Report: Assignment:2

MYESHA MAHAZABEEN

CSC 22100, SOFTWARE DESIGN LABORATORY, 1XB [6822], SUMMER 2022

Instructor: Hesham A. Auda
Date: 07/08/2022

Abstract: This assignment improves assignment-1 by adding some additional shapes, and introducing some OOP concepts such as abstract class, abstract methods, and interfaces. Additionally, this report explains every aspect of assignment-2 by describing each element of the program as well as adding screenshots of the program and outputs.

Department of Computer Science

The City College of CUNY

CSc 22100 [X 6822]: Software Design Laboratory [Summer 2022]

Assignment 2

A report uploaded on the Bloackboard's course page for the section showing:

[1] *the problem,*
[2] *solution methods,*
[3] *codes developed, and*
[4] *outputs produced for the tasks indicated*

is due by 11:00 *pm on Friday,* 8 July 2022. ***The deadline is strictly observed***.

1- Create a hierarchy of Java classes as follows:

A report uploaded on the Bloackboard's course page for the section showing:

[1] *statement of the problem,*
[2] *solution methods,*
[3] *all codes developed, and*
[4] *outputs produced for the tasks indicated,*

is due by **11:59 *pm on Sunday,* 26 June 2022**. ***The deadline is strictly observed***.

2- Create a hierarchy of Java classes as follows:

MyLine *extends* **MyShape**;
MyArc *extends* **MyShape**;
MyRectangle *extends* **MyShape**;
MyOval *extends* **MyShape**;
MyCircle *extends* **MyOval**.

Class MyShape:

Class **MyShape** is an **abstract** class; is the hierarchy's superclass; and inherits Java class Object. The *area*, *perimeter*, and *draw* methods in class **MyShape** are *abstract* methods and hence must be overridden in each subclass in the hierarchy. The implementation of the class defines a reference point **p**(*x*, *y*), an object of type **MyPoint**, and the color of the shape of enum reference type **MyColor**. Otherwise, the classes **MyPoint**, **MyShape**, **MyLine**, **MyRectangle**, **and MyOval** are as defined in Assignment 1 .

Class MyArc:

Class **MyArc** inherits class MyShape. The **MyArc** object is a segment of the boundary of a **MyOval** object, defined by the endpoints **p**1 ($x1, y1$) and **p**2($x2, y2$), or their corresponding angles, measured from the x-axis, on the **MyOval** boundary. The **MyArc** object may be filled with any color of **MyColor** *enum* reference type. The class includes appropriate class constructors and methods, including methods that perform the following operations:

a. *length* — returns the arc length of the **MyArc** object;
b. *toString* — returns a string representation of a **MyArc** object;
c. *draw*— draws a **MyArc** object.

3- Interface **MyShapeInterface** is implemented by the abstract class **MyShape**. All subclasses of the hierarchy must therefore be amended in accordance with the interface. The interface includes *constants* (as needed) and appropriate *abstract*, *static*, and *default* methods that perform functions that describe specific behaviors of the object types of the class hierarchy, including:

a. *getMyBoundingRectangle* — abstract method returns the bounding rectangle of an object in the class hierarchy;
b. *pointInMyShape*— abstract method returns true if a point **p** is located within or on the boundary of an object in the class hierarchy;
c. *intersectMyShapes* — static method returns the intersection of two **MyShape** objects, **S**1 and **S**2 — i.e., the set of all points on or within the boundary of the area — of two objects in the class hierarchy if they do overlap, and **null** otherwise.
d. *drawIntersectMyShapes* — default method returns a canvas with a drawing of the intersection of two objects in the class hierarchy if they do overlap.

4- Use JavaFX graphics and the class hierarchy to draw a geometric configuration of a circular pizza pie arbitrarily sliced as illustrated below, subject to the following additional requirements:

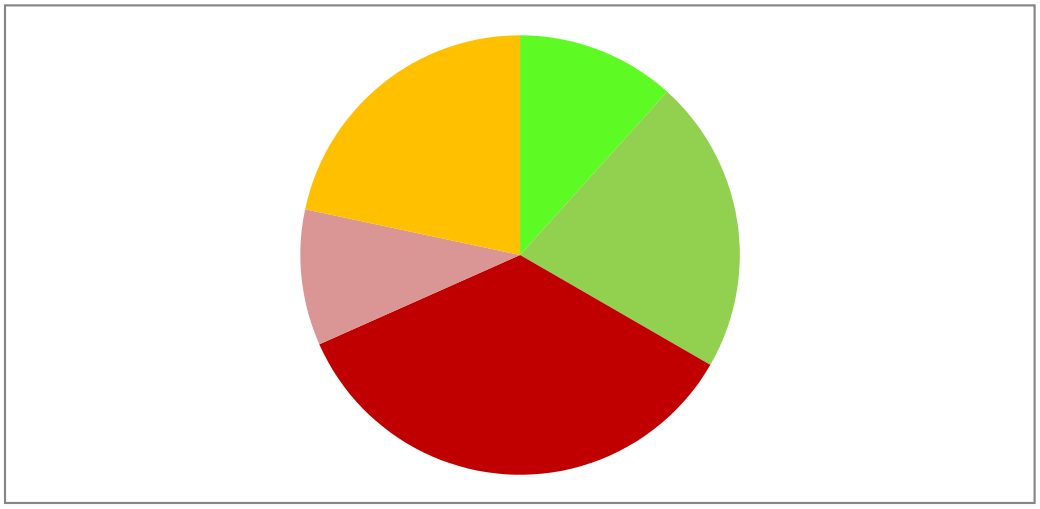
a. The code is applicable to canvases of variable height and width;
b. The dimensions of the shapes are proportional to the smallest dimension of the canvas;
c. The polygons and circles are filled with different colors of your choice, specified through the reference type **MyColor**; and
d. All objects are processed polymorphically.

Further:

- Draw the bounding rectangle of **MyLine**, **MyArc**, and **MyCircle** objects of your choice;
- Draw the area of intersection of:

a. Two **MyRectangle** objects; and
b. A **MyRectangle** object and a **MyCircle** Object; and

5- Explicitly specify all the classes imported and used in your Java code.



Best wishes
Hesham A. Auda
06-28-2022

Table of Contents

Solution:

This project contains seven classes, an enum class, an interface, and a main class. Main class contains everything together to generate the desired output using all the classes. MyShape class implements MyShapeInterface. Here, MyShape acts as a super class to MyLine, MyRectangle, MyOval, MyArc. And these classes extend MyShape. MyOval itself is the superclass of MyCircle. MyArc class has a MyOval object which contains the oval in which the arc lies. MyColor enum class is used by all classes except the Main class which demonstrates the color of shapes/points. And MyPoint is used by all classes to describe their coordinates.

Classes and Methods:

- **Main.java:**
  Uses the JavaFx library to draw shapes using graphics by calling on the appropriate stage, scene, canvas, and graphics classes. The height and width of canvas are determined by variables so that they can be changed. Additionally, all the dimensions of the shapes depend on canvas' smallest dimension. Main class outputs three different windows, each window showing different outputs. Part-1 of the output is the circular pizza pie with some slices, part-2 shows bounding rectangle of a MyLine, a MyCircle and a MyArc object. Part-3 shows intersection of two MyRectangle objects and a MyRectangle object and a MyCircle object.

  - ***Imports:***

    - javafx.application.Application;
    - javafx.scene.Scene;
    - javafx.scene.canvas.Canvas;
    - javafx.scene.canvas.GraphicsContext;
    - javafx.scene.layout.StackPane;
    - javafx.stage.Stage;
    - java.io.IOException;

  - ***Methods:***

    - **start();** method where JavaFx elements are handled. There are three windows that this method outputs. In window-1 it takes a MyCircle object and draws some MyArc objects on it. In window-2 it takes a MyLine, a MyArc and a MyCircle object and draws their bounding rectangle. In window-3 it takes two MyRectangle objects and draws their intersecting MyRectangle object. It then takes another MyRectangle and a MyCircle object and draws their intersection.

- **main();** main function

- **MyPoint.java:**
  This class is used by all classes to describe their coordinates. It also has a method to calculate the distance between two points and a method to calculate the angle line between two points created with the x-axis.

  - ***Attributes:***

    - **double x, y :** represents x and y value of a coordinate.
    - **MyColor color:** describes color of a MyPoint object.

  - ***Constructors:***

    - There are four constructors of MyPoint class one of which is a copy constructor. These constructors initialize value of the attributes.

```
// constructors
public MyPoint(double x, double y) {
    this.x = x;
    this.y = y;
    this.color = MyColor.NOCOLOR;
}
public MyPoint(double x, double y, MyColor color) {
    this.x = x;
    this.y = y;
    this.color = color;
}
public MyPoint() {
    this.x = 0;
    this.y = 0;
    this.color = MyColor.NOCOLOR;
}
public MyPoint(MyPoint obj) { this(obj.x, obj.y, obj.color); }
```

  - ***Methods:***

    - **calculateDistance():** returns Euclidean distance between two MyPoint objects using the formula:$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

```
// returns distance between two points
public double calculateDistance(MyPoint p2) {
    return Math.sqrt((p2.y-this.y)*(p2.y-this.y)+(p2.x-this.x)*(p2.x-this.x));
}
```

- **getXAngle():** returns the angle that the line between two MyPoint objects create with the x-axis by calculating the slope first and then calculating the angle of the slope by using tan inverse formula: $\tan^{-1}(\frac{\Delta y}{\Delta x})$

```
// returns angle made with x-axis by two points
public double getXAngle(MyPoint p2) {
    double slope = (this.y - p2.y)/(this.x - p2.x);
    return Math.atan(slope);
}
```

- **MyColor.java:**
  This enum class is used to describe colors of shapes and return color of a shape in appropriate format.
  - ***Imports:***
    - javafx.scene.paint.Color;
  - ***Attributes:***
    - **double r, g, b, opacity:** describes r, g, b and opacity of a color.
  - ***Constructors:***
    - MyColor has two constructors. These constructors validate and initialize value of r,g,b and opacity attributes using given values.

```
MyColor() {
    this.r = 0;
    this.g = 0;
    this.b = 0;
    this.opacity = 255;
}
// All argument constructors
MyColor(int r, int g, int b, int opacity) {
    if(r<0 || r>255) { r = 0; }
    if(g<0 || g>255) { g = 0; }
    if(b<0 || b>255) { b = 0; }
    if(opacity<0 || opacity>255) { opacity = 0; }
    this.r = r;
    this.g = g;
    this.b = b;
    this.opacity = opacity;
}
```

- ***Methods:***

  - **getHexRepresentation():** returns hexadecimal code for color specified by converting each color to hexadecimal value and then adding them up in a String object.

```
// return hexadecimal value of the color specified by MyColor
public String getHexRepresentation() {
    String hex = "#";
    hex += Integer.toHexString(r);
    hex += Integer.toHexString(g);
    hex += Integer.toHexString(b);

    return hex;
}
```

  - **getColor():** returns the corresponding color value from javafx.scene.paint. Color using inserted rgb and opacity values.

```
    // returns javaFX representation of MyColor
    public Color getColor() { return Color.rgb(r, g, b, v: opacity/255.0); }
}
```

- **MyShape.java:**
  This abstract class is written as a superclass to MyLine, MyRectangle and MyOval class. This class contains a MyPoint object and a MyColor object to describe the shape. There are seven constructors for this class, which include a copy constructor and default constructor that sets all values to 0 and color to black. This class can be used to set the background color of the canvas.

  - ***Imports:***

    - javafx.scene.canvas.GraphicsContext;

  - ***Attributes:***

    - **MyPoint p:** used to keep a reference point of a MyShape object
    - **MyColor color:** is used to describe color of a MyShape object

  - ***Constructors:***

    - All the seven constructors of MyShape class initialized the values of its attributes.

```
public MyShape(MyPoint p) {
    this.p = p;
    color = MyColor.BLACK;
}
public MyShape(MyPoint p, MyColor color) {
    this.p = p;
    this.color = color;
}
public MyShape() {
    this.p = new MyPoint();
    color = MyColor.BLACK;
}
public MyShape(MyColor color) {
    this.p = new MyPoint();
    this.color = color;
}
public MyShape(double x, double y) {
    this.p = new MyPoint(x, y);
    this.color = MyColor.BLACK;
}
public MyShape(double x, double y, MyColor color) {
    this.p = new MyPoint(x, y);
    this.color = color;
}
public MyShape(MyShape obj1) { this(obj1.p, obj1.color); }
```

- ***Methods:***

  - **perimeter():** abstract method meant to return perimeter of a shape.
  - **area():** abstract method meant to return area of a shape.
  - **toString():** describes a MyShape object using it's MyPoint p attribute.
  - **draw():** abstract method meant to draw a shape.

```
    //  returns perimeter and area, meant to be overridden
    abstract double perimeter() ;
    abstract double area() ;

    // string description of MyShape containing MyPoint p
    @Override
    public String toString() {
        return "MyShape {" +
                "\n\tp: (" + p.x +
                ", "+ p.y+")\n}";
    }

    // changes background color
    abstract void draw(GraphicsContext graphicsContext) ;

}
```

- **MyLine.java:**
  This class is a subclass of MyShape class. It is used to draw a straight line connecting two points. This class has five constructors including a copy constructor that sets the end points and color of the line.

- ***Imports:***
  - javafx.scene.canvas.GraphicsContext;
- ***Attributes:***
  - **MyPoint p:** is passed on by its superclass, describes an end point of the line.
  - **MyColor color:** is passed on by its superclass, describing the line's color.
  - **MyPoint p2:** is used to describe another endpoint of the line.
- ***Constructors:***
  - As mentioned before, MyLine class has five constructors, they use superclass MyShape's constructor to initialize the inherited attributes.

```
public MyLine(MyPoint refPoint, MyPoint p2) {
    super(refPoint);
    this.p2 = p2;
}
public MyLine(MyPoint refPoint, MyPoint p2, MyColor color) {
    super(refPoint, color);
    this.p2 = p2;
}

public MyLine(double x1, double y1, double x2, double y2) {
    super(x1, y1);
    this.p2 = new MyPoint(x2, y2);
}

public MyLine(double x1, double y1, double x2, double y2, MyColor color) {
    super(x1, y1, color);
    this.p2 = new MyPoint(x2, y2);
}

public MyLine(MyLine obj) { this(obj.p, obj.p2, obj.color); }
```

- ***Methods:***
  - **getLine():** returns the current MyLine object.
  - **length():** returns length of the line using MyPoint's calculateDistance method.
  - **xAngle():** returns the angle a MyLine object creates with x-axis using MyPoint's calculateAngle method.
  - **perimeter():** obtained from MyShape class, overridden to return MyLine object's length as perimeter.
  - **area():** obtained from MyShape class, overwritten to return MyLine object's area, which is 0.0 as a line does not have area.

```java
// getter method
public MyLine getLine() { return this; }

// length, xAngle, perimeter, area methods
public double length() { return this.p.calculateDistance(this.p2); }
public double xAngle() { return this.p.getXAngle(this.p2); }
@Override
public double perimeter() { return this.length(); }
@Override
public double area() { return 0.0; }
```

- **toString():** Obtained from MyShape and overridden to return a String object containing the endpoints, length, angle with x-axis, area and perimeter
- **draw():** draws the line itself in the given GraphicsContext.

```java
// string representation of MyLine
@Override
public String toString() {
    return "MyLine {\n\tp1: ("+ p.x
            +", "+ p.y+ ")\n\tp2: ("+
            p2.x+ ", "+p2.y+")\n\tlength: "+this.length()+
            "\n\tangle: "+this.xAngle()+"\n}";
}
// Draws a line
@Override
public void draw(GraphicsContext graphicsContext) {
    graphicsContext.setStroke(color.getColor());
    graphicsContext.setLineWidth(2);
    graphicsContext.strokeLine(this.p.x, this.p.y, this.p2.x, this.p2.y);
}
```

- **getMyBoundingRectangle():** Obtained from MyShapeInterface, this method returns the bounding rectangle of the MyLine object. It calculates the height and width of the rectangle from the two end points of the line and creates a MyRectangle object using those dimensions and returns it.

- **pointInMyShape():** Takes a MyPoint object as parameter and checks whether the point is on the MyLine object. It formulates an equation for the MyLine object and returns true if the given MyPoint object satisfies the equation and returns false otherwise.

```java
public MyRectangle getMyBoundingRectangle(){
    double x1= (this.p.x + this.p2.x)/2.0;
    double y1= (this.p.y + this.p2.y)/2.0;
    double h= Math.abs(this.p.y-this.p2.y);
    double w= Math.abs(this.p.x-this.p2.x);
    MyRectangle boundingRectangle=new MyRectangle(x1,y1,h,w);
    return boundingRectangle;
}
public boolean pointInMyShape(MyPoint p){
    double m = (this.p.y-this.p2.y)/(this.p.x-this.p2.x);
    double b = this.p.y -m*this.p.x;
    if (p.y-m*p.x-b==0){
        return true;
    }
    else{
        return false;
    }
}
```

- **MyRectangle.java:**
  This class is a subclass of MyShape class. It is used to draw a rectangle of given height and width. This class has six constructors including a copy constructor that sets the value of all the attributes.

    - ***Imports:***

        - javafx.scene.canvas.GraphicsContext;

    - ***Attributes:***

        - **MyPoint p:** is passed on by superclass MyShape used to describe the geometric center of the rectangle.
        - **MyColor color:** is passed on by superclass MyShape and describes color of the rectangle
        - **double h, w:** describes height and width of the rectangle

    - ***Constructors:***

        - MyRectangle class has six constructors including a copy constructor.

```
public MyRectangle(MyPoint refPoint, double h, double w) {
    super(refPoint);
    this.h = h;
    this.w = w;
}
public MyRectangle(MyPoint refPoint, MyColor color, double h, double w) {
    super(refPoint, color);
    this.h = h;
    this.w = w;
}
public MyRectangle(double h, double w) {
    super();
    this.h = h;
    this.w = w;
}
public MyRectangle(MyColor color, double h, double w) {
    super(color);
    this.h = h;
    this.w = w;
}
public MyRectangle(double x, double y, double h, double w) {
    super(x, y);
    this.h = h;
    this.w = w;
}
public MyRectangle(double x, double y, MyColor color, double h, double w) {
    super(x, y, color);
    this.h = h;
    this.w = w;
}
public MyRectangle(MyRectangle obj) { this(obj.p, obj.color, obj.h, obj.w); }
```

- ***Methods:***

  - **getHeight():** returns height (h) of the rectangle.
  - **getWidth():** returns width (w) of the rectangle.
  - **getTLCP():** calculates and returns top left corner point of a rectangle

```
// get height, width, top left corner point
public double getHeight() { return this.h; }
public double getWidth() { return this.w; }
public MyPoint getTLCP() {
    double x = this.p.x - this.w/2;
    double y = this.p.y - this.h/2;
    MyPoint tclp = new MyPoint(x, y);
    return tclp;
}
```

  - **perimeter(), area():** Obtained from superclass MyShape which are overridden to return perimeter and area of the rectangle.
  - **toString():** Overridden method from superclass MyShape to return a String object containing top left corner, height, width, perimeter, area of the MyRectangle object.

```
@Override
public double perimeter() { return 2 * (this.h + this.w); }
@Override
public double area() { return (this.h * this.w); }

@Override
public String toString() {
    return "MyRectangle{" +
            "\n\ttop left corner: ("+(this.p.x - this.w/2)+
            ", "+(this.p.y + this.h/2)+")"+
            "\n\theight=" + h +
            "\n\twidth=" + w +
            "\n\tperimeter="+this.perimeter()+
            "\n\tarea="+this.area()+
            "\n}";
}
```

  - **draw():** draws the MyRectangle object in the given GraphicsContext.
  - **getMyBoundingRectangle():** Method obtained from MyShapeInterface and overridden to return the MyRectangle object itself as it is the bounding rectangle of itself.
  - **pointInMyShape():** Method obtained from MyShapeInterface which returns if a given point is in a MyRectangle object or not. This method takes a MyPoint object as parameter and checks if the point lies within the MyRectangle object's corners. Returns true if so, false otherwise.

```java
// draw rectangle
@Override
public void draw(GraphicsContext graphicsContext) {
    graphicsContext.setFill(this.color.getColor());
    graphicsContext.fillRect( v: this.p.x-this.w/2,  v1: this.p.y-this.h/2, this.w, this.h);
}

@Override
public MyRectangle getMyBoundingRectangle() { return this; }
public boolean pointInMyShape(MyPoint p){
    MyPoint tlcp= this.getTLCP();
    double x2= tlcp.x+this.w;
    double y2=tlcp.y+this.h;
    if (p.x>=tlcp.x && p.x<=x2 && p.y>=tlcp.y && p.y<=y2){
        return true;
    }
    else{
        return false;
    }
}
```

- **MyOval.java:**
  This class is a subclass of MyShape class. It is used to draw an oval of given height and width. This class has seven constructors including a copy constructor that set the value of all the attributes.

  - ***Imports:***

    - javafx.scene.canvas.GraphicsContext;

  - ***Attributes:***

    - **MyPoint p:** is passed on by superclass MyShape used to describe the center of the oval.
    - **MyColor color:** is passed on by superclass MyShape and describes the color of the oval.
    - **double h, w:** describes height and width of the oval.

  - ***Constructors:***

    - MyOval class has seven constructors including a copy constructor.

```
public MyOval(MyPoint refPoint, double h, double w) {
    super(refPoint);
    this.h = h;
    this.w = w;
}
public MyOval(MyPoint refPoint, MyColor color, double h, double w) {
    super(refPoint, color);
    this.h = h;
    this.w = w;
}
public MyOval(double h, double w) {
    super();
    this.h = h;
    this.w = w;
}
```

```
public MyOval(MyColor color, double h, double w) {
    super(color);
    this.h = h;
    this.w = w;
}
public MyOval(double x, double y, double h, double w) {
    super(x, y);
    this.h = h;
    this.w = w;
}
public MyOval(double x, double y, MyColor color, double h, double w) {
    super(x, y, color);
    this.h = h;
    this.w = w;
}
public MyOval(MyOval obj) { this(obj.p, obj.color, obj.h, obj.w); }
```

- ***Methods:***

  - **getMinorAxis():** returns semi minor axis length of the MyOval object. Semi major axis is the smaller axis of the oval.
  - **getMajorAxis():** returns semi major axis length of the MyOval object. Semi major axis is the larger axis of the oval.
  - **getCenter():** returns the center point of oval.
  - **getWidth():** returns width of the oval.
  - **getHeight():** returns height of the oval.

```
public MyPoint getCenter() { return this.p; }
public double getWidth() { return this.w; }
public double getHeight() { return this.h; }

// get semi major and semi minor axis
public double getMinorAxis() { return Math.min(this.h, this.w)/2; }
public double getMajorAxis() { return Math.max(this.h, this.w)/2; }
```

  - **perimeter(), area():** Overridden method from superclass MyShape which return perimeter and area of the MyOval object. Area of an oval can be calculated using the formula: Π* semi major *semi minor. Reference: https://www.cuemath.com/measurement/perimeter-of-ellipse/

```java
@Override
public double perimeter() {
    double s = ((h-w)*(h-w))/((h+w)*(h+w));
    double p =  Math.PI * (h+w) * (1 + s/4 +
            Math.pow(s, 2)/64 +
            Math.pow(s, 3)/256 +
            25*Math.pow(s, 4)/16384 +
            49*Math.pow(s, 5)/65536 +
            441*Math.pow(s, 6)/1048576);
    return p;
}

// area of oval
@Override
public double area() { return Math.PI * this.h/2 * this.w/2; }
```

- **toString():** Overridden method from superclass MyShape which returns a String object containing semi major axis, semi minor axis length, perimeter, area of the MyOval object.
- **draw():** draws the MyOval object in the given GraphicsContext.

```java
@Override
public String toString() {
    return "MyOval{" +
            "\n\tsemi minor axis=" + this.getMinorAxis() +
            "\n\tsemi major axis=" + this.getMajorAxis() +
            "\n\tperimeter=" + this.perimeter() +
            "\n\tarea=" + this.area() +
            "\n}";
}

// draw the oval
@Override
public void draw(GraphicsContext graphicsContext) {
    graphicsContext.setFill(this.color.getColor());
    graphicsContext.fillOval( v: this.p.x-this.w/2,  v1: this.p.y-this.h/2, this.w, this.h);
}
```

- **getMyBoundingRectangle():** Overridden method from MyShapeInterface which returns a MyRectangle object that bound the MyOval object. It creates a MyRectangle object using center p, height, and width of the MyOval object. Then it returns the created MyRectangle object.
- **pointInMyShape():** overridden method from MyShapeInterface which takes a MyPoint object and checks whether the MyPoint object lies within the MyOval object. This method derives an equation for the MyOval object and returns true if the given point suffices the equation, false otherwise.

```java
@Override
public MyRectangle getMyBoundingRectangle() {
    MyRectangle boundingRectangle = new MyRectangle(this.p, this.h,this.w);
    return boundingRectangle;
}

@Override
public boolean pointInMyShape(MyPoint p) {
    double chk = (Math.pow(p.x-this.p.x,2)/Math.pow(this.w/2,2))+(Math.pow(p.y-this.p.y,2)/Math.pow(this.h/2,2));
    if(chk<=1){
        return true;
    }
    else{
        return false;
    }
}
```

- **MyArc.java:**
  This class is a subclass of MyShape class. It is used to draw an arc of an oval given the starting angle and body angle of an arc. This class has 3 constructors including a copy constructor. These constructors set initial values of all the attributes using given parameters.
  - ***Imports:***
    - Javafx.scene.canvas.GraphicsContext
    - Javafx.scene.shape.ArcType
  - ***Attributes:***
    - **MyPoint p:** is obtained from its superclass MyShape and this point describes the starting point of the arc, which is calculated in constructor using appropriate formula.
    - **MyColor color:** is obtained from superclass MyShape which describes the color of the arc.
    - **MyOval o:** Describes the oval in which the arc resides.
    - **double startAngle:** Describes the starting angle of the arc, which means at which angle the arc starts.
    - **double angle:** Describes the angle of the arc itself.
  - ***Constructors:***
    - MyArc class has three constructors including a copy constructor

```java
public MyArc(MyColor color, MyOval o, double startAngle, double angle) {
    super(color);
    this.o = o;
    this.startAngle = startAngle;
    this.angle = angle;

    double h=this.o.getHeight(), w=this.o.getWidth();
    double ang1=Math.toRadians(startAngle);
    double sqrt = Math.sqrt(Math.pow(h, 2) +
            Math.pow(w * Math.tan(ang1), 2));
    double x = this.o.p.x+(h*w)/ sqrt;
    double y = this.o.p.y+(w*h*Math.tan(ang1))/ sqrt;

    this.p = new MyPoint(x, y);
}

public MyArc(MyOval o, double startAngle, double angle) {
    super();
    this.o = o;
    this.startAngle = startAngle;
    this.angle = angle;

    double h=this.o.getHeight(), w=this.o.getWidth();
    double ang1=Math.toRadians(startAngle);
    double sqrt = Math.sqrt(Math.pow(h,2) +
            Math.pow(w*Math.tan(ang1), 2));
    double x = this.o.p.x+(h*w)/sqrt;
    double y = this.o.p.y+(w*h*Math.tan(ang1))/sqrt;

    this.p = new MyPoint(x, y);
}

public MyArc(MyArc arc) { this(arc.color, arc.o, arc.startAngle, arc.angle); }
```

- ***Methods:***
  - **area():** This method is overridden from MyShape superclass. It calculates and returns area of the MyArc object. This method first calculates radian value of startAngle, angle and the finishAngle of the MyArc object. Then it calculates area using the angles, height, and width of the oval by applying appropriate formula.
  - **perimeter():** This method is overridden from superclass MyShape, it calculates and returns perimeter of the MyArc object. This method first calculates finish point p2 of the arc and then calculates perimeter using length() method and MyPoint class's calculateDistance(). Perimeter equals to length of the arc+length of both sides.

```java
@Override
double perimeter() {
    double h=this.o.getHeight(), w=this.o.getWidth();
    double ang2= Math.toRadians(startAngle +angle);
    double x, y;
    MyPoint p1 = this.p;
    double len = p1.calculateDistance(this.o.p);
    double v = Math.pow(h, 2) +
            Math.pow(w * Math.tan(ang2), 2);
    x = this.o.p.x+(h*w)/Math.sqrt(v);
    y = this.o.p.y+(w*h*Math.tan(ang2))/Math.sqrt(v);
    MyPoint p2 = new MyPoint(x, y);
    len = len + p2.calculateDistance(this.o.p);

    return len+this.length();
}

@Override
double area() {
    double h=this.o.getHeight(), w=this.o.getWidth();
    double ang1=Math.toRadians(startAngle), ang2= Math.toRadians(startAngle +angle), ang3=Math.toRadians(angle);

    double area = 0.5*h*w*(ang3-
            (Math.atan((h-w)*Math.sin(2.0*ang2))/ ((h+w)+(h-w)*Math.cos(2*ang2)))-
            Math.atan((h-w)*Math.sin(2*ang1))/((h+w)+(h-w)*Math.cos(2*ang1)));
    return area;
}
```

- **length():** This method calculates and returns the length of the MyArc object. This method calculates finish point p2 of the arc and then uses the arc's starting point p1 and finishing point p2 to calculate length of the arc using appropriate formula.
- **toString():** This method is overridden from superclass MyShape which returns a String object containing information about the containing MyOval, staring angle, angle, area, length, perimeter.
- **draw():** draws the MyArc object in the given GraphicsContext object.

```java
public String toString(){
    return "MyArc{" +
            "\n\tStart Angle= " + this.startAngle +
            "\n\tBody Angle=" + this.angle +
            "\n\tLength= "+this.length()+
            "\n\tperimeter=" + this.perimeter() +
            "\n\tarea=" + this.area() +
            "\n\tstart point: ("+this.p.x+", "+this.p.y+")"+
            "\n\tAssociated Oval: "+this.o.toString()+
            "\n}";
}
@Override
void draw(GraphicsContext graphicsContext){
    graphicsContext.setFill(this.color.getColor());
    graphicsContext.fillArc( v: this.o.p.x-this.o.getWidth()/2, v1: this.o.p.y-this.o.getHeight()/2,
            this.o.getWidth(),this.o.getHeight(), startAngle, angle, ArcType.ROUND);
}
public double length(){
    double h=this.o.getHeight(), w=this.o.getWidth();
    double ang2= Math.toRadians(startAngle +angle);
    double x, y;
    MyPoint p1 = this.p;
    double sqrt = Math.sqrt(Math.pow(h, 2) +
            Math.pow(w * Math.tan(ang2), 2));
    x = this.o.p.x+(h*w)/ sqrt;
    y = this.o.p.y+(w*h*Math.tan(ang2))/ sqrt;
    MyPoint p2 = new MyPoint(x, y);
    return 0.5 * Math.PI/Math.sqrt(2)*p1.calculateDistance(p2);
}
```

- **getMyBoundingRectangle():** This method is overridden from the interface MyShapeInterface. It returns the oval's bounding rectangle on which the arc resides.

- **PointInMyShape():** This method is overridden from the interface MyShapeInterface. It takes a MyPoint object in parameter and return true if the object lies within the MyArc object, and false otherwise. This method checks if the given point is in the oval of the arc at first. If it is, then it calculates the angle the line between given point and center of oval creates with x-axis. If this angle is between start angle and finish angle of the arc then the point is in the arc, otherwise it's not.

```
    @Override
    public MyRectangle getMyBoundingRectangle() { return this.o.getMyBoundingRectangle(); }

    @Override
    public boolean pointInMyShape(MyPoint p) {
        if(this.o.pointInMyShape(p)) {
            double ang = this.o.p.getXAngle(p);
            ang = Math.toDegrees(ang);
            if(ang >= startAngle && ang <= startAngle +angle){
                return true;
            }
        }
        return false;
    }
}
```

- **MyCircle.java:**
  This class is extended from class MyOval. This class has three constructors and all of them takes radius instead of height and width, then it calls super constructor to create appropriate circle.

  - ***Attributes:***

    - **MyPoint p:** defines the center point of the MyCircle object.
    - **MyColor color:** defines the fill color of the MyCircle object.
    - **double h, w:** used to describe radius of the circle. Constructor sets value of h and w to the radius of the circle

  - ***Constructors:***

    - There are three constructors in MyCircle class, they use superclass MyOval's constructors to initialize all the values.

```
public MyCircle(MyPoint center, double r) { super(center,r,r); }
public MyCircle(MyPoint center, double r, MyColor c) { super(center,c,r,r); }
public MyCircle(double x,double y, double r) { super(x,y,r,r); }
public MyCircle(double x,double y, double r, MyColor c ) { super(x,y,c,r,r); }
```

- ***Methods:***

  - **area():** overridden method from superclass MyOval which is used to calculate and return Area of the MyCircle object.
  - **perimeter():** overridden method from superclass MyOval which is used to calculate and return perimeter of the MyCircle object.
  - **toString():** overridden method from superclass MyOval which returns a String object containing center, radius, perimeter and the area of the MyCircle object.

```
    @Override
    public double area() { return Math.PI*Math.pow(this.getHeight(),2); }

    @Override
    public double perimeter() { return 2*Math.PI*this.getMajorAxis(); }
    @Override
    public String toString() {
        return "MyCircle{" +
                "\n\tcenter= ("+this.p.x+", "+this.p.y+")"+
                "\n\tradius=" + this.getMinorAxis() +
                "\n\tperimeter=" + this.perimeter() +
                "\n\tarea=" + this.area() +
                "\n}";
    }
}
```

  - **getHeight(), getWidth():** obtained from superclass MyOval, returns 2*radius of the MyCircle object.
  - **getMajorAxis(), getMinorAxis():** obtained from superclass MyOval, returns radius of the MyCircle object.
  - **draw():** obtained from superclass MyOval, this method draws MyCircle object in the given GraphicsContext.
  - **getMyBoundingRectangle():** Method obtained from MyOval class which returns a MyRectangle object that bounds the MyCircle object. It creates a MyRectangle object using center p, and radius of the MyCircle object as height and width. Then it returns the created MyRectangle object.

  - **pointInMyShape():** This method is obtained from MyShapeInterface which takes a MyPoint object and checks whether the MyPoint object lies within the MyCircle object. This method derives an equation for the MyCircle object and returns true if the given point suffices the equation, false otherwise.

- **MyShapeInterface.java:**
  This interface is implemented by MyShape class, which itself is extended by other classes. This interface has some abstract methods to be defined by the classes implementing it. It has a method to calculate intersection between two objects and draw the intersection.

  - ***Imports:***

    - Javafx.scene.canvas.Canvas
    - Javafx.scene.canvas.GraphicsContext

  - ***Methods:***

    - **getMyBoundingRectangle():** returns the bounding rectangle of the Shape implementing the method. This method is abstract, so each class implementing MyShapeInterface has to different this method in its own way.
    - **pointInMyShape():** abstract method, which takes a MyPoint object as parameter and returns whether the point lies within the MyShape object implementing the method.

```
abstract MyRectangle getMyBoundingRectangle();
abstract boolean pointInMyShape(MyPoint p);
```

    - **IntersectMyShapes():** static method returns the area two given MyShape objects intersect in. Takes two MyShape objects in parameter. If the shapes do not intersect, returns null. It is a static method so it cannot be overridden by the subclasses. This method first gets the bounding rectangle of the given MyShape using getBoundingRectangle() methods of respective class. Then it calculates the intersection of the calculated bounding rectangles and returns a MyRectangle if the bounding rectangles intersect and form a rectangle.

```
public static MyRectangle intersectMyShapes(MyShape s1, MyShape s2){
    MyRectangle rectangle1= s1.getMyBoundingRectangle();
    MyRectangle rectangle2= s2.getMyBoundingRectangle();

    MyPoint tlcp1 = rectangle1.getTLCP();
    MyPoint brcp1 = new MyPoint( x: tlcp1.x+rectangle1.getWidth(),  y: tlcp1.y+rectangle1.getHeight());

    MyPoint tlcp2 = rectangle2.getTLCP();
    MyPoint brcp2 = new MyPoint( x: tlcp2.x+ rectangle2.getWidth(),  y: tlcp2.y+rectangle2.getHeight());
    MyPoint n_tlcp=null, n_brcp=null;

    if(rectangle1.pointInMyShape(tlcp2)){
        n_tlcp = tlcp2;
        n_brcp = brcp1;
    }
    if(rectangle1.pointInMyShape(brcp2)) {
        n_brcp = brcp2;
        if(n_tlcp == null) {
            n_tlcp = tlcp1;
        }
    }
    if(n_brcp != null && n_tlcp != null) {
        double h = n_brcp.y - n_tlcp.y;
        double w = n_brcp.x - n_tlcp.x;
        MyRectangle intersection = new MyRectangle( x: n_tlcp.x+w/2,  y: n_tlcp.y+h/2, h, w);
        return intersection;
    }
    return null;
}
```

- **drawIntersectMyShape():** default method takes two MyShape objects in parameter and calculates their intersection using the intersectMyShapes() method. Then it creates a canvas and draws the intersecting rectangle on to it and returns the square.

```
    public default Canvas drawIntersectMyShape(MyShape shape1, MyShape shape2) {
        MyRectangle intersection = MyShapeInterface.intersectMyShapes(shape1, shape2);
        if(intersection==null) intersection = MyShapeInterface.intersectMyShapes(shape2, shape1);
        Canvas canvas = new Canvas( v: 1000,  v1: 600);
        GraphicsContext gc = canvas.getGraphicsContext2D();
        if(intersection!=null) {
            intersection.draw(gc);
        }
        return canvas;
    }
}
```

Screenshots of Source Code:

- **MyPoint.java:**

```java
package com.example.assignment2;
public class MyPoint {
    // x and y coordinate of a cartesian point
    public double x;
    public double y;
    // color of the shape
    public MyColor color;
    // constructors
    public MyPoint(double x, double y) {
        this.x = x;
        this.y = y;
        this.color = MyColor.NOCOLOR;
    }
    public MyPoint(double x, double y, MyColor color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }
    public MyPoint() {
        this.x = 0;
        this.y = 0;
        this.color = MyColor.NOCOLOR;
    }
    public MyPoint(MyPoint obj) { this(obj.x, obj.y, obj.color); }

    // returns distance between two points
    public double calculateDistance(MyPoint p2) {
        return Math.sqrt((p2.y-this.y)*(p2.y-this.y)+(p2.x-this.x)*(p2.x-this.x));
    }
    // returns angle made with x-axis by two points
    public double getXAngle(MyPoint p2) {
        double slope = (this.y - p2.y)/(this.x - p2.x);
        return Math.atan(slope);
    }
}
```

- **MyColor.java:**

```
package com.example.assignment2;
import javafx.scene.paint.Color;
public enum MyColor {

    ALICEBLUE( r: 240,  g: 248,  b: 255,  opacity: 255),
    ANTIQUEWHITE( r: 250,  g: 235,  b: 215,  opacity: 255),
    AQUA( r: 0,  g: 255,  b: 255,  opacity: 255),
    AQUAMARINE( r: 127,  g: 255,  b: 212,  opacity: 244),
    AZURE( r: 240,  g: 255,  b: 255,  opacity: 255),
    BEIGE( r: 245,  g: 245,  b: 220,  opacity: 255),
    BISQUE( r: 255,  g: 228,  b: 196,  opacity: 255),
    BLACK ( r: 0,  g: 0,  b: 0,  opacity: 255),
    BLANCHEDAUMOND( r: 255,  g: 235,  b: 205,  opacity: 255),
    BLUE( r: 0,  g: 0,  b: 255,  opacity: 255),
    BLUEVIOLET( r: 238,  g: 43,  b: 226,  opacity: 255),
    BROWN ( r: 165,  g: 42,  b: 42,  opacity: 255),
    BURLYWOOD( r: 222,  g: 184,  b: 135,  opacity: 255),
    CADETBLUE( r: 95,  g: 158,  b: 160,  opacity: 255),
    CHARTREUSE( r: 127,  g: 255,  b: 0,  opacity: 255),
    CHOCOLATE( r: 210,  g: 105,  b: 30,  opacity: 255),
    CORAL( r: 255,  g: 127,  b: 80,  opacity: 255),
    CORNFLOWERBLUE( r: 100,  g: 149,  b: 237,  opacity: 255),
    CORNSILK( r: 255,  g: 248,  b: 220,  opacity: 255),
    CRIMSON( r: 220,  g: 20,  b: 60,  opacity: 255),
    CYAN( r: 0,  g: 255,  b: 255,  opacity: 255),
    DARKBLUE( r: 0,  g: 0,  b: 13,  opacity: 255),
    DARKCYAN( r: 0,  g: 139,  b: 139,  opacity: 255),
    DARKGOLDENROD( r: 184,  g: 134,  b: 11,  opacity: 255),
    DARKGREY( r: 169,  g: 169,  b: 169,  opacity: 255),
    DARKGREEN( r: 0,  g: 100,  b: 0,  opacity: 255),
    DARKKHAKI( r: 189,  g: 183,  b: 107,  opacity: 255),
    DARKMAGENTA( r: 139,  g: 0,  b: 139,  opacity: 255),
    DARKOLIVEGREEN( r: 85,  g: 107,  b: 47,  opacity: 255),
    DARKORANGE( r: 255,  g: 140,  b: 0,  opacity: 255),
    DARKORCHID( r: 153,  g: 50,  b: 204,  opacity: 255),
```

```
    DARKRED( r: 139,  g: 0,  b: 0,  opacity: 255),
    DARKSALMON( r: 233,  g: 150,  b: 122,  opacity: 255),
    DARKSEAGREEN( r: 143,  g: 188,  b: 143,  opacity: 255),
    DARKSLATEBLUE( r: 72,  g: 61,  b: 139,  opacity: 255),
    DARKSLATEGRAY( r: 47,  g: 79,  b: 79,  opacity: 255),
    DARKTURQUOISE( r: 0,  g: 206,  b: 209,  opacity: 255),
    DARKVIOLET( r: 148,  g: 0,  b: 211,  opacity: 255),
    DARKPINK( r: 255,  g: 20,  b: 147,  opacity: 255),
    DARKSKYBLUE( r: 0,  g: 191,  b: 255,  opacity: 255),
    DARKGRAY( r: 105,  g: 105,  b: 105,  opacity: 255),

    LAVANDER ( r: 215,  g: 180,  b: 243,  opacity: 255),
    YELLOW ( r: 255,  g: 255,  b: 0,  opacity: 255),
    SKYBLUE ( r: 0,  g: 181,  b: 226,  opacity: 255),
    LIME ( r: 199,  g: 234,  b: 70,  opacity: 255),
    WHITE( r: 255,  g: 255,  b: 255,  opacity: 255),
    RED ( r: 255,  g: 0,  b: 0,  opacity: 255),
    GREEN( r: 0,  g: 255,  b: 0,  opacity: 255),

    NOCOLOR( r: 0,  g: 0,  b: 0,  opacity: 0);

    private int r, g, b, opacity;

    // Default constructor
    MyColor() {
        this.r = 0;
        this.g = 0;
        this.b = 0;
        this.opacity = 255;
    }

    // All argument constructors
    MyColor(int r, int g, int b, int opacity) {
        if(r<0 || r>255){
            r = 0;
        }
        if(g<0 || g>255){
            g = 0;
        }
        if(b<0 || b>255){
            b = 0;
        }
        if(opacity<0 || opacity>255){
            opacity = 0;
        }
        this.r = r;
        this.g = g;
        this.b = b;
        this.opacity = opacity;
    }

    // return hexadecimal value of the color specified by MyColor
    public String getHexRepresentation() {
        String hex = "#";
        hex += Integer.toHexString(r);
        hex += Integer.toHexString(g);
        hex += Integer.toHexString(b);

        return hex;
    }

    // returns javaFX representation of MyColor
    public Color getColor() { return Color.rgb(r, g, b,  v: opacity/255.0); }
}
```

- **MyShape.java:**

```
package com.example.assignment2;

import javafx.scene.canvas.GraphicsContext;

public abstract class MyShape implements MyShapeInterface {
    public MyPoint p;
    public MyColor color;

    // constructors
    public MyShape(MyPoint p) {
        this.p = p;
        color = MyColor.BLACK;
    }
    public MyShape(MyPoint p, MyColor color) {
        this.p = p;
        this.color = color;
    }
    public MyShape() {
        this.p = new MyPoint();
        color = MyColor.BLACK;
    }
    public MyShape(MyColor color) {
        this.p = new MyPoint();
        this.color = color;
    }
    public MyShape(double x, double y) {
        this.p = new MyPoint(x, y);
        this.color = MyColor.BLACK;
    }
    public MyShape(double x, double y, MyColor color) {
        this.p = new MyPoint(x, y);
        this.color = color;
    }
    public MyShape(MyShape obj1) { this(obj1.p, obj1.color); }

    //  returns perimeter and area, meant to be overridden
    abstract double perimeter() ;
    abstract double area() ;

    // string description of MyShape containing MyPoint p
    @Override
    public String toString() {
        return "MyShape {" +
                "\n\tp: (" + p.x +
                ", "+ p.y+")\n}";
    }

    // changes background color
    abstract void draw(GraphicsContext graphicsContext) ;

}
```

- **MyLine.java:**

```java
package com.example.assignment2;

import javafx.scene.canvas.GraphicsContext;

public class MyLine extends MyShape{
    public MyPoint p2;

    // all constructors
    public MyLine(MyPoint refPoint, MyPoint p2) {
        super(refPoint);
        this.p2 = p2;
    }
    public MyLine(MyPoint refPoint, MyPoint p2, MyColor color) {
        super(refPoint, color);
        this.p2 = p2;
    }

    public MyLine(double x1, double y1, double x2, double y2) {
        super(x1, y1);
        this.p2 = new MyPoint(x2, y2);
    }

    public MyLine(double x1, double y1, double x2, double y2, MyColor color) {
        super(x1, y1, color);
        this.p2 = new MyPoint(x2, y2);
    }

    public MyLine(MyLine obj) { this(obj.p, obj.p2, obj.color); }

    // getter method
    public MyLine getLine() { return this; }

    // length, xAngle, perimeter, area methods
    public double length() {
        return this.p.calculateDistance(this.p2);
    public double xAngle() {
        return this.p.getXAngle(this.p2);
    }
    @Override
    public double perimeter(){
        return this.length();
    }
    @Override
    public double area() {
        return 0.0;
    }
    // string representation of MyLine
    @Override
    public String toString() {
        return "MyLine {\n\tp1: ("+ p.x
                +", "+ p.y+ ")\n\tp2: ("+
                p2.x+ ", "+p2.y+")\n\tlength: "+this.length()+
                "\n\tangle: "+this.xAngle()+"\n}";
    }
    // Draws a line
    @Override
    public void draw(GraphicsContext graphicsContext) {
        graphicsContext.setStroke(color.getColor());
        graphicsContext.setLineWidth(2);
        graphicsContext.strokeLine(this.p.x, this.p.y, this.p2.x, this.p2.y);
    }
```

** this part has been added in assignment-2

```
    public MyRectangle getMyBoundingRectangle(){
        double x1= (this.p.x + this.p2.x)/2.0;
        double y1= (this.p.y + this.p2.y)/2.0;
        double h= Math.abs(this.p.y-this.p2.y);
        double w= Math.abs(this.p.x-this.p2.x);
        MyRectangle boundingRectangle=new MyRectangle(x1,y1,h,w);
        return boundingRectangle;
    }
    public boolean pointInMyShape(MyPoint p){
        double m = (this.p.y-this.p2.y)/(this.p.x-this.p2.x);
        double b = this.p.y -m*this.p.x;
        if (p.y-m*p.x-b==0){
            return true;
        }
        else{
            return false;
        }
    }
}
```

- **MyRectangle.java:**

```
package com.example.assignment2;
import javafx.scene.canvas.GraphicsContext;
public class MyRectangle extends MyShape{
    // height and width of rectangle
    private double h, w;

    // constructors
    public MyRectangle(MyPoint refPoint, double h, double w) {
        super(refPoint);
        this.h = h;
        this.w = w;
    }
    public MyRectangle(MyPoint refPoint, MyColor color, double h, double w) {
        super(refPoint, color);
        this.h = h;
        this.w = w;
    }
    public MyRectangle(double h, double w) {
        super();
        this.h = h;
        this.w = w;
    }
    public MyRectangle(MyColor color, double h, double w) {
        super(color);
        this.h = h;
        this.w = w;
    }
    public MyRectangle(double x, double y, double h, double w) {
        super(x, y);
        this.h = h;
        this.w = w;
    }
    public MyRectangle(double x, double y, MyColor color, double h, double w) {
        super(x, y, color);
        this.h = h;
```

```
        this.w = w;
    }
    public MyRectangle(MyRectangle obj) { this(obj.p, obj.color, obj.h, obj.w); }

    // get height, width, top left corner point
    public double getHeight() { return this.h; }
    public double getWidth() { return this.w; }
    public MyPoint getTLCP() {
        double x = this.p.x - this.w/2;
        double y = this.p.y - this.h/2;
        MyPoint tclp = new MyPoint(x, y);
        return tclp;
    }

    //perimeter and area of rectangle
    @Override
    public double perimeter() { return 2 * (this.h + this.w); }
    @Override
    public double area() { return (this.h * this.w); }

    @Override
    public String toString() {
        return "MyRectangle{" +
                "\n\ttop left corner: ("+(this.p.x - this.w/2)+
                ", "+(this.p.y + this.h/2)+")"+
                "\n\theight=" + h +
                "\n\twidth=" + w +
                "\n\tperimeter="+this.perimeter()+
                "\n\tarea="+this.area()+
                "\n}";
    }

    // draw rectangle
    @Override
    public void draw(GraphicsContext graphicsContext) {
```

** this part has been added in assignment-2

```
        graphicsContext.setFill(this.color.getColor());
        graphicsContext.fillRect( v: this.p.x-this.w/2,  v1: this.p.y-this.h/2, this.w, this.h);
    }

    @Override
    public MyRectangle getMyBoundingRectangle(){
        return this;
    }
    public boolean pointInMyShape(MyPoint p){
        MyPoint tlcp= this.getTLCP();
        double x2= tlcp.x+this.w;
        double y2=tlcp.y+this.h;
        if (p.x>=tlcp.x && p.x<=x2 && p.y>=tlcp.y && p.y<=y2){
            return true;
        }
        else{
            return false;
        }
    }
}
```

- **MyOval.java:**

```java
package com.example.assignment2;
import javafx.scene.canvas.GraphicsContext;
public class MyOval extends MyShape{
    // height and width of the oval
    private double h, w;
    // constructors
    public MyOval(MyPoint refPoint, double h, double w) {
        super(refPoint);
        this.h = h;
        this.w = w;
    }
    public MyOval(MyPoint refPoint, MyColor color, double h, double w) {
        super(refPoint, color);
        this.h = h;
        this.w = w;
    }
    public MyOval(double h, double w) {
        super();
        this.h = h;
        this.w = w;
    }
    public MyOval(MyColor color, double h, double w) {
        super(color);
        this.h = h;
        this.w = w;
    }
    public MyOval(double x, double y, double h, double w) {
        super(x, y);
        this.h = h;
        this.w = w;
    }
    public MyOval(double x, double y, MyColor color, double h, double w) {
        super(x, y, color);
        this.h = h;
        this.w = w;
```

```java
    public MyOval(MyOval obj) {
        this(obj.p, obj.color, obj.h, obj.w);
    }

    public MyPoint getCenter() {
        return this.p;
    }
    public double getWidth(){
        return this.w;
    }
    public double getHeight(){
        return this.h;
    }

    // get semi major and semi minor axis
    public double getMinorAxis() { return Math.min(this.h, this.w)/2; }
    public double getMajorAxis() {
        return Math.max(this.h, this.w)/2;
    }

    // close estimation of perimeter calculated
    @Override
    public double perimeter() {
        double s = ((h-w)*(h-w))/((h+w)*(h+w));
        double p =  Math.PI * (h+w) * (1 + s/4 +
                Math.pow(s, 2)/64 +
                Math.pow(s, 3)/256 +
                25*Math.pow(s, 4)/16384 +
                49*Math.pow(s, 5)/65536 +
                441*Math.pow(s, 6)/1048576);
        return p;
    }
```

```java
// area of oval
@Override
public double area() {
    return Math.PI * this.h/2 * this.w/2;
}
@Override
public String toString() {
    return "MyOval{" +
            "\n\tsemi minor axis=" + this.getMinorAxis() +
            "\n\tsemi major axis=" + this.getMajorAxis() +
            "\n\tperimeter=" + this.perimeter() +
            "\n\tarea=" + this.area() +
            "\n}";
}
// draw the oval
@Override
public void draw(GraphicsContext graphicsContext) {
    graphicsContext.setFill(this.color.getColor());
    graphicsContext.fillOval( v: this.p.x-this.w/2,  v1: this.p.y-this.h/2, this.w, this.h);
}
```

** this part has been added in assignment-2

```java
@Override
public MyRectangle getMyBoundingRectangle() {
    MyRectangle boundingRectangle = new MyRectangle(this.p, this.h,this.w);
    return boundingRectangle;
}
@Override
public boolean pointInMyShape(MyPoint p) {
    double chk = (Math.pow(p.x-this.p.x,2)/Math.pow(this.w/2,2))+(Math.pow(p.y-this.p.y,2)/Math.pow(this.h/2,2));
    if(chk<=1){
        return true;
    }
    else{
        return false;
    }
}
```

Classes added in Assignment-2:

- **MyArc.java:**

```java
package com.example.assignment2;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;

public class MyArc extends MyShape{
    public MyOval o;
    public double startAngle;
    public double angle;

    public MyArc(MyColor color, MyOval o, double startAngle, double angle) {
        super(color);
        this.o = o;
        this.startAngle = startAngle;
        this.angle = angle;

        double h=this.o.getHeight(), w=this.o.getWidth();
        double ang1=Math.toRadians(startAngle);
        double v = Math.pow(h, 2) +
                Math.pow(w * Math.tan(ang1), 2);
        double x = this.o.p.x+(h*w)/Math.sqrt(v);
        double y = this.o.p.y+(w*h*Math.tan(ang1))/Math.sqrt(v);

        this.p = new MyPoint(x, y);
    }

    public MyArc(MyOval o, double startAngle, double angle) {
        super();
        this.o = o;
        this.startAngle = startAngle;
        this.angle = angle;

        double h=this.o.getHeight(), w=this.o.getWidth();
        double ang1=Math.toRadians(startAngle);
        double v = Math.pow(h, 2) +
                Math.pow(w * Math.tan(ang1), 2);
        double x = this.o.p.x+(h*w)/Math.sqrt(v);
        double y = this.o.p.y+(w*h*Math.tan(ang1))/Math.sqrt(v);

        this.p = new MyPoint(x, y);
    }

    public MyArc(MyArc arc) {
        this(arc.color, arc.o, arc.startAngle, arc.angle);
    }

    @Override
    double perimeter() {
        double h=this.o.getHeight(), w=this.o.getWidth();
        double ang2= Math.toRadians(startAngle +angle);
        double x, y;
        MyPoint p1 = this.p;
        double len = p1.calculateDistance(this.o.p);
        double v = Math.pow(h, 2) +
                Math.pow(w * Math.tan(ang2), 2);
        x = this.o.p.x+(h*w)/Math.sqrt(v);
        y = this.o.p.y+(w*h*Math.tan(ang2))/Math.sqrt(v);
        MyPoint p2 = new MyPoint(x, y);
        len = len + p2.calculateDistance(this.o.p);

        return len+this.length();
    }

    @Override
    double area() {
        double h=this.o.getHeight(), w=this.o.getWidth();
        double ang1=Math.toRadians(startAngle), ang2= Math.toRadians(startAngle +angle), ang3=Math.toRadians(angle);

        double area = 0.5*h*w*(ang3-
                (Math.atan((h-w)*Math.sin(2.0*ang2))/ ((h+w)+(h-w)*Math.cos(2*ang2)))-
```

```
                Math.atan((h-w)*Math.sin(2*ang1))/((h+w)+(h-w)*Math.cos(2*ang1)));
        return area;
    }

    public String toString(){
        return "MyArc{" +
                "\n\tStart Angle= " + this.startAngle +
                "\n\tBody Angle=" + this.angle +
                "\n\tLength= "+this.length()+
                "\n\tperimeter=" + this.perimeter() +
                "\n\tarea=" + this.area() +
                "\n\tstart point: ("+this.p.x+", "+this.p.y+")"+
                "\n\tAssociated Oval: "+this.o.toString()+
                "\n}";
    }
    @Override
    void draw(GraphicsContext graphicsContext){
        graphicsContext.setFill(this.color.getColor());
        graphicsContext.fillArc( v: this.o.p.x-this.o.getWidth()/2, v1: this.o.p.y-this.o.getHeight()/2,
                this.o.getWidth(),this.o.getHeight(), startAngle, angle, ArcType.ROUND);
    }
```

```
    public double length(){
        double h=this.o.getHeight(), w=this.o.getWidth();
        double ang2= Math.toRadians(startAngle +angle);
        double x, y;
        MyPoint p1 = this.p;
        double sqrt = Math.sqrt(Math.pow(h, 2) +
                Math.pow(w * Math.tan(ang2), 2));
        x = this.o.p.x+(h*w)/ sqrt;
        y = this.o.p.y+(w*h*Math.tan(ang2))/ sqrt;
        MyPoint p2 = new MyPoint(x, y);
        return 0.5 * Math.PI/Math.sqrt(2)*p1.calculateDistance(p2);
    }
    @Override
    public MyRectangle getMyBoundingRectangle() { return this.o.getMyBoundingRectangle(); }

    @Override
    public boolean pointInMyShape(MyPoint p) {
        if(this.o.pointInMyShape(p)) {
            double ang = this.o.p.getXAngle(p);
            ang = Math.toDegrees(ang);
            if(ang >= startAngle && ang <= startAngle +angle){
                return true;
            }
        }
        return false;
    }
}
```

- **MyCircle.java:**

```java
package com.example.assignment2;

public class MyCircle extends MyOval{
    public MyCircle(MyPoint center, double r) { super(center,r,r); }
    public MyCircle(MyPoint center, double r, MyColor c) { super(center,c,r,r); }
    public MyCircle(double x,double y, double r) { super(x,y,r,r); }
    public MyCircle(double x,double y, double r, MyColor c ) { super(x,y,c,r,r); }

    @Override
    public double area() { return Math.PI*Math.pow(this.getHeight(),2); }

    @Override
    public double perimeter() { return 2*Math.PI*this.getMajorAxis(); }
    @Override
    public String toString() {
        return "MyCircle{" +
                "\n\tcenter= ("+this.p.x+", "+this.p.y+")"+
                "\n\tradius=" + this.getMinorAxis() +
                "\n\tperimeter=" + this.perimeter() +
                "\n\tarea=" + this.area() +
                "\n}";
    }
}
```

- **MyShapeInterface.java:**

```java
package com.example.assignment2;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

public interface MyShapeInterface {
    abstract MyRectangle getMyBoundingRectangle();
    abstract boolean pointInMyShape(MyPoint p);
    public static MyRectangle intersectMyShapes(MyShape s1, MyShape s2){
        MyRectangle rectangle1= s1.getMyBoundingRectangle();
        MyRectangle rectangle2= s2.getMyBoundingRectangle();

        MyPoint tlcp1 = rectangle1.getTLCP();
        MyPoint brcp1 = new MyPoint( x: tlcp1.x+rectangle1.getWidth(),  y: tlcp1.y+rectangle1.getHeight());

        MyPoint tlcp2 = rectangle2.getTLCP();
        MyPoint brcp2 = new MyPoint( x: tlcp2.x+ rectangle2.getWidth(),  y: tlcp2.y+rectangle2.getHeight());
        MyPoint n_tlcp=null, n_brcp=null;

        if(rectangle1.pointInMyShape(tlcp2)){
            n_tlcp = tlcp2;
            n_brcp = brcp1;
        }
        if(rectangle1.pointInMyShape(brcp2)) {
            n_brcp = brcp2;
            if(n_tlcp == null) {
                n_tlcp = tlcp1;
            }
        }
```

```
        if(n_brcp != null && n_tlcp != null) {
            double h = n_brcp.y - n_tlcp.y;
            double w = n_brcp.x - n_tlcp.x;
            MyRectangle intersection = new MyRectangle( x: n_tlcp.x+w/2,  y: n_tlcp.y+h/2, h, w);
            return intersection;
        }
        return null;
    }

    public default Canvas drawIntersectMyShape(MyShape shape1, MyShape shape2) {
        MyRectangle intersection = MyShapeInterface.intersectMyShapes(shape1, shape2);
        if(intersection==null) intersection = MyShapeInterface.intersectMyShapes(shape2, shape1);
        Canvas canvas = new Canvas( v: 1000,  v1: 600);
        GraphicsContext gc = canvas.getGraphicsContext2D();
        if(intersection!=null) {
            intersection.draw(gc);
        }
        return canvas;
    }
}
```

- **Main.java:**

```
package com.example.assignment2;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
import java.io.IOException;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        // variable height and width
        int w = 1000;
        int h = 600;

        // height, width, center of the shapes variable dependent on w, h
        double r = 0.6*Math.min(h, w);
        double centerX = w/2.0;
        double centerY = h/2.0;

        StackPane root = new StackPane();
        Scene scene = new Scene(root, w, h);
        Canvas canvas = new Canvas(w, h);
        GraphicsContext graphicsContext = canvas.getGraphicsContext2D();

        MyCircle pizza = new MyCircle(centerX, centerY, r);
        pizza.draw(graphicsContext);

        MyArc slice1 = new MyArc(MyColor.LIME, pizza,  startAngle: 45,  angle: 45);
        slice1.draw(graphicsContext);

        MyArc slice2 = new MyArc(MyColor.CORAL, pizza,  startAngle: 325,  angle: 80);
        slice2.draw(graphicsContext);
```

```
MyArc slice3 = new MyArc(MyColor.RED, pizza,  startAngle: 200,  angle: 125);
slice3.draw(graphicsContext);

MyArc slice4 = new MyArc(MyColor.CYAN, pizza,  startAngle: 155,  angle: 45);
slice4.draw(graphicsContext);

MyArc slice5 = new MyArc(MyColor.BURLYWOOD, pizza,  startAngle: 90,  angle: 65);
slice5.draw(graphicsContext);

// showing drawn shapes on window
root.getChildren().add(canvas);
stage.setScene(scene);
stage.setTitle("Assignment-2");
stage.show();

// 2nd part
StackPane root1 = new StackPane();
Scene scene1 = new Scene(root1, w, h);
Canvas canvas1 = new Canvas(w, h);
GraphicsContext graphicsContext1 = canvas1.getGraphicsContext2D();

MyLine b_line = new MyLine( x1: 45,  y1: 155,  x2: 410,  y2: 514);
MyRectangle b_line_rec = b_line.getMyBoundingRectangle();
b_line_rec.color = MyColor.BISQUE;
b_line_rec.draw(graphicsContext1);
b_line.draw(graphicsContext1);

MyCircle b_circle = new MyCircle( x: 580,  y: 110,  r: 190, MyColor.CYAN);
MyRectangle b_circ_rec = b_circle.getMyBoundingRectangle();
b_circ_rec.color = MyColor.GREEN;
b_circ_rec.draw(graphicsContext1);
b_circle.draw(graphicsContext1);

MyOval b_oval = new MyOval( x: 650,  y: 400, h: 250,  w: 400);
MyArc b_arc = new MyArc(MyColor.BLANCHEDAUMOND, b_oval,  startAngle: 90,  angle: 270);
```

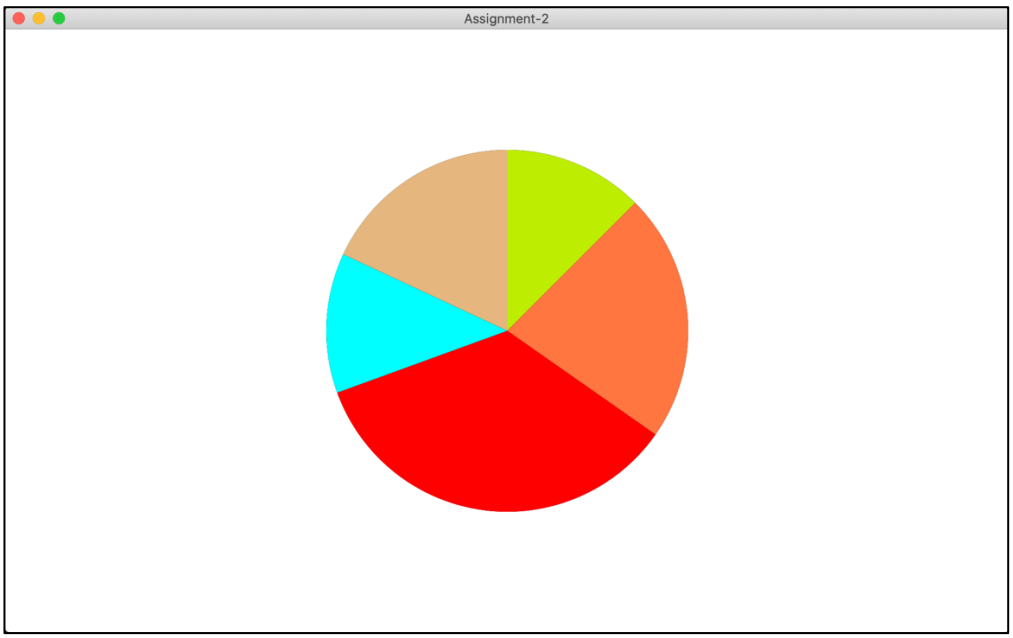
```
MyRectangle b_arc_rec = b_arc.getMyBoundingRectangle();
b_arc_rec.color = MyColor.CORAL;
b_arc_rec.draw(graphicsContext1);
b_arc.draw(graphicsContext1);

Stage stage1 = new Stage();
root1.getChildren().add(canvas1);
stage1.setScene(scene1);
stage1.setTitle("Assignment-2 part-2");
stage1.show();

// 3rd part
StackPane root2 = new StackPane();
Scene scene2 = new Scene(root2, w, h);
Canvas canvas2 = new Canvas(w, h);
GraphicsContext graphicsContext2 = canvas2.getGraphicsContext2D();

MyRectangle rec1 = new MyRectangle( x: 200,  y: 200, MyColor.BURLYWOOD,  h: 200,  w: 200);
MyRectangle rec2 = new MyRectangle( x: 350,  y: 300, MyColor.CORAL,  h: 200,  w: 300);
rec1.draw(graphicsContext2);
rec2.draw(graphicsContext2);

MyRectangle rec3 = new MyRectangle( x: 700,  y: 250, MyColor.BROWN,  h: 200,  w: 250);
MyCircle circ1 = new MyCircle( x: 830,  y: 300,  r: 150, MyColor.AQUAMARINE);
rec3.draw(graphicsContext2);
circ1.draw(graphicsContext2);

Stage stage2 = new Stage();
Canvas canvas3 = rec1.drawIntersectMyShape(rec1, rec2);
Canvas canvas4 = rec3.drawIntersectMyShape(rec3, circ1);
root2.getChildren().add(canvas2);
root2.getChildren().add(canvas3);
root2.getChildren().add(canvas4);
stage2.setScene(scene2);
stage2.setTitle("Assignment-2 part-3");
```

```
        stage2.show();

        System.out.println(rec1.toString());
        System.out.println(pizza.toString());
        System.out.println(slice3.toString());
        System.out.println(b_oval.toString());
        System.out.println(circ1.toString());
        System.out.println(b_line.toString());
    }

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

Screenshots of Output:

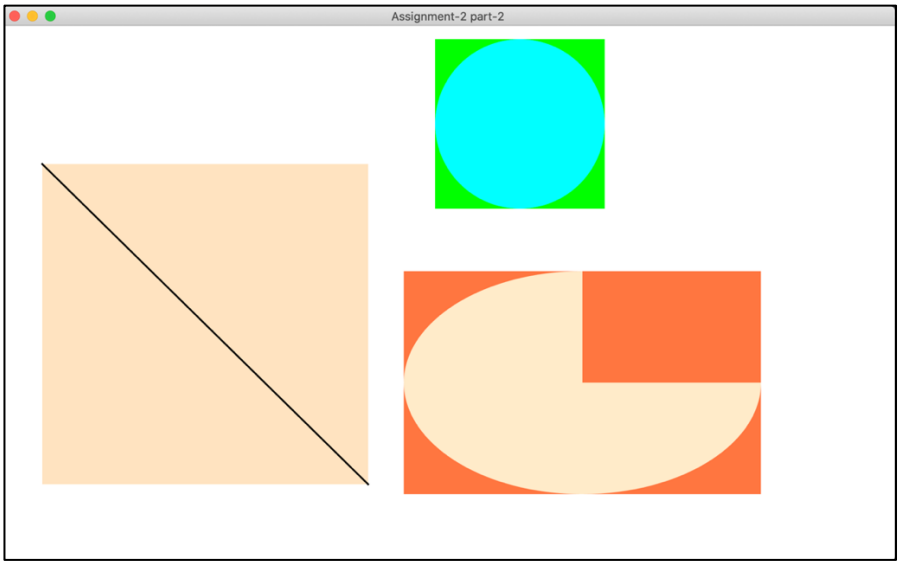
- **JavaFX Window 1:**

  Here, a geometric configuration of a circular pizza pie with some arbitrarily sized pies have been drawn using MyCircle and MyArc class. The height and width of the canvas are stored in variable h and w, and the size of the shapes are proportional to smallest of the canvas.



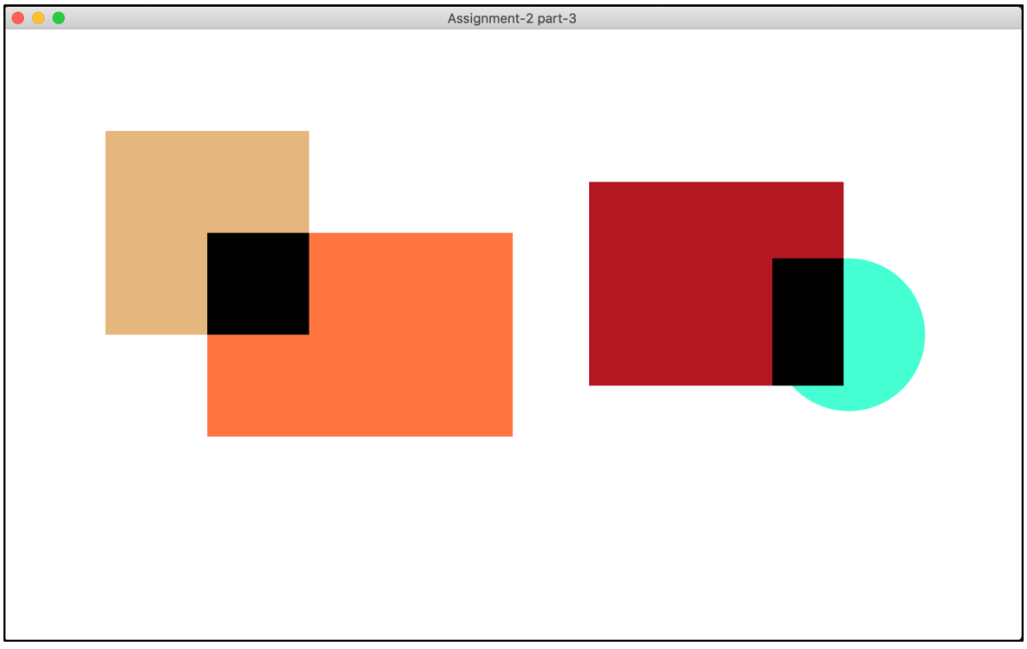
- **JavaFX Window 2:**

  In this stage, an instance of MyLine, MyCircle and MyArc has been drawn and then their bounding rectangle has been calculated and drawn.

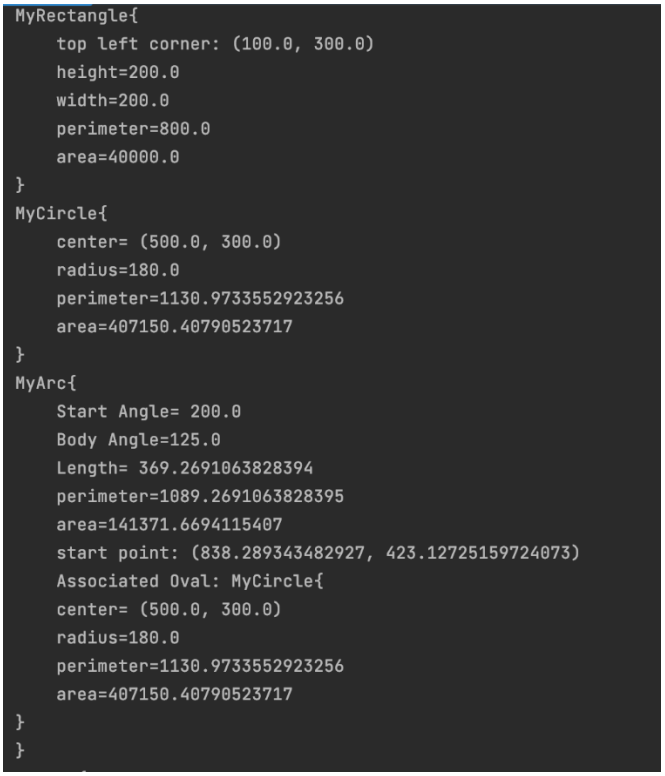


- **JavaFX Window 3:**

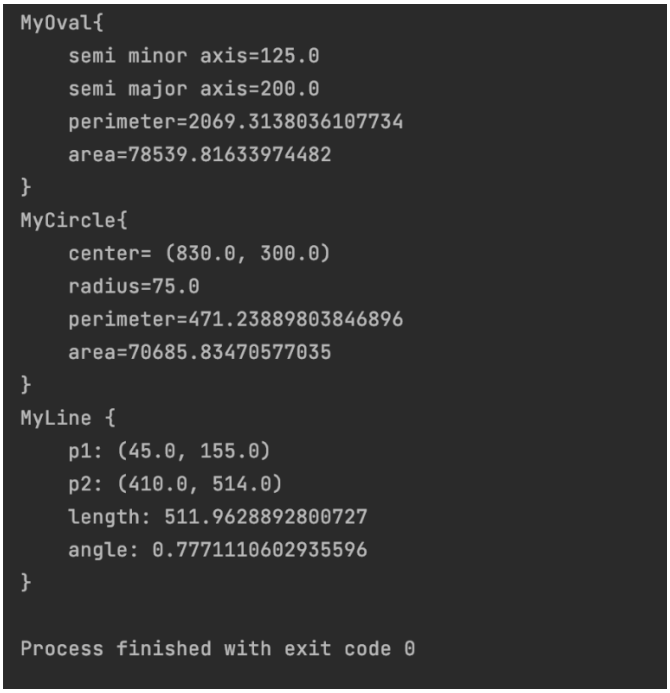
  In this stage, there are three MyRectangle objects rec1, rec2, rec3 and a MyCircle object circ1. Intersection of rec1 and rec2 has been drawn in left side and intersection of rec3 and circ1 has been drawn in right.



- **Console**:
  toString of some of the objects created have been printed in console for testing purposes.



```
MyRectangle{
    top left corner: (100.0, 300.0)
    height=200.0
    width=200.0
    perimeter=800.0
    area=40000.0
}
MyCircle{
    center= (500.0, 300.0)
    radius=180.0
    perimeter=1130.9733552923256
    area=407150.40790523717
}
MyArc{
    Start Angle= 200.0
    Body Angle=125.0
    Length= 369.2691063828394
    perimeter=1089.2691063828395
    area=141371.6694115407
    start point: (838.289343482927, 423.12725159724073)
    Associated Oval: MyCircle{
    center= (500.0, 300.0)
    radius=180.0
    perimeter=1130.9733552923256
    area=407150.40790523717
}
}
```



```
MyOval{
    semi minor axis=125.0
    semi major axis=200.0
    perimeter=2069.3138036107734
    area=78539.81633974482
}
MyCircle{
    center= (830.0, 300.0)
    radius=75.0
    perimeter=471.23889803846896
    area=70685.83470577035
}
MyLine {
    p1: (45.0, 155.0)
    p2: (410.0, 514.0)
    length: 511.9628892800727
    angle: 0.7771110602935596
}

Process finished with exit code 0
```