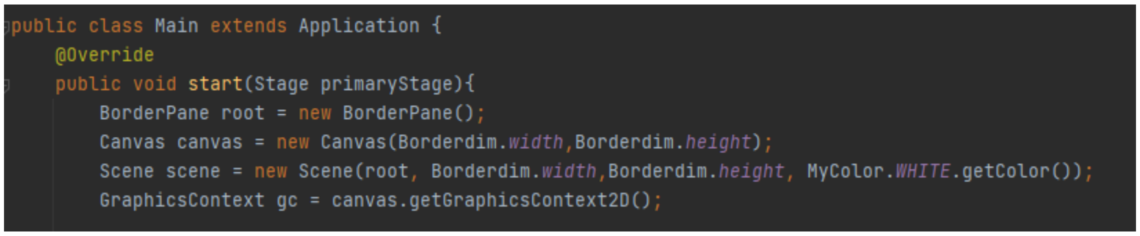


```java
package sample;
import javafx.scene.paint.Color;

public enum MyColor {
    RED( Red: 255, Green: 0, Blue: 0),
    BLUE( Red: 0, Green: 0, Blue: 255),
    LIME( Red: 0, Green: 255, Blue: 0),
    CYAN( Red: 0, Green: 255, Blue: 255),
    GREEN( Red: 0, Green: 128, Blue: 0),
    GREY( Red: 128, Green: 128, Blue: 128),
    MAGENTA( Red: 255, Green: 0, Blue: 255),
    PURPLE( Red: 128, Green: 0, Blue: 128),
    VIOLET( Red: 148, Green: 0, Blue: 211),
    YELLOW( Red: 255, Green: 255, Blue: 0),
    WHITE( Red: 255, Green: 255, Blue: 255),
    BLACK( Red: 0, Green: 0, Blue: 0),
    HOTPINK( Red: 255, Green: 105, Blue: 180);

    int Red;
    int Green;
    int Blue;

    MyColor(int Red, int Green, int Blue){
        this.Red = Red;
        this.Green = Green;
        this.Blue = Blue;
    }
    public Color getColor() { return Color.rgb(Red,Green,Blue); }
}
```

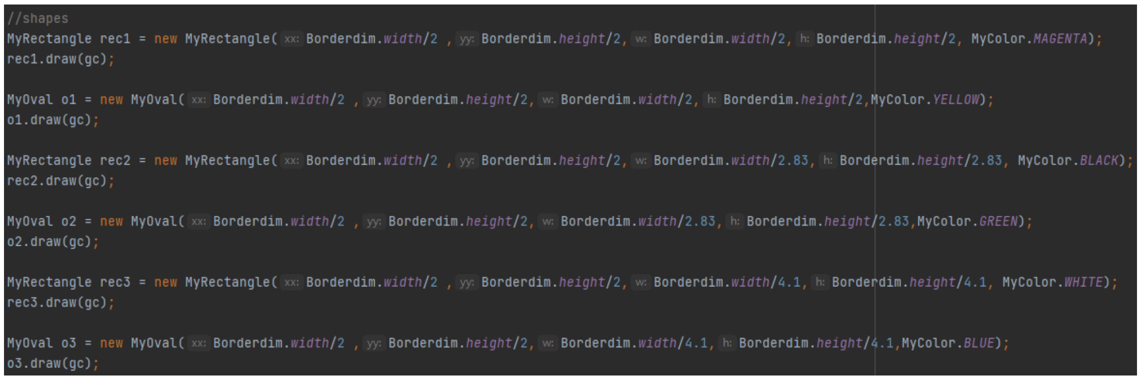
Class Main



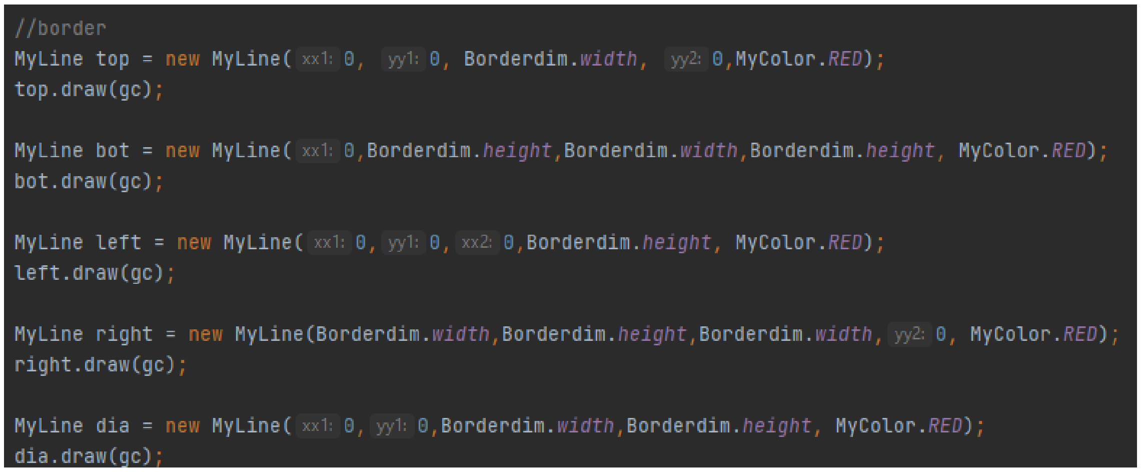
```java
package sample;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

class Borderdim{
    static double height = 600;
    static double width = 800;
}
```



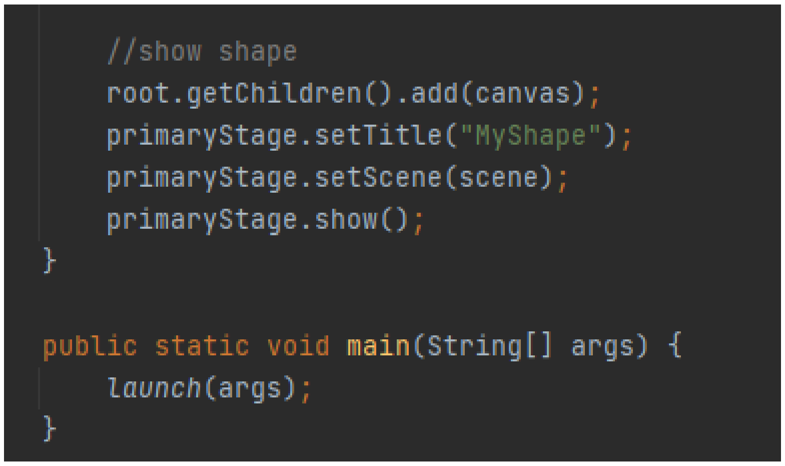
```java
public class Main extends Application {
    @Override
    public void start(Stage primaryStage){
        BorderPane root = new BorderPane();
        Canvas canvas = new Canvas(Borderdim.width,Borderdim.height);
        Scene scene = new Scene(root, Borderdim.width,Borderdim.height, MyColor.WHITE.getColor());
        GraphicsContext gc = canvas.getGraphicsContext2D();
```



```java
//shapes
MyRectangle rec1 = new MyRectangle( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2, h: Borderdim.height/2, MyColor.MAGENTA);
rec1.draw(gc);

MyOval o1 = new MyOval( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2, h: Borderdim.height/2,MyColor.YELLOW);
o1.draw(gc);

MyRectangle rec2 = new MyRectangle( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2.83, h: Borderdim.height/2.83, MyColor.BLACK);
rec2.draw(gc);

MyOval o2 = new MyOval( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/2.83, h: Borderdim.height/2.83,MyColor.GREEN);
o2.draw(gc);

MyRectangle rec3 = new MyRectangle( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/4.1, h: Borderdim.height/4.1, MyColor.WHITE);
rec3.draw(gc);

MyOval o3 = new MyOval( xx: Borderdim.width/2 , yy: Borderdim.height/2, w: Borderdim.width/4.1, h: Borderdim.height/4.1,MyColor.BLUE);
o3.draw(gc);
```



```java
//border
MyLine top = new MyLine( xx1: 0,  yy1: 0, Borderdim.width,  yy2: 0,MyColor.RED);
top.draw(gc);

MyLine bot = new MyLine( xx1: 0,Borderdim.height,Borderdim.width,Borderdim.height, MyColor.RED);
bot.draw(gc);

MyLine left = new MyLine( xx1: 0, yy1: 0, xx2: 0,Borderdim.height, MyColor.RED);
left.draw(gc);

MyLine right = new MyLine(Borderdim.width,Borderdim.height,Borderdim.width, yy2: 0, MyColor.RED);
right.draw(gc);

MyLine dia = new MyLine( xx1: 0, yy1: 0,Borderdim.width,Borderdim.height, MyColor.RED);
dia.draw(gc);
```



```java
        //show shape
        root.getChildren().add(canvas);
        primaryStage.setTitle("MyShape");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

4. Outputs Produced