SOFTWARE DESIGN LABORATORY
Assignment 3

Josh Miranda

Hesham Auda

Section P

21 April 2021

1. Problem

Amend the **MyShape** class hierarchy in Assignment 2 as follows:

MyArc is a **MyShape**;

The objective of this assignment is to create a pie chart that will display the probabilities of how many n times a letter is shown in a passage. From assignment 2 we will use the MyShape class which MyArc inherits from. The MyArc object is a segment of the MyOval object that was also used in the previous assignment. There will also be a MyPieChart class that uses the class Slice. The Slice will show the individual probabilities of each letter shown in the pie. The class HistogramAlphaBet will calculate the n most frequent alphabet characters shown in the text file “Alice in Wonderland”. This data will be given to the Slices making a whole pie of probabilities. JavaFX graphics will be used to display all the different shapes to create the pie chart. For methods reused in this project please refer to the previous assignments.

2. Solution Methods

Class MyArc

The first method that will be talked about is **Class MyArc**. The MyArc object is filled with any color from the MyColor enum reference type that was also reused from the previous assignment. MyArc will have toString which returns a string representation of the MyArc object and then a draw method drawing the object.

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

The two packages that will be imported for this class will be javafx.scene.canvas.GraphicsContext which allows us to draw shapes into the canvas. Graphics context for this class allows us to draw the arc. The next package javafx.scene.shape.ArcType allows us to create the arc.

```
public class MyArc extends MyShape {
    double x,y,radius, startAngle, centralAngle;
    public MyColor color;
```

Here is the public class MyArc that extends to the MyShape class. The class **MyShape** is the hierarchy's superclass and allows other classes to inherit from it. Underneath is where the variable types are declared. The variables included are x, y, radius, startAngle and centralAngle declared as type double. Lastly is public MyColor color which is used to color the shapes drawn on the canvas.

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

This is the constructor used to initialize all the variables that we declared before. Within the parameters of public MyArc is double x, double y, double radius, double startAngle, double centralAngle and MyColor color. Below we **super x**, **y** and **color** which will be inheriting from class **MyShape.** After that we use the **this** attribute to guarantee that we are using the current values within the class. It will be used for all the variables we have declared and initialized.

```
public double getRadius() { return this.radius; }
public double getStartAngle() { return this.getStartAngle(); }
public double getCentralAngle() { return this.getCentralAngle(); }
public double getMyArea() { return (Math.PI * (Math.pow(radius, 2)) * (centralAngle/360)); }
public MyRectangle getMyBoundingRectangle(){
    return new MyRectangle(super.getX(), super.getY(), this.radius, this.radius, this.color);
}
```

These are the get methods used to return the current values to the variable. This is always used in the previous assignments, for example public double getRadius() will return the current value for radius with this.radius. This applies to getStartAngle() and getCentralAngle(). For getMyArea() it will return the area for an arc which is $\frac{\Theta}{360} * \prod * r^2$. The method getMyBoundingRectangle() is used in the previous assignment which returns a bounding rectangle that bounds to the arc that we created.

```
public String toString() {
    return "Radius:" + getRadius() + "\n Starting Angle: " + getStartAngle() +
            "\nCentral Angle " + getCentralAngle() + "\nArc Area:" + getMyArea();
}
@Override
public void draw(GraphicsContext gc) {
    gc.setFill(this.color.getColor());
    gc.strokeArc(this.x, this.y, this.radius, this.radius,  this.startAngle, this.centralAngle, ArcType.ROUND);
    gc.fillArc(this.x, this.y, this.radius, this.radius, this.startAngle, this.centralAngle, ArcType.ROUND);
}
```

The toString() method just returns the representation of the arc object with getRadius, getStartAngle(), getCentralAngle() and getMyArea(). Lastly we have the draw method using the GraphicsContext package imported earlier. Here there is gc.setFill() and gc.fillArc used to fill the arc drawn with color and the gc.strokeArc to draw the stroke. All within the parameters are the variables we declared and initialized above with the ArcType to help draw the arc on the canvas.

Class Slice

One of the only packages that will be imported is javafx.scene.canvas.GraphicsContext that will be used to draw the slice on to the canvas.

```
public class Slice extends MyArc{
    private double x, y ,radius,startAngle;
    private float angle;
    private MyColor color;
    public String key;
    public String value;

    public Slice(double x, double y, double radius, double startAngle, float angle, MyColor color){
        super(x, y, radius,startAngle, angle, color);
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.startAngle = startAngle;
        this.angle = angle;
        this.color = color;
    }
```

The Class Slice is similar to MyArc so we will be inheriting it through extending Slice to MyArc. Below are the variables being declared as double with x, y, radius and startAngle, float for angle, MyColor color for coloring the shapes and type String for key and value. The String key and value will be used for Hashmaps and explained in the Class HistogramAlphaBet section of this report. In the Slice constructor we are initializing all the variables that we declared previously. We are also taking the super of x, y, radius, startAngle. angle and color, this is to make calls to the parent class MyArc. Similar to all the other classes there are **this** attributes that set each of the values to the current class variables.

```
public double getRadius() { return radius; }
public double getStartAngle() { return startAngle; }
public float getAngle() { return angle; }
public MyColor getColor() { return color; }
public double area() { return 0.5 * angle * Math.pow(radius,2); }
```

These get methods return the current values to the variables as explained in other classes.

The public double area() is the area of the slice with the formula $\frac{1}{2}*\theta*r^2$.

```
public String toString(){
    return "Slice:" + "\nX:" + this.x + ", Y:" + this.y + "\nRadius:" + radius +
            "\nStarting Angle, Angle: (" + startAngle  + "," + angle +")," + color.getColor();
}

public void draw(GraphicsContext gc) {
    MyArc a1 = new MyArc(this.x, this.y, this.radius, this.startAngle, this.centralAngle, this.color);
    a1.draw(gc);
}
```

The toString() returns the String representation of the Slice object returning x, y, radius, startAngle, angle and color. Then the slice is drawn by creating an arc since they are similar, so we input the slice values into the arc shape to draw the arc on the canvas.

Class HistogramAlphaBet

The Class HistogramAlphaBet was one of the difficult classes to create as there are many things going on within.

```
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.text.Font;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import static java.util.Collections.reverseOrder;
```

javafx.scene.canvas.GraphicsContext just like in previous classes draws shapes onto the canvas.

javafx.scene.canvas.text.Font will print font onto the canvas similar to drawing shapes onto the canvas.

java.io.IOException an exception will occur if there was a failure or interruption.

java.nio.files.Files helps use files such as the “Alice in Wonderland” text file we are reading.

java.nio.file.Paths and java.nio.file.Path are basically the same thing as they return a path by converting it to a string or URL.

java.util.HashMap, java.util.list and java.util.Map allows the use of hashmaps, list and maps which will be used for counting the occurrences of letters in the text file we are reading.

java.util.stream.Collectors and java.util.Collections.reverseOrder are used for manipulations of hashmaps, lists and maps for things such as sorting or reversing.

```
public class HistogramAlphaBet {
    String data;
    HashMap<Character, Integer> charCountMap;
    float total = 0;
    List<Map.Entry<Character, Integer>> sorted_map;
    List<Map.Entry<Character, Float>> sorted_probabilitymap;
    HashMap<Character, Float> probability_map;
```

Here are the declaration of variables, hashmaps and list.

```
public HistogramAlphaBet() {
    Path path = Paths.get( first: "D:\\Users\\pug\\IdeaProjects\\project_3\\Alice in Wonderland.txt");
    try {
        this.data = new String(Files.readAllBytes(path));
    } catch (IOException e) {
        e.printStackTrace();
    }
    this.data = this.data.toLowerCase();
    this.data = this.data.replaceAll( regex: "[^a-zA-Z]", replacement: "");

    this.charCountMap = new HashMap<Character, Integer>();
    this.probability_map = new HashMap<Character, Float>();
    char[] strArray = this.data.toCharArray();
```

For HistogramAlphaBet() we begin by getting and reading the file. Within the parameter of Path.get() is the location of where the “Alice in Wonderland” text file is located. Next we create a try catch that will read all the bytes in the file, convert it to a String and then insert into the variable this.data. If there is an error the catch() will handle them. The next steps are just converting all the text to lowercase with toLowerCase() and replacing all the special characters with space through replaceAll. Within the parameters (regex[^a-zA-Z],

replacement ” ”), letters that are lowercase and capital are ignored so all the special characters will be removed or replaced with (“ ”). Next we create two HashMaps one for charCountMap which counts the characters in the text files and the other probability_map which will be used for the probability of the occurrences. char[] strArray is where we will be storing the characters of the text file that have already been filtered.

```
for (char c : strArray) {
    if (this.charCountMap.containsKey(c)) {
        this.charCountMap.put(c, this.charCountMap.get(c) + 1);
        this.probability_map.put(c, this.probability_map.get(c) + 1);
    }
    else {
        this.charCountMap.put(c, 1);
        this.probability_map.put(c, (float)1);
    }
}
```

Now we are going through all the characters in the string array using a for loop. Previously in the class Slice we created two Strings key and value. This is where they will be called, key will be the letter and value will be the number of occurrences for that letter. If the character is in the string array then it will be added to the HashMap key and the value will increase by one. This will be the same for the two HashMaps charCountMap and probability_map.

```
for (int entry : this.charCountMap.values()) {
    this.total += entry;
}

this.sorted_map = this.charCountMap.entrySet() Set<Map<K, V>.Entry<Character, Integer>>
        .stream() Stream<Map<K, V>.Entry<Character, Integer>>
        .sorted(reverseOrder(Map.Entry.comparingByValue()))
        .collect(Collectors.toList());

for (Map.Entry<Character, Float> entry: probability_map.entrySet()) {
    //divides hash by total
    float prob = entry.getValue()/this.total;
    //replace values in hash map with result
    probability_map.put(entry.getKey(), prob);
}
this.sorted_probabilitymap = this.probability_map.entrySet() Set<Map<K, V>.Entry<Character, Float>>
        .stream() Stream<Map<K, V>.Entry<Character, Float>>
        .sorted(reverseOrder(Map.Entry.comparingByValue()))
        .collect(Collectors.toList());
```

The for loop will go through the HashMap values and add all the elements within it to get the total amount. Below that we sorted charCountMap so that it is read by greatest to least and is stored in variable this.sortedmap. In the next for loop we go through the HashMap and divide the values in the HashMap by the total amount to get the probability. Then the values that we calculated will be replaced and put back into probability_map. Now we want to sort the probability_map so we use the same steps that we did for the this.sortedmap and now the probability_map is sorted from greatest to least in this.sorted_probabilitymap.

```
public static class MyPieChart {
    double x, y, height, width, StartAngle;
    int n;
    float total;
    MyColor color;
    List<Map.Entry<Character, Float>> pieChartMap;
```

Now we will talk about the inner class of class HistogramAlphaBet which is MyPieChart. The variables declared as double are x, y, height, width and StartAngle, n is declared as n, total declared as float and then we have the color and pieChartMap HashMap.

```
public MyPieChart(double x, double y, double height, double width, int n){
    this.x = x;
    this.y = y;
    this.height = height;
    this.width = width;
    this.StartAngle = 0;
    this.n = n;

    HistogramAlphaBet histoalphabet = new HistogramAlphaBet();
    this.total = 1;
    this.pieChartMap = (List<Map.Entry<Character, Float>>) histoalphabet.sorted_probabilitymap;
}
```

The parameters within MyPieChart initialize the variables that we declared above: double x, double y, double height, double width and int n. Then the this attribute is used to assign those variables to the current class, only difference is this.StartAngle where it has a value of 0 because it starts at 0. Below we are calling the HistogramAlphabet and assigning it to histoalpabet. The total is equal to 1 because the pie chart total is 1 and the probabilities of all letters add to 1. The HashMap for the sorted_probabilitymap is assigned to this.pieChartMap.

```
public void draw(GraphicsContext gc){
    int i = 0;
    float sum_total = 0;
    float rest_total = 0;
    Slice [] slice_pie_chart = new Slice[this.n+1];
    MyColor [] color_slice = MyColor.values();

    float [] float_value = new float[25];
    //copy map values to float array for easy indexing
    int p = 0;
    for(Map.Entry<Character, Float> entry : pieChartMap){
        float_value[p] = entry.getValue();
        p++;
        if (p == 25){
            break;
        }
    }
```

Here is the draw function where we will be drawing the pie chart. We set some variables to 0 so that we may use that later and increment. We create two arrays for slice_pie_chart which adds slice probabilities and color_slice for the color of the slices. The next array float_value is for the number of letters in the alphabet, so the size is 25. Now we have a for loop that copies the map values to the float array for east indexing. Outside the loop is a dummy variable and inside p will increment every time a value in added. If it reaches the max letters then the loop will stop.

```
while (i < n){
    sum_total += float_value[i];
    slice_pie_chart[i] = new Slice(this.x, this.y, this.width,this.StartAngle, angle: float_value[i]*360 , color_slice[i]);
    this.StartAngle += float_value[i]*360;
    i++;
    if (i >= n){
        rest_total = this.total - sum_total;
        //last slice
        slice_pie_chart[this.n] = new Slice(this.x, this.y, this.width,this.StartAngle, angle: rest_total*360 , MyColor.HOTPINK);
        slice_pie_chart[this.n].key = String.valueOf("Remaining");
        slice_pie_chart[this.n].value = String.valueOf(rest_total);
        break;
    }
}
```

We create a while loop which will keep adding slices until it is less than n. The sum_total will go through the float_value[i] and add the values into sum_total. Remember float_value is a copy of the array of probabilities. Now we will be adding to the slice_pie_chart[i], everytime i is incremented a new slice is added. The new Slice() takes in parameters x, y, width, StartAngle, float_value[i] * 360 and color_slice[i]. We multiply the float_value by 360 to convert to degrees and color_slice will take in a color. The StartAngle will be incremented by float_value[i] * 360 or incremented by the degrees. Underneath the while loop is a for loop that will run if i is greater than or equal n. This is for the empty space or the last big slice. The rest_total is calculated by the total subtracted by the sum_total, which is the total of all n slices. Once we find that we make a new slice and input the parameters similar to a normal slice. However instead of float_value[i] * 360 we will be inputting rest_total*360 since that will give the remaining angle. To make things clear that this is the last slice the key and value will be printed as strings.

```
int u = 0;
for(Map.Entry<Character, Float> entry : pieChartMap) {
    slice_pie_chart[u].key = String.valueOf(entry.getKey());
    slice_pie_chart[u].value = String.valueOf(entry.getValue());
    u++;
    if (u == this.n){
        break;
    }
}

int space = 50;
for (int o = 0; o < slice_pie_chart.length; o++){
    slice_pie_chart[o].draw(gc);
    gc.setFont(new Font( v: 20));
    gc.fillText(String.valueOf(slice_pie_chart[o].key),  v: 50, space);
    gc.fillText(slice_pie_chart[o].value,  v: 200, space);
    space += 20;
}
```

Similar to before we will be using a for loop that will go through the pieChartMap and print the key and value as strings on the canvas. This for loop will stop once the imputed n is reached. Underneath is the drawing for the pie chart on the canvas. The for loop illustrates each slice on the canvas. The setFont is the font size and fillText parameters for the key convert the array to a string and fillText for value just prints the value of the array.

Class Main

```
//button
int n = Integer.parseInt(JOptionPane.showInputDialog( parentComponent: null, message: "Add Slices"));

HistogramAlphaBet h1 = new HistogramAlphaBet();
HistogramAlphaBet.MyPieChart p = new HistogramAlphaBet.MyPieChart( x: 400, y: 10, height: 200, width: 550, n);
p.draw(gc);
```

The Class Main is identical to my previous assignments with only a few changes. A button is created so that the user can see the number of slices they want to see on the pie chart. Under that is the drawing for the actual pie chart with the dimensions within the parameters.

Class MyColor, MyShape, MyOval, Rectangle, MyShapeInterface, MyPoint

For these classes they have been reused from my previous assignments so please refer back to them for an explanation to avoid redundancy in this report.

3. Code Developed

Class Main

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

```
class Borderdim{
   static double height = 700;
   static double width = 1000;
}

public class Main extends Application {

   public void start(Stage primaryStage){
       BorderPane root = new BorderPane();
       Canvas canvas = new
Canvas(Borderdim.width,Borderdim.height);
       Scene scene = new Scene(root,
Borderdim.width,Borderdim.height,
MyColor.WHITE.getColor());
       GraphicsContext gc =
canvas.getGraphicsContext2D();

       //button
       int n =
Integer.parseInt(JOptionPane.showInputDialog(null,
"Add Slices"));

       HistogramAlphaBet h1 = new
HistogramAlphaBet();
       HistogramAlphaBet.MyPieChart p = new
HistogramAlphaBet.MyPieChart(400, 10, 200, 550, n);
       p.draw(gc);

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

   }

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

```
        launch(args);
    }
}
,Borderdim.height/6,Borderdim.width/9,Borderdim.hei
ght/6, MyColor.YELLOW);
        r.draw(gc);


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

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

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

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

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


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

```
        primaryStage.show();
    }

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

Class HistogramAlphaBet

```
package sample;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.text.Font;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import static java.util.Collections.reverseOrder;

public class HistogramAlphaBet {
    String data;
    HashMap<Character, Integer> charCountMap;
    float total = 0;
    List<Map.Entry<Character, Integer>> sorted_map;
    List<Map.Entry<Character, Float>>
sorted_probabilitymap;
    HashMap<Character, Float> probability_map;

    public HistogramAlphaBet() {
        Path path =
```

```
Paths.get("D:\\Users\\pug\\IdeaProjects\\project_3\\Alice
in Wonderland.txt");
        try {
            this.data = new
String(Files.readAllBytes(path));
        } catch (IOException e) {
            e.printStackTrace();
        }
        this.data = this.data.toLowerCase();
        this.data = this.data.replaceAll("[^a-zA-Z]", "");

        this.charCountMap = new HashMap<Character,
Integer>();
        this.probability_map = new HashMap<Character,
Float>();
        char[] strArray = this.data.toCharArray();

        for (char c : strArray) {
            if (this.charCountMap.containsKey(c)) {
                this.charCountMap.put(c,
this.charCountMap.get(c) + 1);
                this.probability_map.put(c,
this.probability_map.get(c) + 1);
            }
            else {
                this.charCountMap.put(c, 1);
                this.probability_map.put(c, (float)1);
            }
        }

        for (int entry : this.charCountMap.values()) {
            this.total += entry;
        }

        this.sorted_map = this.charCountMap.entrySet()
                .stream()
```

```
.sorted(reverseOrder(Map.Entry.comparingByValue()))
                .collect(Collectors.toList());

        for (Map.Entry<Character, Float> entry:
probability_map.entrySet()) {
            //divides hash by total
            float prob = entry.getValue()/this.total;
            //replace values in hash map with result
            probability_map.put(entry.getKey(), prob);
        }
        this.sorted_probabilitymap =
this.probability_map.entrySet()
                .stream()

.sorted(reverseOrder(Map.Entry.comparingByValue()))
                .collect(Collectors.toList());
    }

    public static class MyPieChart {
        double x, y, height, width, StartAngle;
        int n;
        float total;
        MyColor color;
        List<Map.Entry<Character, Float>> pieChartMap;

        public MyPieChart(double x, double y, double
height, double width, int n){
            this.x = x;
            this.y = y;
            this.height = height;
            this.width = width;
            this.StartAngle = 0;
            this.n = n;

            HistogramAlphaBet histoalphabet = new
HistogramAlphaBet();
            this.total = 1;
```

```
            this.pieChartMap = (List<Map.Entry<Character,
Float>>) histoalphabet.sorted_probabilitymap;
        }

        public void draw(GraphicsContext gc){
            int i = 0;
            float sum_total = 0;
            float rest_total = 0;
            Slice [] slice_pie_chart = new
Slice[this.n+1];
            MyColor [] color_slice = MyColor.values();

            float [] float_value = new float[25];
            //copy map values to float array for easy
indexing
            int p = 0;
            for(Map.Entry<Character, Float> entry :
pieChartMap){
                float_value[p] = entry.getValue();
                p++;
                if (p == 25){
                    break;
                }
            }

            while (i < n){
                sum_total += float_value[i];
                slice_pie_chart[i] = new Slice(this.x,
this.y, this.width,this.StartAngle,float_value[i]*360 ,
color_slice[i]);
                this.StartAngle += float_value[i]*360;
                i++;
                if (i >= n){
                    rest_total = this.total - sum_total;
                    //last slice
                    slice_pie_chart[this.n] = new
Slice(this.x, this.y,
```

```
this.width,this.StartAngle,rest_total*360 ,
MyColor.HOTPINK);
                    slice_pie_chart[this.n].key =
String.valueOf("Remaining");
                    slice_pie_chart[this.n].value =
String.valueOf(rest_total);
                    break;
                }
            }

            int u = 0;
            for(Map.Entry<Character, Float> entry :
pieChartMap) {
                slice_pie_chart[u].key =
String.valueOf(entry.getKey());
                slice_pie_chart[u].value =
String.valueOf(entry.getValue());
                u++;
                if (u == this.n){
                    break;
                }
            }

            int space = 50;
            for (int o = 0; o < slice_pie_chart.length;
o++){
                slice_pie_chart[o].draw(gc);
                gc.setFont(new Font(20));

gc.fillText(String.valueOf(slice_pie_chart[o].key), 50,
space);
                gc.fillText(slice_pie_chart[o].value, 200,
space);
                space += 20;
            }
        }
    }
```

```
}
```

Class MyArc

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

public class MyArc extends MyShape {
   double x,y,radius, startAngle, centralAngle;
   public MyColor color;

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

   public double getRadius() {
        return this.radius;
   }
   public double getStartAngle() {
        return this.getStartAngle();
   }
   public double getCentralAngle() {
        return this.getCentralAngle();
   }
   public double getMyArea() {
        return (Math.PI * (Math.pow(radius, 2)) *
(centralAngle/360));
```

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

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

    public String toString() {
        return "Radius:" + getRadius() + "\n
Starting Angle: " + getStartAngle() +
                "\nCentral Angle " +
getCentralAngle() + "\nArc Area:" + getMyArea();
    }
    @Override
    public void draw(GraphicsContext gc) {
        gc.setFill(this.color.getColor());
        gc.strokeArc(this.x, this.y, this.radius,
this.radius,  this.startAngle, this.centralAngle,
ArcType.ROUND);
        gc.fillArc(this.x, this.y, this.radius,
this.radius, this.startAngle, this.centralAngle,
ArcType.ROUND);
    }
}
```

Class MyColor

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

public enum MyColor {
    RED(255,0,0),
    BLUE(0,0,255),
    LIME(0,255,0),
    CYAN(0,255,255),
    GREEN(0,128,0),
    GREY(128,128,128),
    MAGENTA(255,0,255),
    PURPLE(128,0,128),
    VIOLET(148,0,211),
    YELLOW(255,255,0),
    WHITE(255,255,255),
    BLACK(0,0,0),
    HOTPINK(255,105,180),
    CHOCOLATE(210, 105, 30),
    DARKSLATEGRAY(47,79,79),
    MIDNIGHTBLUE(25,25,112),
    INDIGO(75,0,130),
    AQUAMARINE(127,255,212),
    SPRINGGREEN(0,255,127),
    DARKOLIVEGREEN(85,107,47),
    ORANGERED(255,69,0),
    MAROON(128,0,0),
    ORANGE(255,165,0),
    OLIVE(128,128,0),
    DARKSEAGREEN(143,188,143);

    int Red;
    int Green;
    int Blue;

    MyColor(int Red, int Green, int Blue){
```

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

Class MyOval

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

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

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

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

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

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

//maxX
```

```
//maxY
        return p.getXpoint() >= getX() && p.getXpoint() <= getX()
+ width && p.getYpoint() >= getY() && p.getYpoint() <= getY() +
height;
    }
    @Override
    public MyRectangle getMyBoundingRectangle() {
        return new MyRectangle(getX(), getY(),this.majorAxis,
this.minorAxis,this.color);
    }
    @Override
    public void draw(GraphicsContext gc){
        gc.setFill(this.color.getColor());
        gc.strokeOval(getX(), getY(), width, height);
        gc.fillOval(getX(), getY(), width, height);
        gc.strokeRect(getX(), getY(), width, height);

    }


}
```

Class MyPoint

```java
package sample;


public class MyPoint {
    private double x,y;

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

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

}
```

Class MyRectangle

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


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

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

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

```
    }

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

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

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

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

    }
}
```

Class MyShape

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

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

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

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

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

   public String toString(){
        return "X:" + getX() + "\nY:" + getY();
   }
   abstract void draw (GraphicsContext gc);
}
```

Class MyShapeInterface

```
package sample;

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

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

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

4. Outputs Produced