Fall 2022 | CSC 22100 | Assignment 2

Alhamza Muswara

# Problem/Task:

1. Create a hierarchy of Java classes as follows: “MyArc” is a “MyShape”; “MyCircle” is a “MyOval”.
2. Create an Interface called MyShapeInterface which is to be implemented by the abstract class MyShape. MyShapeInterface methods include:
    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. SimilarObjects— static method returns true if two MyShape objects S1 and S2 are similar;
    d. intersectMyRectangles — static method returns the intersection of two MyRectangles objects R1 and R2 if they do overlap, and null otherwise.
    e. intersectMyShapes — static method returns the set of all points on or within the boundary of the area of intersection of two MyShape objects S1 and S2 if they do overlap, and null otherwise.
    f. drawIntersectMyShapes — default method returns a canvas with a drawing of the intersection of two objects in the class hierarchy if they do overlap.
3. 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 pizza slices 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.

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

## [2] Solution Methods

**MyArc is a subclass of MyShape which draws an arc, or similar to a portion of an oval's perimeter. Uses same strokeArc() and fillArc() to draw the arc line and color fill. Length() method uses algebra to find length of arc. The same goes for pointInMyShape() method. getMyBoundingRectangle() method is necessary to find the resulting shape of two intersecting shapes by returning the bounding intersection rectangle.**

### MyArc:

```
public class MyArc extends MyShape {
	MyPoint p1, p2;
	private double width;
	private double height;
	private double startAngle;
	private double arcExtent;
	private ArcType closuretype;

	MyArc(double x, double y, double w, double h, double angle1, double angle2,
ArcType closure) {
		p.setX(x);
		p.setY(y);
		this.width = w;
		this.height = h;
		this.startAngle = angle1;
		this.arcExtent = angle2 - angle1;
		this.closuretype = closure;
	}

	// approximate of arc length
	double length() {
		double r1 = (width / 2) * (height / 2) / (Math.sqrt(
				Math.pow((width / 2) * Math.sin(startAngle), 2) +
Math.pow((width / 2) * Math.sin(startAngle), 2)));
		double r2 = (width / 2) * (height / 2) / (Math.sqrt(Math.pow((width / 2)
* Math.sin(startAngle + arcExtent), 2)
				+ Math.pow((width / 2) * Math.sin(startAngle + arcExtent),
2)));
		double averageR = (r1 + r2) / 2;
		return ((Math.PI) / 180) * arcExtent * averageR;
	}

	@Override
	double getPerimeter() {
		return length();
	}

	@Override
	double getArea() {
		return 1;
	}

	@Override
```

```
	public String toString() {
		return "Arc[x=" + p.getX() + ", y=" + p.getY() + ", width=" + width + ",
height=" + height + ", fill="
				+ fill.toString() + ", length=" + (2 * width + 2 * height)
+ "]";
	}

	@Override
	void draw(GraphicsContext gc) {
		gc.strokeArc(p.getX(), p.getY(), width, height, startAngle, arcExtent,
closuretype);
		gc.setFill(fill.myColor());
		gc.fillArc(p.getX(), p.getY(), width, height, startAngle, arcExtent,
closuretype);
	}

	@Override
	double getWidth() {
		return width;
	}

	@Override
	double getHeight() {
		return height;
	}

	@Override
	public boolean pointInMyShape(double x, double y) {
		if ((Math.pow((x - p.getX() + (width / 2)), 2) / (Math.pow(width / 2,
2)))
				+ (Math.pow((y - p.getY() + (height / 2)), 2) /
(Math.pow(height / 2, 2))) <= 1) {
			if (x > p.getX() + (width / 2) && y < p.getY() - (height / 2)) {
				if (Math.atan(((p.getY() + (height / 2)) - y) / (x -
(p.getX() + (width / 2)))) >= startAngle
						&& Math.atan(((p.getY() + (height / 2)) - y)
/ (x - (p.getX() + (width / 2)))) <= startAngle
								+ arcExtent) {
					return true;
				}
			}
			if (x < p.getX() + (width / 2) && y < p.getY() - (height / 2)) {
				if (Math.atan(((p.getX() + (width / 2)) -x) / ((p.getY() +
(height / 2)) - y)) + 90 >= startAngle
						&& Math.atan((p.getX() + (width / 2)-x) /
((p.getY() + (height / 2)) - y)) + 90 <= startAngle
								+ arcExtent) {
					return true;
				}
			}
			if (x < p.getX() + (width / 2) && y > p.getY() - (height / 2)) {
				if (Math.atan((y - (p.getY() + (height / 2))) / ((p.getX()
+ (width / 2)-x)))+180 >= startAngle
						&& Math.atan((y-(p.getY() + (height / 2))) /
((p.getX() + (width / 2)-x)))+180 <= startAngle
```

```
                                                        + arcExtent) {
                                    return true;
                              }
                        }

                        if (x > p.getX() + (width / 2) && y > p.getY() - (height / 2)) {
                              if (Math.atan((x-(p.getX() + (width / 2))) / (y-(p.getY()
+ (height / 2)) )) + 270 >= startAngle
                                          && Math.atan(x-(p.getX() + (width / 2)) / (y-
(p.getY() + (height / 2)))) + 270 <= startAngle
                                                        + arcExtent) {
                                    return true;
                              }
                        }
                        if(x == p.getX() + (width / 2)&&y < p.getY() - (height / 2)) {
                              if(90>=startAngle&&90<=startAngle+arcExtent) {
                                    return true;
                              }
                        }
                        if(x == p.getX() + (width / 2)&&y > p.getY() - (height / 2)) {
                              if(270>=startAngle&&270<=startAngle+arcExtent) {
                                    return true;
                              }
                        }
                        if(x>p.getX() + (width / 2)&&y == p.getY() - (height / 2)) {
                              if(0>=startAngle&&0<=startAngle+arcExtent) {
                                    return true;
                              }
                        }
                        if(x<p.getX() + (width / 2)&&y == p.getY() - (height / 2)) {
                              if(180>=startAngle&&180<=startAngle+arcExtent) {
                                    return true;
                              }
                        }

                  }
                  return false;
            }

            @Override
            public MyRectangle getMyBoundingRectangle() {
                  MyRectangle S1 = new MyRectangle(p.getX(), p.getY(), getWidth(),
getHeight());
                  return S1;
            }

}
```

**MyCircle class is A subclass of MyOval class. pointInMyShape() method returns a Boolean value to see if a point is in the circle.**

## MyCircle:

```
public class MyCircle extends MyOval {

	MyCircle(double x, double y, double w, double h) {
		super(x, y, w, w);

	}
	@Override
	public String toString() {
		return "MyCircle";
	}

	public boolean pointInMyShape(double x, double y) {

		if (Math.sqrt(Math.pow((x - (p.getX() + (getWidth() / 2))), 2)
				+ Math.pow((y - (p.getX() + (getWidth() / 2))), 2)) <=
(getWidth() / 2)) {
			return true;
		}
		return false;

	}

}
```

**The MyShapeInterface interface is used to ensure that my shape class implements the methods described in the interface. The methods in the interface will be used to create shapes made through two intersecting shape objects. Abstract methods are overridden in each subclass of MyShape in reference to the interface.**

## MyShapeInterface:

```
public interface MyShapeInterface {

	abstract MyRectangle getMyBoundingRectangle();

	static MyRectangle intersectMyRectangles(MyRectangle R1, MyRectangle R2) {
		double x = 0, y = 0, w = 0, h = 0;
		// if rectangle has area 0, no overlap
		if (R1.getWidth() == 0 || R1.getHeight() == 0 || R2.getWidth() == 0 ||
R2.getHeight() == 0)
			return null;

		// If one rectangle is on left side of other
		if (R1.p.getX() > R2.p.getX() + R2.getWidth() || R2.p.getX() >
R1.p.getX() + R1.getWidth()) {
			return null;
		}

		// If one rectangle is above other
		if (R1.p.getY() > R2.p.getY() + R2.getHeight() || R2.p.getY() >
R1.p.getY() + R1.getHeight()) {
			return null;
		}

		if (R1.p.getY() >= R2.p.getY())
			y = R1.p.getY();
		else
			y = R2.p.getY();

		if (R1.p.getX() >= R2.p.getX())
			x = R1.p.getX();
		else
			x = R2.p.getX();

		if ((R1.p.getY() + R1.getHeight()) >= (R2.p.getY() + R2.getHeight()))
			h = R2.p.getY() + R2.getHeight() - y;
		else
			h = R1.p.getY() + R1.getHeight() - y;

		if ((R1.p.getX() + R1.getWidth()) >= (R2.p.getX() + R2.getWidth()))
			w = R2.p.getX() + R2.getWidth() - x;
		else
			w = R1.p.getX() + R1.getWidth() - x;

		MyRectangle R3 = new MyRectangle(x, y, w, h);
```

```
		return R3;

	}

	static boolean similarObjects(MyShape S1, MyShape S2) {
		if (S1.toString() == S2.toString() && S1.getWidth() == S2.getWidth() &&
S1.getHeight() == S2.getHeight()) {
			// System.out.print(S1.toString());
			return true;
		}

		// System.out.print(S1.toString());
		return false;
	}

	abstract boolean pointInMyShape(double x, double y);

	static ArrayList<MyPoint> intersectMyShapes(MyRectangle R, MyShape S1, MyShape
S2) {
		ArrayList<MyPoint> List = new ArrayList<MyPoint>();

		for (double i = R.p.getY(); i <= R.p.getY() + R.getHeight(); i++) {
			for (double j = R.p.getX(); j <= R.p.getX() + R.getWidth(); j++)
{
				if (S1.pointInMyShape(j, i) && S2.pointInMyShape(j, i)) {
					List.add(new MyPoint(j, i));
					// System.out.print(i+" "+j+"\n");
				}
			}
		}

		return List;
	}

	static Canvas drawIntersectMyShapes(MyRectangle R, ArrayList<MyPoint> L,
MyColor color) {

		Canvas canvas = new Canvas(R.p.getX() + R.getWidth(), R.p.getY() +
R.getHeight());
		GraphicsContext gc = canvas.getGraphicsContext2D();
		gc.setFill(color.myColor());
		for (int i = 0; i < L.size() - 1; i++) {
			gc.fillRect(L.get(i).getX(), L.get(i).getY(), 1, 1);
		}
		return canvas;
	}
}
```

**The abstract MyShape Clash differs from the previous assignment in that it behaves similar to an interface. All methods were overridden in subclass in the respective to its shape.**

## MyShape(abstract)

```
public abstract class MyShape implements MyShapeInterface {
	// MyPoint,MyColor
	MyPoint p = new MyPoint(0, 0);
	MyColor fill = MyColor.BLACK;

	// area, parameter

	abstract double getArea();

	void draw() {

	}

	abstract double getWidth();

	abstract double getHeight();

	abstract double getPerimeter();

	// toString

	public String toString() {
		return "";
	}

	// draw
	abstract void draw(GraphicsContext gc);

	public abstract boolean pointInMyShape(double x, double y);

	public abstract MyRectangle getMyBoundingRectangle();
}
```

## Main:

**The following main class depicts two overlapping rectangles and it's intersecting shape, as well as an overlapping circle and rectangle and its intersecting shape. the main function also depicts a circle composed of different arts filled with different color.**

```
public class Main extends Application {
	@Override
	public void start(Stage primaryStage) {
		primaryStage.setTitle("Assignment 2");
		Group root = new Group();

		// CHANGE CANVAS VARAIBLES
		double dx = 700;
		double dy = 700;
		// ^

		Canvas canvas = new Canvas(dx, dy);
		GraphicsContext gc = canvas.getGraphicsContext2D();

		//bounding rectangles and intersection shapes
		gc.setLineWidth(1);

		MyRectangle R1 = new MyRectangle(0, 0, 50, 50);

		R1.fill=MyColor.DARKORCHID;
		MyCircle C1 = new MyCircle(0, 0, 100, 100);

		MyRectangle R2 = R1.getMyBoundingRectangle();
		MyRectangle R3 = C1.getMyBoundingRectangle();
		R2.fill = MyColor.BLUE;
		R3.fill = MyColor.DARKKHAKI;

		R2.draw(gc);
		R3.draw(gc);

		MyRectangle R4 = MyShapeInterface.intersectMyRectangles(R2, R3);

		C1.draw(gc);
		R1.draw(gc);

		ArrayList<MyPoint> List = new ArrayList<MyPoint>();
		List=MyShapeInterface.intersectMyShapes(R4, R1, C1);
		//new Canvas with intersection shape drawn
		Canvas canvas2 = MyShapeInterface.drawIntersectMyShapes(R4, List,
MyColor.RED);



		//two rectangles
		MyRectangle R5=new MyRectangle(150,150, 100, 100);
		MyRectangle R6=new MyRectangle(125, 125, 50, 75);
```

```
                MyRectangle R7 = R5.getMyBoundingRectangle();
                MyRectangle R8 = R6.getMyBoundingRectangle();

                R7.fill = MyColor.BLUE;
                R8.fill = MyColor.DARKKHAKI;

                R7.draw(gc);
                R8.draw(gc);

                MyRectangle R9 = MyShapeInterface.intersectMyRectangles(R7, R8);

                ArrayList<MyPoint> List2 = new ArrayList<MyPoint>();
                List2=MyShapeInterface.intersectMyShapes(R9, R5, R6);
                //new Canvas with intersection shape drawn
                Canvas canvas3 = MyShapeInterface.drawIntersectMyShapes(R9, List2,
MyColor.RED);




                /////pie graph

                MyArc arc1= new MyArc(300, 300, 150, 150, 0, 50, ArcType.ROUND);
                MyArc arc2= new MyArc(300, 300, 150, 150, 50, 100, ArcType.ROUND);
                MyArc arc3= new MyArc(300, 300, 150, 150, 100, 140, ArcType.ROUND);
                MyArc arc4= new MyArc(300, 300, 150, 150, 140, 180, ArcType.ROUND);
                MyArc arc5= new MyArc(300, 300, 150, 150, 180, 250, ArcType.ROUND);
                MyArc arc6= new MyArc(300, 300, 150, 150, 250, 360, ArcType.ROUND);


                arc2.fill=MyColor.RED;
                arc3.fill=MyColor.BLUE;
                arc4.fill=MyColor.DARKKHAKI;
                arc5.fill=MyColor.DARKORANGE;
                arc6.fill=MyColor.GREEN;

                arc1.draw(gc);
                arc2.draw(gc);
                arc3.draw(gc);
                arc4.draw(gc);
                arc5.draw(gc);
                arc6.draw(gc);


                root.getChildren().add(canvas);
                root.getChildren().add(canvas2);
                root.getChildren().add(canvas3);
                primaryStage.setScene(new Scene(root));
                primaryStage.show();
```

```
        }
}
```

# [3] Codes Developed

## Classes Imported

```
import java.lang.Math;
import java.util.ArrayList;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;

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

## Enum MyColor:

```
package application;

import javafx.scene.paint.Color;

public enum MyColor {
	RED(255, 0, 0, 1),
	GREEN(0, 255, 0, 1),
	BLUE(0, 0, 255, 1),
	BLACK(0, 0, 0, 1),
	DARKKHAKI(189, 183, 107, 1),
	DARKORANGE(255, 140, 0, 1),
	DARKORCHID(99,32,204,1);

	private int red;
	private int green;
	private int blue;
	private double opacity;

	MyColor(int red, int green, int blue, double opacity) {
		this.red = red;
		this.blue = blue;
		this.green = green;
		this.opacity = opacity;
	}

	public Color myColor() {
		return Color.rgb(red, green, blue, opacity);
	}

	public String toString() {
		return String.format("#%02x%02x%02x", red, green, blue);
```

```
	}
}
```

## Class MyPoint:

```
package application;

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

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

	void setX(double x) {
		this.x=x;
	}

	void setY(double y) {
		this.y=y;
	}

	double getX() {
		return x;
	}
	double getY() {
		return y;
	}

	void shiftX(double x) {
	    this.x+=x;
	}
	void shiftY(double y) {
		this.y+=y;
	}
	public String toString() {
		return "MyPoint [x = "+x+", y = "+y+"]";
	}


	//distance from P(x,y) to some other point x2,y2
	void getDistance(double x2, double y2){
		System.out.println(Math.sqrt(Math.pow(x2-this.x,2)+Math.pow(y2-
this.y,2)));
	}
	//angle of line from the +x counterclockwise
	void getAngle(double x2, double y2) {
		  double m=((y2-this.y)/(x2-this.x));
		  System.out.println(Math.atan(m));
	}

}
```

## Class MyShape:

```
package application;

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

public abstract class MyShape implements MyShapeInterface {
	// MyPoint,MyColor
	MyPoint p = new MyPoint(0, 0);
	MyColor fill = MyColor.BLACK;

	// area, parameter

	abstract double getArea();

	void draw() {

	}

	abstract double getWidth();

	abstract double getHeight();

	abstract double getPerimeter();

	// toString

	public String toString() {
		return "";
	}

	// draw
	abstract void draw(GraphicsContext gc);

	public abstract boolean pointInMyShape(double x, double y);

	public abstract MyRectangle getMyBoundingRectangle();

}
```

## Class MyRectangle:

```
package application;

import javafx.scene.canvas.GraphicsContext;

public class MyRectangle extends MyShape {
	private double width;
	private double height ;

	MyRectangle(double x, double y, double w, double h) {
		this.width = w;
		this.height = h;
		p.setX(x);
		p.setY(y);
	}

	@Override
	double getWidth() {
		return width;
	}

	@Override
	double getHeight() {
		return height;
	}

	@Override
	public String toString() {
		return "MyRectangle";
	}

	@Override
	double getArea() {
		// TODO
		return (width * height);
	}

	@Override
	double getPerimeter() {
		// TODO
		return (2 * width + 2 * height);
	}

	@Override
	void draw(GraphicsContext gc) {
		// TODO
		gc.strokeRect(p.getX(), p.getY(), this.width, this.height);
```

```
                gc.setFill(fill.myColor());
                gc.fillRect(p.getX(), p.getY(), width, height);
        }

        public boolean pointInMyShape(double x, double y) {

                if (x >= p.getX() && x <= p.getX() + this.width && y >= p.getY() && y <=
p.getY() + this.height) {
                        return true;
                }
                return false;

        }


        @Override
        public MyRectangle getMyBoundingRectangle() {
                MyRectangle S1= new
MyRectangle(p.getX(),p.getY(),getWidth(),getHeight());
                return S1;
        }

}
```

## Class MyOval:

```
package application;

import javafx.scene.canvas.GraphicsContext;

public class MyOval extends MyShape {
	private double width = 0;
	private double height = 0;

	// constructor
	MyOval(double x, double y, double w, double h) {
		p.setX(x);
		p.setY(y);
		this.width = w;
		this.height = h;
	}

	@Override
	double getWidth() {
		return width;
	}

	@Override
	double getHeight() {
		return height;
	}

	// area, perimeter
	@Override
	double getArea() {
		return (Math.PI * (width / 2) * (height / 2));
	}

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

	// getX, getY, getA, getB

	// getX(center)
	double getCenterX() {
		return p.getX() + (width / 2);
	}

	// getY,(center)
	double getCenterY() {
		return p.getY() + (height / 2);
	}
```

```
	// toString
	@Override
	public String toString() {
		return "MyOval";
	}

	@Override
	void draw(GraphicsContext gc) {
		gc.strokeOval(p.getX(), p.getY(), this.width, this.height);
		gc.setFill(fill.myColor());
		gc.fillOval(p.getX(), p.getY(), width, height);
	}

	@Override
	public boolean pointInMyShape(double x, double y) {
		if ((Math.pow((x - p.getX() + (width / 2)),2) / (Math.pow(width / 2,
2)))
				+ (Math.pow((y - p.getY() + (height / 2)), 2) / (Math.pow(
height / 2, 2))) <= 1) {
			return true;
		}
		return false;
	}

	@Override
	public MyRectangle getMyBoundingRectangle() {
		MyRectangle S1= new
MyRectangle(p.getX(),p.getY(),getWidth(),getHeight());
		return S1;
	}
}
```

## Class MyArc:

```
package application;

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

public class MyArc extends MyShape {
	MyPoint p1, p2;
	private double width;
	private double height;
	private double startAngle;
	private double arcExtent;
	private ArcType closuretype;

	MyArc(double x, double y, double w, double h, double angle1, double angle2,
ArcType closure) {
		p.setX(x);
		p.setY(y);
		this.width = w;
		this.height = h;
		this.startAngle = angle1;
		this.arcExtent = angle2 - angle1;
		this.closuretype = closure;
	}

	// approximate of arc length
	double length() {
		double r1 = (width / 2) * (height / 2) / (Math.sqrt(
				Math.pow((width / 2) * Math.sin(startAngle), 2) +
Math.pow((width / 2) * Math.sin(startAngle), 2)));
		double r2 = (width / 2) * (height / 2) / (Math.sqrt(Math.pow((width / 2)
* Math.sin(startAngle + arcExtent), 2)
				+ Math.pow((width / 2) * Math.sin(startAngle + arcExtent),
2)));
		double averageR = (r1 + r2) / 2;
		return ((Math.PI) / 180) * arcExtent * averageR;
	}

	@Override
	double getPerimeter() {
		return length();
	}

	@Override
	double getArea() {
		return 1;
	}

	@Override
	public String toString() {
		return "Arc[x=" + p.getX() + ", y=" + p.getY() + ", width=" + width + ",
height=" + height + ", fill="
```

```
                        + fill.toString() + ", length=" + (2 * width + 2 * height)
+ "]";
	}

	@Override
	void draw(GraphicsContext gc) {
		gc.strokeArc(p.getX(), p.getY(), width, height, startAngle, arcExtent,
closuretype);
		gc.setFill(fill.myColor());
		gc.fillArc(p.getX(), p.getY(), width, height, startAngle, arcExtent,
closuretype);
	}

	@Override
	double getWidth() {
		return width;
	}

	@Override
	double getHeight() {
		return height;
	}

	@Override
	public boolean pointInMyShape(double x, double y) {
		if ((Math.pow((x - p.getX() + (width / 2)), 2) / (Math.pow(width / 2,
2)))
				+ (Math.pow((y - p.getY() + (height / 2)), 2) /
(Math.pow(height / 2, 2))) <= 1) {
			if (x > p.getX() + (width / 2) && y < p.getY() - (height / 2)) {
				if (Math.atan(((p.getY() + (height / 2)) - y) / (x -
(p.getX() + (width / 2)))) >= startAngle
						&& Math.atan(((p.getY() + (height / 2)) - y)
/ (x - (p.getX() + (width / 2)))) <= startAngle
								+ arcExtent) {
					return true;
				}
			}
			if (x < p.getX() + (width / 2) && y < p.getY() - (height / 2)) {
				if (Math.atan(((p.getX() + (width / 2)) - x) / ((p.getY()
+ (height / 2)) - y)) + 90 >= startAngle
						&& Math.atan((p.getX() + (width / 2) - x) /
((p.getY() + (height / 2)) - y)) + 90 <= startAngle
								+ arcExtent) {
					return true;
				}
			}
			if (x < p.getX() + (width / 2) && y > p.getY() - (height / 2)) {
				if (Math.atan((y - (p.getY() + (height / 2))) / ((p.getX()
+ (width / 2) - x))) + 180 >= startAngle
						&& Math.atan((y - (p.getY() + (height / 2)))
/ ((p.getX() + (width / 2) - x)))
								+ 180 <= startAngle + arcExtent)
{
					return true;
```

```
                        }
                    }

                    if (x > p.getX() + (width / 2) && y > p.getY() - (height / 2)) {
                        if (Math.atan((x - (p.getX() + (width / 2))) / (y -
(p.getY() + (height / 2)))) + 270 >= startAngle
                                && Math.atan(x - (p.getX() + (width / 2)) /
(y - (p.getY() + (height / 2)))) + 270 <= startAngle
                                                + arcExtent) {
                            return true;
                        }
                    }
                    if (x == p.getX() + (width / 2) && y < p.getY() - (height / 2)) {
                        if (90 >= startAngle && 90 <= startAngle + arcExtent) {
                            return true;
                        }
                    }
                    if (x == p.getX() + (width / 2) && y > p.getY() - (height / 2)) {
                        if (270 >= startAngle && 270 <= startAngle + arcExtent) {
                            return true;
                        }
                    }
                    if (x > p.getX() + (width / 2) && y == p.getY() - (height / 2)) {
                        if (0 >= startAngle && 0 <= startAngle + arcExtent) {
                            return true;
                        }
                    }
                    if (x < p.getX() + (width / 2) && y == p.getY() - (height / 2)) {
                        if (180 >= startAngle && 180 <= startAngle + arcExtent) {
                            return true;
                        }
                    }

                }
                return false;
        }

        @Override
        public MyRectangle getMyBoundingRectangle() {
                MyRectangle S1 = new MyRectangle(p.getX(), p.getY(), getWidth(),
getHeight());
                return S1;
        }

}
```

## Class MyCircle:

```
package application;

public class MyCircle extends MyOval {

	MyCircle(double x, double y, double w, double h) {
		super(x, y, w, w);

	}
	@Override
	public String toString() {
		return "MyCircle";
	}

	public boolean pointInMyShape(double x, double y) {

		if (Math.sqrt(Math.pow((x - (p.getX() + (getWidth() / 2))), 2)
				+ Math.pow((y - (p.getX() + (getWidth() / 2))), 2)) <=
(getWidth() / 2)) {
			return true;
		}
		return false;

	}

}
```

## Interface MyShapeInterface:

```
package application;

import java.util.ArrayList;


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


public interface MyShapeInterface {

	abstract MyRectangle getMyBoundingRectangle();

	static MyRectangle intersectMyRectangles(MyRectangle R1, MyRectangle R2) {
		double x = 0, y = 0, w = 0, h = 0;
		// if rectangle has area 0, no overlap
		if (R1.getWidth() == 0 || R1.getHeight() == 0 || R2.getWidth() == 0 ||
R2.getHeight() == 0)
			return null;

		// If one rectangle is on left side of other
		if (R1.p.getX() > R2.p.getX() + R2.getWidth() || R2.p.getX() >
R1.p.getX() + R1.getWidth()) {
			return null;
		}

		// If one rectangle is above other
		if (R1.p.getY() > R2.p.getY() + R2.getHeight() || R2.p.getY() >
R1.p.getY() + R1.getHeight()) {
			return null;
		}

		if (R1.p.getY() >= R2.p.getY())
			y = R1.p.getY();
		else
			y = R2.p.getY();

		if (R1.p.getX() >= R2.p.getX())
			x = R1.p.getX();
		else
			x = R2.p.getX();

		if ((R1.p.getY() + R1.getHeight()) >= (R2.p.getY() + R2.getHeight()))
			h = R2.p.getY() + R2.getHeight() - y;
		else
			h = R1.p.getY() + R1.getHeight() - y;

		if ((R1.p.getX() + R1.getWidth()) >= (R2.p.getX() + R2.getWidth()))
			w = R2.p.getX() + R2.getWidth() - x;
		else
			w = R1.p.getX() + R1.getWidth() - x;
```

```
            MyRectangle R3 = new MyRectangle(x, y, w, h);
            return R3;

      }

      static boolean similarObjects(MyShape S1, MyShape S2) {
            if (S1.toString() == S2.toString() && S1.getWidth() == S2.getWidth() &&
S1.getHeight() == S2.getHeight()) {
                  // System.out.print(S1.toString());
                  return true;
            }

            // System.out.print(S1.toString());
            return false;
      }

      abstract boolean pointInMyShape(double x, double y);

      static ArrayList<MyPoint> intersectMyShapes(MyRectangle R, MyShape S1, MyShape
S2) {
            ArrayList<MyPoint> List = new ArrayList<MyPoint>();

            for (double i = R.p.getY(); i <= R.p.getY() + R.getHeight(); i++) {
                  for (double j = R.p.getX(); j <= R.p.getX() + R.getWidth(); j++)
{
                        if (S1.pointInMyShape(j, i) && S2.pointInMyShape(j, i)) {
                              List.add(new MyPoint(j, i));
                              // System.out.print(i+" "+j+"\n");
                        }
                  }
            }

            return List;
      }

      static Canvas drawIntersectMyShapes(MyRectangle R, ArrayList<MyPoint> L,
MyColor color) {

            Canvas canvas = new Canvas(R.p.getX() + R.getWidth(), R.p.getY() +
R.getHeight());
            GraphicsContext gc = canvas.getGraphicsContext2D();
            gc.setFill(color.myColor());
            for (int i = 0; i < L.size() - 1; i++) {
                  gc.fillRect(L.get(i).getX(), L.get(i).getY(), 1, 1);
            }
            return canvas;
      }
}
```

## Class Main:

```
package application;

import java.lang.Math;
import java.util.ArrayList;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;


public class Main extends Application {
	@Override
	public void start(Stage primaryStage) {
		primaryStage.setTitle("Assignment 2");
		Group root = new Group();

		// CHANGE CANVAS VARAIBLES
		double dx = 700;
		double dy = 700;
		// ^

		Canvas canvas = new Canvas(dx, dy);
		GraphicsContext gc = canvas.getGraphicsContext2D();

		//
		gc.setLineWidth(1);

		MyRectangle R1 = new MyRectangle(0, 0, 50, 50);

		R1.fill=MyColor.DARKORCHID;
		MyCircle C1 = new MyCircle(0, 0, 100, 100);

		MyRectangle R2 = R1.getMyBoundingRectangle();
		MyRectangle R3 = C1.getMyBoundingRectangle();
		R2.fill = MyColor.BLUE;
		R3.fill = MyColor.DARKKHAKI;

		R2.draw(gc);
		R3.draw(gc);

		MyRectangle R4 = MyShapeInterface.intersectMyRectangles(R2, R3);

		C1.draw(gc);
		R1.draw(gc);

		ArrayList<MyPoint> List = new ArrayList<MyPoint>();
		List=MyShapeInterface.intersectMyShapes(R4, R1, C1);
		//new Canvas with intersection shape drawn
```

```
            Canvas canvas2 = MyShapeInterface.drawIntersectMyShapes(R4, List,
MyColor.RED);



            //two rectangles
            MyRectangle R5=new MyRectangle(150,150, 100, 100);
            MyRectangle R6=new MyRectangle(125, 125, 50, 75);


            MyRectangle R7 = R5.getMyBoundingRectangle();
            MyRectangle R8 = R6.getMyBoundingRectangle();

            R7.fill = MyColor.BLUE;
            R8.fill = MyColor.DARKKHAKI;

            R7.draw(gc);
            R8.draw(gc);

            MyRectangle R9 = MyShapeInterface.intersectMyRectangles(R7, R8);

            ArrayList<MyPoint> List2 = new ArrayList<MyPoint>();
            List2=MyShapeInterface.intersectMyShapes(R9, R5, R6);
            //new Canvas with intersection shape drawn
            Canvas canvas3 = MyShapeInterface.drawIntersectMyShapes(R9, List2,
MyColor.RED);




            /////pie graph

            MyArc arc1= new MyArc(300, 300, 150, 150, 0, 50, ArcType.ROUND);
            MyArc arc2= new MyArc(300, 300, 150, 150, 50, 100, ArcType.ROUND);
            MyArc arc3= new MyArc(300, 300, 150, 150, 100, 140, ArcType.ROUND);
            MyArc arc4= new MyArc(300, 300, 150, 150, 140, 180, ArcType.ROUND);
            MyArc arc5= new MyArc(300, 300, 150, 150, 180, 250, ArcType.ROUND);
            MyArc arc6= new MyArc(300, 300, 150, 150, 250, 360, ArcType.ROUND);


            arc2.fill=MyColor.RED;
            arc3.fill=MyColor.BLUE;
            arc4.fill=MyColor.DARKKHAKI;
            arc5.fill=MyColor.DARKORANGE;
            arc6.fill=MyColor.GREEN;

            arc1.draw(gc);
            arc2.draw(gc);
            arc3.draw(gc);
            arc4.draw(gc);
            arc5.draw(gc);
            arc6.draw(gc);
```

```
                root.getChildren().add(canvas);
                root.getChildren().add(canvas2);
                root.getChildren().add(canvas3);
                primaryStage.setScene(new Scene(root));
                primaryStage.show();


        }
}
```

## [4] Outputs produced for the tasks indicated