SOFTWARE DESIGN LABORATORY
Assignment 2

Josh Miranda

Hesham Auda

Section P

26 March 2021

1. Problem

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

MyLine is_a MyShape;

MyRectangle is_a MyShape;

MyOval is_a MyShape;

MyCircle is_a MyOval;

MyPolygon is_a MyShape.

The objective for this assignment is similar to the first assignment in which we use the JavaFX package to create a hierarchy of Java classes to illustrate different shapes. In the previous assignment the shapes **MyLine**, **MyRectangle** and **MyOval** were created and now we must create the left over shapes **MyCircle** and **MyPolygon**. We will also be reusing the class **MyColor** which was created last assignment to color the shapes. New additions include the **MyShapeInterface** implemented by class **MyShape** and methods within. These methods are **getMyBoundingRectangle**, **pointInMyShape** and **intersectMyShapes**. Please refer to the first assignment for explanation on code that has been reused.

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. Changes that have been added from the first assignment is **Class MyShape** has been converted into an abstract class that implements the interface **MyShapeInterface**. By creating a **Class MyShape** abstract that means there may or may not be an abstract method within it. However, in our situation we do have an abstract method that draws the shapes. The abstract method makes sure that each shape is being drawn.

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

At the top here we will be importing from the JavaFX package **javafx.scene.canvas.GraphicsContext;** which will draw our shapes.

```
abstract class MyShape implements MyShapeInterface{
    private MyPoint point;
    private MyColor color;
```

In the previous assignment we individually assigned **x** and **y** for our points but, since we are making a new class called **MyPoint** it will take in **x** and **y** together.

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

Since **Class MyShape** is a super class the parameters in the constructor **double x**, **double y** and **MyColor color** will be used for the other subclasses. **MyPoint(x, y)** will be used for the top left corner of the shapes that will be drawn from the subclasses.

```
//get
public double getX() { return this.point.getXpoint(); }
public double getY() { return this.point.getYpoint(); }

// set
public void setX(double x) { this.point.setXpoint(x); }
public void setY(double y) { this.point.setYpoint(y); }
```

Similar to our previous assignment we also have the get and set methods which get and set the **(x, y)** points for our shapes from **Class MyPoint**. However, this time we are returning to this.point for both **x** and **y**.

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

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

For **toString()** and **draw()** please refer to the first assignment for explanation.

Class MyPoint

As stated before in **Class MyShape**, the **Class MyPoint** is a reference point for the top left of the shape using coordinates **(x, y)**.

```
public class MyPoint {
    private double x,y;

    MyPoint(double x, double y) {
        this.x = x;
        this.y = y;
```

Values **x** and **y** are declared as doubles and are inserted within the constructors parameters. Underneath **x** is assigned to **this.x;** and **y** is assigned to **this.y;**.

```
//get
public double getXpoint() { return this.x; }
public double getYpoint() { return this.y; }

// set
public void setXpoint(double x) { this.x = x; }
public void setYpoint(double y) { this.y = y; }
```

Lastly we have the get and set methods which get and set the points for **x** and **y** similarly in **Class MyShape** and in the previous assignment.

Class MyOval

Unsatisfied with my solution for **Class MyOval** in the previous assignment I rewrote it since later on we will need to extend **MyCircle** to **MyOval**.

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

To prevent repetition in this report, for the most of the shape classes I will be using these packages. The JavaFX package **javafx.scene.canvas.GraphicsContext** will be used for drawing the shapes and **java.lang.Math** will be used for mathematical calculations such as formulas for shapes.

```
public class MyOval extends MyShape{
    public double x, y, height, width, xCenter, yCenter, majorAxis, minorAxis, maxX, maxY, minX, minY;
    private MyColor color;
```

The **Class MyOval** will be extended to **MyShape** since **MyShape** is the top of the class hierarchy enabling it to inherit from it. Underneath is where I will be declaring my variables as double for **x**, **y**, **height**, **width**, **xCenter**, **yCenter**, **majorAxis**, **minorAxis**, **maxX**, **maxY**, **minX** and **minY**. Also as always we have **private MyColor color** which will be used to color our shapes and is shown in most of the other subclasses.

```
public MyOval (double x, double y, double w, double h, MyColor color) {
    super(x, y, color);
    this.width = w;
    this.height = h;
    this.color = color;
    setCenter(width, height);
    setAxes();
    this.maxX = this.getX() + width;
    this.maxY = this.getY() + height;
    this.minX = this.getX();
    this.minY = this.getY();
}
```

Here is the constructor **MyOval** which will take in the parameters **double x**, **double y**, **double w**, **double h** and **MyColor color**. These are the values we will be using for calculating our oval. Below we **super x**, **y** and **color** which will be inheriting from class **MyShape**. Next we use the class attribute this to set our variables so the values are for the current class. For example assigning w to this.width so that it will not confuse itself and take values from other classes. Also within the constructor we have **setCenter();** which takes in

width and **height** and we have **setAxes();**, this helps us later on for our oval. The last part is setting the **max** and **min** values for our shape. This will be further explained when talking about **MyShapeInterface**.

```
public double getArea(){
    return (Math.PI * width/2 * height/2);
}
public double getPerimeter(){
    return (2 * Math.PI * Math.sqrt((Math.pow(width/2, 2) + Math.pow(height/2, 2)) / 2));
}
```

The **getArea()** method returns the area of the oval using the formula $A = \pi * a * b$. Where **a** is half of the width and **b** is half of the height. The **getPerimeter()** method returns the perimeter of the oval using the formula $P = 2\pi\sqrt{\frac{a^2+b^2}{2}}$, same logic is used for **a** and **b** as previously stated.

```
public double getXCenter() { return xCenter; }
public double getYCenter() { return yCenter; }
public double getMyArea() { return getArea(); }

public void setAxes(){
    this.majorAxis = width;
    this.minorAxis = height;
}
private void setCenter(double width, double height) {
    this.xCenter = (width/2) + super.getX();
    this.yCenter = (height/2) + super.getY();
}
```

Here are the get and set methods that will get the values from the variables we initially stated in the beginning. **getXCenter** and **getYCenter** will be returning the values of **xCenter** and **yCenter**. The same will be done for **getMyArea()** returning the value of **getArea()**. Next are the setters with the first one being **setAxes()**. This will set the axis of the oval for easier use, **width** will be assigned to **this.majorAxis** and **height** will be assigned to t**his.minorAxis**. **setCenter** takes in parameters **double width** and **double height**. This will be used to find the **(x, y)** center coordinates of the oval. We will be assigning the calculation **(width/2) + super.getX()** to **this.xCenter** and **(height/2) + super.getY()** to **this.yCenter**.

```
@Override
public String toString(){
    return "Perimeter:" + getPerimeter() + "\nArea:" + getArea() + "\nX:" + getXCenter() + ", Y:" + getYCenter() +
            "\nMajor Axis: " + this.majorAxis + "\nMinor Axis: " + this.minorAxis;
}
```

toString will just return the string values of the variables for perimeter, area, center coordinates, major axis and minor axis.

```
@Override
public boolean pointInMyShape(MyPoint p) {
                                                                    //maxX                                                    //maxY
    return p.getXpoint() >= getX() && p.getXpoint() <= getX() + width && p.getYpoint() >= getY() && p.getYpoint() <= getY() + height;
}
@Override
public MyRectangle getMyBoundingRectangle() {
    return new MyRectangle(getX(), getY(),this.majorAxis, this.minorAxis,this.color);
}
```

pointInMyShape() and **getMyBoundingRectangle()** are two methods that are being called from **MyShapeInterface**. **pointInMyShape** returns if coordinates of the shape and if the shape overlaps with another shape coordinates it will return a boolean expression. This will

be explained further when talking about **MyShapeInterface**. **getboundingRectangle()** creates a rectangular border around the shape, for this class it will create a box around the oval. Returned is a **new MyRectangle** taking in parameters which are the dimensions of the oval. Remember the **width** is set to the **majorAxis** and **height** is set to the **minorAxis**, it is just to make it easier to read in terms of the oval shape.

```
@Override
public void draw(GraphicsContext gc){
    gc.setFill(this.color.getColor());
    gc.strokeOval(getX(), getY(), width, height);
    gc.fillOval(getX(), getY(), width, height);
    gc.strokeRect(getX(), getY(), width, height);

}
```

The draw function will be the same for all other subclasses, it will draw the shape and the border which is **gc.strokeRect()** with the parameters being the dimensions of the oval. To refrain from being repetitive on the draw function please refer back to the first assignment for an explanation.

Class MyCircle

```
public class MyCircle extends MyOval{
    public double radius, maxX, maxY, minX, minY;
    private MyColor color;
```

As stated in the problem, **class MyCircle** will be a subclass of **MyOval**. Below are the variables we will be working with mainly using the **radius**. The **maxX**, **maxY**, **minX**, **minY** are seen throughout all the subclasses as a way to see intersecting shapes and will be explained in **MyShapeInterface**.

```
public MyCircle (double x, double y, double radius, MyColor color){
    super(x,y, radius, radius, color);
    this.radius = radius;
    this.color = color;
}
```

The constructor **MyCircle** will be taking parameters **double x**, **double y**, **double radius** and **MyColor color**. Then we will take the super the the **x**, **y** and **color** since they will be taken from the superclass **MyOval**. Since **MyCircle** is extended from **MyOval** that is why there are four parameters entered to mirror **MyOval**. Lastly in the constructor we set the **radius** to **this.radius** and **color** to **this.color** to use the values in the current class.

```
public double getArea() { return (Math.PI*Math.pow(this.radius, 2)); }
public double getPerimeter() { return (2*Math.pow(this.radius,2)); }
public double getRadius() { return this.radius; }
```

The **getArea()** is returning the area of the circle with the formula $A = \pi * r^2$. The **getPerimeter()** is return the perimeter of the circle with the formula $P = 2 * r^2$. Lastly **getRadius()** returns the radius.

```java
@Override
public String toString() {
    return "Center:" + super.toString() + "\nArea:(" + getArea() + "\nPerimeter:" + getPerimeter() + "\nRadius:" + getRadius();
}

@Override
public void draw(GraphicsContext gc) {
    gc.setFill(this.color.getColor());
    gc.strokeOval(getX(), getY(), v2: getRadius()*2, v3: getRadius()*2);
    gc.fillOval(getX(), getY(), v2: getRadius()*2, v3: getRadius()*2);
    gc.strokeRect(getX(), getY(), v2: getRadius()*2, v3: getRadius()*2);
}
```

```java
@Override
public MyRectangle getMyBoundingRectangle() {
    return new MyRectangle(this.getX(), this.getY(), w: this.radius * 2, h: this.radius * 2, this.color);
}

@Override
public double getMyArea() { return 0; }

@Override
public boolean pointInMyShape(MyPoint p) {
                                                                    //maxX                                                  //maxY
    return p.getXpoint() >= getX() && p.getXpoint() <= getX() + radius && p.getYpoint() >= getY() && p.getYpoint() <= getY() + radius;
}
```

The following have been previously explained in the report or explained in the first assignment so please refer to them: **toString()**, **draw()**, **getBoundingRectangle()**, **getMyArea()** and **pointMyShape()**.

Class MyRectangle

While looking back at my previous assignment I was not only unhappy with **MyOval** but also **MyRectangle**.

```java
public class MyRectangle extends MyShape {
    public double height, width, maxX, maxY, minX, minY;
    private MyColor color;
```

MyRectangle will be extending to **MyShape** since it is a subclass and will inherit its features. Similar to **MyOval** we will be using the variables **double height** and **double width**. Also for convenience for the **MyShapeInterface** we have the max and min for the (x, y) coordinates. Lastly the **MyColor color** to color our shapes.

```
public MyRectangle(double x, double y, double w, double h, MyColor color) {
    super(x, y, color);
    this.width = w;
    this.height = h;
    this.color = color;
    this.maxX = this.getX() + width;
    this.maxY = this.getY() + height;
    this.minX = this.getX();
    this.minY = this.getY() ;
}
```

This is the **MyRectangle** constructor taking in parameters **double x**, **double y**, **double w**, **double h** and **MyColor color**. We are calling **super** for **x**, **y** and **color** since we will be inheriting it from the superclass. For **width**, **height** and **color** we use **this** attribute like in the previous subclasses so we guarantee we are using the values in the current class. Below that is the **max** and **min** for **(x, y)** coordinates which will be explained in the **MyShapeInterface**.

```
public double getWidth() { return this.width; }
public double getHeight() { return this.height; }
public double getPerimeter() { return (2*width + 2*height); }
public double getArea() { return (getWidth() * getHeight()); }
public double getMyArea() { return getArea(); }
```

For **getWidth()** it will return **width** and **getHeight()** will return **height**. **getPerimeter()** will return the **perimeter** with the formula P = 2 * width + 2 * height. **getArea()** returns the **area** by multiplying the **width** by the **height**. Lastly **getMyArea()** will return the **area**.

```
@Override
public boolean pointInMyShape(MyPoint p) {

    return p.getXpoint() >= getX() && p.getXpoint() <= getX() + width && p.getYpoint() >= getY() && p.getYpoint() <= getY() + height;
}

public MyRectangle getMyBoundingRectangle() {
    return new MyRectangle(this.getX(), this.getY(), this.width, this.height, this.color);
}
```

```
@Override
public String toString() {
    return "Height:" + getHeight() + "\nWidth:" + getWidth() + "\nPerimeter:" + getPerimeter() + "\nArea:" + getArea();
}

@Override
public void draw(GraphicsContext gc) {
    gc.setFill(this.color.getColor());
    gc.strokeRect(getX(), getY(), getWidth(), getHeight());
    gc.fillRect(getX(), getY(), getWidth(), getHeight());
    gc.strokeRect(getX(), getY(), getWidth(), getHeight());

}
```

The following have been previously explained in the report or explained in the first assignment so please refer to them: **toString()**, **draw()**, **getBoundingRectangle()** and **pointMyShape()**.

Class MyPolygon

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

Since we will be working with arrays an additional package that will be imported is java.utilArrays. This will help organize arrays and allow us to find the min and max values with ease.

```
public class MyPolygon extends MyShape {
    public int r, N, maxX, maxY, minX, minY;
    private MyColor color;
```

Like most of the other shapes we will be inheriting from the superclass **MyShape**. Declared as int is **r** for the **radius**, **N** for the sides of the polygon and the **max** and **min** values of coordinates **(x, y)**. Lastly we have **MyColor** color for coloring the shapes.

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

For the **MyPolygon** constructor we will have most of the previously declared variables excluding the **min** and **max** values of coordinates **(x, y)** for the parameters. They include **double x**, **double y**, **int r**, **int N** and **Mycolor color**. As always we take the super of **x**, **y** and **color** as we inherit it from the superclass. Then we assign **r** and **N** so that we guarantee we use the current values of the class using the this class attribute.

```
public double getArea() { return (int) (this.N * (Math.pow(this.r, 2)) * Math.tan((Math.PI / this.N))); }
public double getPerimeter() { return (int) (this.N * getSide()); }
public double getAngle() { return (this.N - 2) * (180.0 / this.N); }
public double getSide() { return (2 * this.r * Math.tan(Math.PI / this.N)); }
```

To get **getArea()** we use the area formula for a polygon which is $A = N * r^2 * tan(\frac{\pi}{N})$. For **getPerimeter()** the formula is $P = N * s$, where the number of sides is multiplied by the radius of the perimeter. The **getAngle()** returns the interior angles with (N-2)*($\frac{180}{N}$). Lastly is **getSide()** which returns the radius from the amount of sides since polygons can have as many sides. The formula is $2 * r * tan(\frac{\pi}{N})$.

```
public MyRectangle getMyBoundingRectangle(){
    return new MyRectangle(this.minX, this.minY, w: this.maxX - this.minX, h: this.maxY - this.minY, this.color);
}

@Override
public double getMyArea() { return 0; }

@Override
public boolean pointInMyShape(MyPoint p) {
    return p.getXpoint() >= getX() && p.getXpoint() <= maxX && p.getYpoint() >= getY() && p.getYpoint() <= maxY;
}

@Override
public String toString() {
    return "Center:" + super.toString() + "\nArea:" + getArea() + "\nPerimeter:" + getPerimeter() + "\nAngle:" + getAngle() + "\nSides:" + getSide();
}
```

The following have been previously explained in the report or explained in the first assignment so please refer to them: **getBoundingRectangle(), getMyArea(), pointMyShape()** and **toString()**.

```
@Override
public void draw(GraphicsContext gc) {

    double[] x_cords = new double[this.N];
    double[] y_cords = new double[this.N];
```

What sets this apart from other subclasses is that it takes more to draw the polygon. First we start entering the sides into an array for the x and y coordinates.

```
// find the vertices
for (int j = 0; j < this.N; j++) {
    x_cords[j] = getX() + this.r * Math.sin(2 * Math.PI * j / this.N);
    y_cords[j] = getY() + this.r * Math.cos(2 * Math.PI * j / this.N);
}
```

Then we use a for loop to find the vertices of the polygon through the formulas for the x coordinates **getX() + this.r * Math.sin(2 * Math.PI * j / this.N)** and y coordinates **getY() + this.r * Math.cos(2 * Math.PI * j / this.N)**.

```
//convert double to int
final int[] ix_cords = new int[x_cords.length];
for (int i = 0; i<x_cords.length; ++i)
    ix_cords[i] = (int) x_cords[i];
```

Since in the previous step we have an array of type doubles we will need to convert it to int in order for us to find the **min** and **max** values of coordinates **(x, y)**.

```
//find min max x values
this.minX = Arrays.stream(ix_cords).min().getAsInt();
this.maxX = Arrays.stream(ix_cords).max().getAsInt();
```

This is where the Arrays package comes in so that finding the **min** and **max** values become easier.

```
//convert double to int
final int[] iy_cords = new int[y_cords.length];
for (int i = 0; i<y_cords.length; ++i)
    iy_cords[i] = (int) y_cords[i];

//find min max y values
this.minY = Arrays.stream(iy_cords).min().getAsInt();
this.maxY = Arrays.stream(iy_cords).max().getAsInt();
```

All the steps done previously for the x coordinates will be done the same for the y coordinates.

```
gc.setLineWidth(2);
gc.setFill(color.getColor());
gc.setStroke(MyColor.BLACK.getColor());
gc.strokePolygon(x_cords, y_cords, this.N);
gc.fillPolygon(x_cords, y_cords, this.N);
gc.strokeRect(this.minX, this.minY, v2: this.maxX - this.minX, v3: this.maxY - this.minY);
```

The reason why we have to find the min and max values for coordinates **(x, y)** is because unlike the other shapes rectangle and circle, the polygon is more complex as the number of

sides can change from the user inputs. This means that the **height** and **width** must be calculated to draw the border for the polygon.

Class MyShapeInterface

As previously stated **Class MyShape** implements from **Class MyShapeInterface**. **Class MyShapeInterface** is where methods **getMyBoundingRectangle()**, **pointInMyShape()** and **getMyArea()** are stored.

```
public interface MyShapeInterface {
    MyRectangle getMyBoundingRectangle();
    public double getMyArea();
    abstract boolean pointInMyShape(MyPoint p);
```

The **getMyBoundingRectangle()** method creates a border around the selected shape by starting its dimensions at the top left corner with **getX()** and **getY()** then expands out with the selected **height** and **width**. For example the top of the rectangle will be from **getX()** to **width** since the **width** will be the **max** length of the rectangle. Likewise can be said with the vertical dimensions **getY()** to **height**. **getMyArea()** simply just returns the **area** of the shapes through **(x, y)** coordinates. Next we have a boolean **pointInMyShape** that takes in the parameter **MyPoint p** from **Class MyPoint**. What this method achieves is that it will detect if a point is within the bounds of shape.

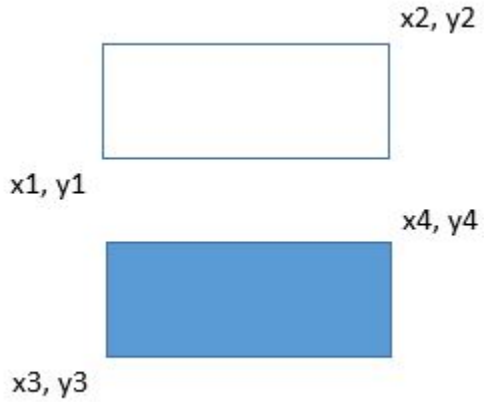
```
@Override
public boolean pointInMyShape(MyPoint p) {
                                                    //maxX                                  //maxY
    return p.getXpoint() >= getX() && p.getXpoint() <= getX() + width && p.getYpoint() >= getY() && p.getYpoint() <= getY() + height;
}
```

If the point is within the bounds then it will return **True**. This is shown by a line of boolean conditions. We will be using **Class MyOval** as an example. **p.getXpoint()** and **p.getYpoint()** is the coordinates of the point. Translating the return statement in english will be as described, if the **Xpoint** is greater than or equal than the top left of the shape and **Xpoint** is less than or equal to the **max width** of the shape and **Ypoint** in greater than or equal to the top left of shape and **Ypoint** is less than or equal to the **max height** of the shape then it will return **True**.

```
static public String intersectMyShapes(MyShape one, MyShape two){
    MyRectangle r1 = one.getMyBoundingRectangle();
    MyRectangle r2 = two.getMyBoundingRectangle();

    //x1,y1 = (r1.minX, r1.maxY)
    //x2,y2 = (r1.maxX, r1.minY)
    //x3,y3 = (r2.minX, r2.maxY)
    //x4,y4 = (r2.maxX, r2.minY)
    return "Shapes one and two do not overlap " + ( r2.minX > r1.maxX || r2.maxY < r1.minY || r1.minX > r2.maxX || r1.maxY < r2.minY );
}
```

intersectMyShape() is a static method that takes in parameters **MyShape one** and **MyShape two**. This shows if two shapes are intersecting and returns **True** if they do not overlap. This is done by a line of boolean conditions in the return statement. First we create two rectangles for shape one and two that simulate the border of the shapes. An easy way to determine if two shapes intersect is by looking at the diagonal coordinates.



This is seen in the comments made above, **x1** is the **minx** value of **shape one** and **y1** is the **maxy** value of **shape one** and if you look it will show the bottom left of the rectangle. **x2** and **y2** will show the top right of the rectangle and the same can be said for **(x3, y3)** and **(x4, y4)**. If the conditions in the return statement are met then they do not overlap.

Class Main, Class MyLine, Class MyColor

For these classes they mostly stayed the same from the first assignment. The Class Main will draw the included instructions for drawing the various shapes. **Class MyLine** will have **getMyBoundingRectangle()**, **getMyArea()**, **pointInMyShape()** and **toString()** which are explained in the previous subclasses or in the previous assignment so please refer back. **Class MyColor** also stayed the same.

3. Code Developed

Class Main

```
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;
}

public class Main extends Application {

   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();


       MyPoint p1= new MyPoint(500,600);
       //shapes
       MyPolygon r1 = new MyPolygon(500,300,50,
6,MyColor.BLUE);
       r1.draw(gc);
       r1.pointInMyShape(p1);
```

```
        System.out.println("Point in Shape:" +
r1.pointInMyShape(p1));

        MyPolygon r2 = new MyPolygon(500,350,50,
6,MyColor.RED);
        r2.draw(gc);

        MyShapeInterface.intersectMyShapes(r1,r2);

System.out.println(MyShapeInterface.intersectMyShap
es(r1,r2));

        MyCircle c1 = new
MyCircle(200,100,20,MyColor.BLUE);
        c1.draw(gc);

        MyOval o1 = new
MyOval(Borderdim.width/4,Borderdim.height/4,
Borderdim.width/4, Borderdim.height/4,
MyColor.GREEN);
        o1.draw(gc);

        MyRectangle r = new
MyRectangle(Borderdim.width/9
,Borderdim.height/6,Borderdim.width/9,Borderdim.hei
ght/6, MyColor.YELLOW);
        r.draw(gc);


        //border
        MyLine top = new MyLine(0, 0,
Borderdim.width, 0,MyColor.RED);
        top.draw(gc);

        MyLine bot = new
MyLine(0,Borderdim.height,Borderdim.width,Borderdim
.height, MyColor.RED);
```

```
        bot.draw(gc);

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

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

        //MyLine dia = new
MyLine(0,0,Borderdim.width,Borderdim.height,
MyColor.RED);
        //dia.draw(gc);


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

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

Class MyCircle

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

public class MyCircle extends MyOval{
   public double radius, maxX, maxY, minX, minY;
   private MyColor color;

   public MyCircle (double x, double y, double
radius, MyColor color){
       super(x,y, radius, radius, color);
       this.radius = radius;
       this.color = color;
   }

   public double getArea(){
       return (Math.PI*Math.pow(this.radius, 2));
   }
   public double getPerimeter(){
       return (2*Math.pow(this.radius,2));
   }
   public double getRadius(){
       return this.radius;
   }

   @Override
   public String toString() {
       return "Center:" + super.toString() +
"\nArea:(" + getArea() + "\nPerimeter:" +
getPerimeter() + "\nRadius:" + getRadius();
   }

   @Override
   public void draw(GraphicsContext gc) {
```

```
        gc.setFill(this.color.getColor());
        gc.strokeOval(getX(), getY(), getRadius()*2,
getRadius()*2);
        gc.fillOval(getX(), getY(), getRadius()*2,
getRadius()*2);
        gc.strokeRect(getX(), getY(), getRadius()*2,
getRadius()*2);
    }

    @Override
    public MyRectangle getMyBoundingRectangle() {
        return new MyRectangle(this.getX(),
this.getY(), this.radius * 2, this.radius * 2,
this.color);
    }

    @Override
    public double getMyArea() {
        return 0;
    }

    @Override
    public boolean pointInMyShape(MyPoint p) {

//maxX
//maxY
        return p.getXpoint() >= getX() &&
p.getXpoint() <= getX() + radius && p.getYpoint()
>= getY() && p.getYpoint() <= getY() + radius;
    }
}
```

Class MyColor

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

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

    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 MyLine

```
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()))));
    }

    public MyRectangle getMyBoundingRectangle() {
        return new MyRectangle(this.x1, this.y1,
this.x2 - this.x1, this.y2 - this.y1, this.color);
    }

    @Override
    public double getMyArea() {
        return 0;
    }

    @Override
    public boolean pointInMyShape(MyPoint p) {
        return false;
    }


    @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);
        gc.strokeRect(this.x1, this.y1, this.x2 -
this.x1, this.y2 - this.y1);
    }
}
```

Class MyOval

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

public class MyOval extends MyShape{
   public double x, y, height, width, xCenter,
yCenter, majorAxis, minorAxis, maxX, maxY, minX,
minY;
   private MyColor color;

   public MyOval (double x, double y, double w,
double h, MyColor color) {
       super(x, y, color);
       this.width = w;
       this.height = h;
       this.color = color;
       setCenter(width, height);
       setAxes();
       this.maxX = this.getX() + width;
       this.maxY = this.getY() + height;
       this.minX = this.getX();
       this.minY = this.getY();
   }

   public double getArea(){
       return (Math.PI * width/2 * height/2);
   }
   public double getPerimeter(){
       return (2 * Math.PI *
Math.sqrt((Math.pow(width/2, 2) +
Math.pow(height/2, 2)) / 2));
   }
   public double getXCenter(){
       return xCenter;
```

```
    }
    public double getYCenter() {
        return yCenter;
    }
    public double getMyArea() {
        return getArea();
     }

    public void setAxes(){
        this.majorAxis = width;
        this.minorAxis = height;
    }
    private void setCenter(double width, double
height) {
        this.xCenter = (width/2) + super.getX();
        this.yCenter = (height/2) + super.getY();
    }

    @Override
    public String toString(){
        return "Perimeter:" + getPerimeter() +
"\nArea:" + getArea() + "\nX:" + getXCenter() + ",
Y:" + getYCenter() +
                "\nMajor Axis: " + this.majorAxis +
"\nMinor Axis: " + this.minorAxis;
    }
    @Override
    public boolean pointInMyShape(MyPoint p) {

//maxX
//maxY
        return p.getXpoint() >= getX() &&
p.getXpoint() <= getX() + width && p.getYpoint() >=
getY() && p.getYpoint() <= getY() + height;
    }
    @Override
    public MyRectangle getMyBoundingRectangle() {
```

```
        return new MyRectangle(getX(),
getY(),this.majorAxis, this.minorAxis,this.color);
    }

    @Override
    public void draw(GraphicsContext gc){
        gc.setFill(this.color.getColor());
        gc.strokeOval(getX(), getY(), width,
height);
        gc.fillOval(getX(), getY(), width, height);
        gc.strokeRect(getX(), getY(), width,
height);

    }


}
```

Class MyPoint

```java
package sample;


public class MyPoint {
    private double x,y;

    MyPoint(double x, double y) {
        this.x = x;
        this.y = y;
    }
    //get
    public double getXpoint(){
        return this.x;
    }
    public double getYpoint(){
        return this.y;
    }

    // set
    public void setXpoint(double x){
        this.x = x;
    }
    public void setYpoint(double y){
        this.y = y;
    }

}
```

Class MyPolygon

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



public class MyPolygon extends MyShape {
    public int r, N, maxX, maxY, minX, minY;
    private MyColor color;

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

    public double getArea() {
        return (int) (this.N * (Math.pow(this.r, 2))
* Math.tan((Math.PI / this.N)));
    }
    public double getPerimeter() {
        return (int) (this.N * getSide());
    }
    public double getAngle() {
        return (this.N - 2) * (180.0 / this.N);
    }
    public double getSide() {
        return (2 * this.r * Math.tan(Math.PI /
this.N));
    }
```

```
    public MyRectangle getMyBoundingRectangle(){
        return new MyRectangle(this.minX, this.minY,
this.maxX - this.minX, this.maxY - this.minY,
this.color);
    }

    @Override
    public double getMyArea() {
        return 0;
    }

    @Override
    public boolean pointInMyShape(MyPoint p) {
        return p.getXpoint() >= getX() &&
p.getXpoint() <= maxX && p.getYpoint() >= getY() &&
p.getYpoint() <= maxY;
    }

    @Override
    public String toString() {
        return "Center:" + super.toString() +
"\nArea:" + getArea() + "\nPerimeter:" +
getPerimeter() + "\nAngle:" + getAngle() +
"\nSides:" + getSide();
    }

    @Override
    public void draw(GraphicsContext gc) {

        double[] x_cords = new double[this.N];
        double[] y_cords = new double[this.N];

        // find the vertices
        for (int j = 0; j < this.N; j++) {
            x_cords[j] = getX() + this.r *
Math.sin(2 * Math.PI * j / this.N);
            y_cords[j] = getY() + this.r *
```

```
Math.cos(2 * Math.PI * j / this.N);
        }

        //convert double to int
        final int[] ix_cords = new
int[x_cords.length];
        for (int i = 0; i<x_cords.length; ++i)
            ix_cords[i] = (int) x_cords[i];

        //find min max x values
        this.minX =
Arrays.stream(ix_cords).min().getAsInt();
        this.maxX =
Arrays.stream(ix_cords).max().getAsInt();

        //convert double to int
        final int[] iy_cords = new
int[y_cords.length];
        for (int i = 0; i<y_cords.length; ++i)
            iy_cords[i] = (int) y_cords[i];

        //find min max y values
        this.minY =
Arrays.stream(iy_cords).min().getAsInt();
        this.maxY =
Arrays.stream(iy_cords).max().getAsInt();

        gc.setLineWidth(2);
        gc.setFill(color.getColor());
        gc.setStroke(MyColor.BLACK.getColor());
        gc.strokePolygon(x_cords, y_cords, this.N);
        gc.fillPolygon(x_cords, y_cords, this.N);
        gc.strokeRect(this.minX, this.minY,
this.maxX - this.minX, this.maxY - this.minY);

    }
}
```

Class MyRectangle

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


public class MyRectangle extends MyShape {
   public double height, width, maxX, maxY, minX,
minY;
   private MyColor color;

   public MyRectangle(double x, double y, double w,
double h, MyColor color) {
       super(x, y, color);
       this.width = w;
       this.height = h;
       this.color = color;
       this.maxX = this.getX() + width;
       this.maxY = this.getY() + height;
       this.minX = this.getX();
       this.minY = this.getY() ;
   }

   public double getWidth() {
       return this.width;
   }
   public double getHeight() {
       return this.height;
   }
   public double getPerimeter(){
       return (2*width + 2*height);
   }
   public double getArea(){
       return (getWidth() * getHeight());
   }
   public double getMyArea(){
```

```
        return getArea();
    }

    @Override
    public boolean pointInMyShape(MyPoint p) {
        return p.getXpoint() >= getX() &&
p.getXpoint() <= getX() + width && p.getYpoint() >=
getY() && p.getYpoint() <= getY() + height;
    }

    public MyRectangle getMyBoundingRectangle() {
        return new MyRectangle(this.getX(),
this.getY(), this.width, this.height, this.color);
    }

    @Override
    public String toString() {
        return "Height:" + getHeight() + "\nWidth:"
+ getWidth() + "\nPerimeter:" + getPerimeter() +
"\nArea:" + getArea();
    }

    @Override
    public void draw(GraphicsContext gc) {
        gc.setFill(this.color.getColor());
        gc.strokeRect(getX(), getY(), getWidth(),
getHeight());
        gc.fillRect(getX(), getY(), getWidth(),
getHeight());
        gc.strokeRect(getX(), getY(), getWidth(),
getHeight());

    }
}
```

Class MyShape

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

abstract class MyShape implements MyShapeInterface{
   private MyPoint point;
   private MyColor color;

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

   //get
   public double getX(){
        return this.point.getXpoint();
   }
   public double getY(){
        return this.point.getYpoint();
   }

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

   public String toString(){
        return "X:" + getX() + "\nY:" + getY();
   }

   public void draw (GraphicsContext gc){
```

```
        gc.setStroke(this.color.getColor());
        gc.setFill(this.color.getColor());
    }
}
```

Class MyShapeInterface

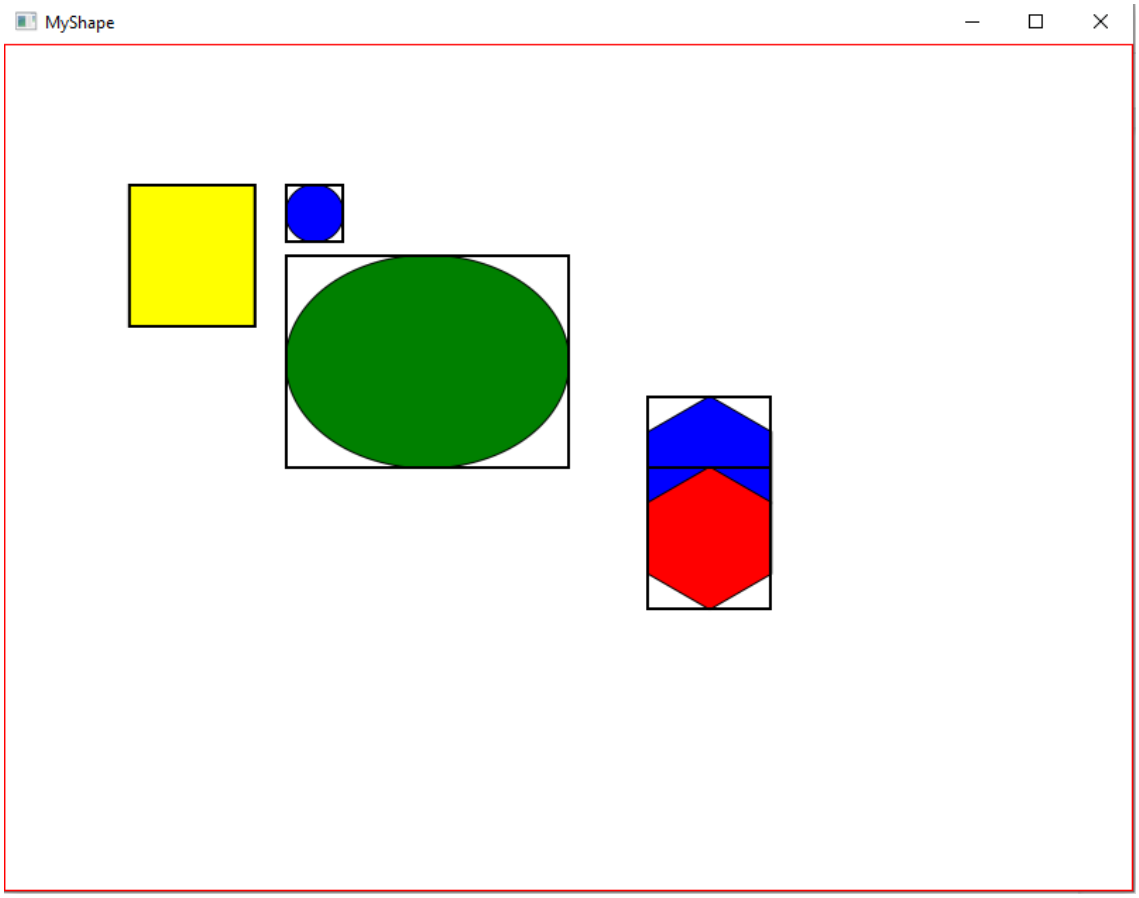
```
package sample;

public interface MyShapeInterface {
   MyRectangle getMyBoundingRectangle();
   public double getMyArea();
   abstract boolean pointInMyShape(MyPoint p);

   static public String intersectMyShapes(MyShape
one, MyShape two){
       MyRectangle r1 =
one.getMyBoundingRectangle();
       MyRectangle r2 =
two.getMyBoundingRectangle();

       //x1,y1 = (r1.minX, r1.maxY)
       //x2,y2 = (r1.maxX, r1.minY)
       //x3,y3 = (r2.minX, r2.maxY)
       //x4,y4 = (r2.maxX, r2.minY)
       return "Shapes one and two do not overlap "
+ ( r2.minX > r1.maxX || r2.maxY < r1.minY ||
r1.minX > r2.maxX || r1.maxY < r2.minY );
   }
}
```

4. Outputs Produced




```
Point in Shape:false
Shapes one and two do not overlap false
```