Report: Assignment:3

MYESHA MAHAZABEEN

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

Instructor: Hesham A. Auda
Date: 07/17/2022

Abstract: This assignment improves assignment-1 and assignment-2 by adding some additional classes which make use of previous classes in solving a real problem. Additionally, this assignment introduces inner class, Streams, and Maps. This report also explains every aspect of assignment-3 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 3

A report in PDF or MS Word 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 11:59 *pm on Sunday,* 17 July 2022. ***The deadline is strictly observed***.

1- Implement a Java class **HistogramAlphaBet** that calculates the *frequencies* of the alphabet characters in by *Leo Tolstoy*'s "*War and Peace*" (file *War and Peace.txt*) and their *probabilities*. The **HistogramAlphaBet** class utilizes **Map** collections, *Map*<**Character**, **Integer**> and *Map*<**Character**, **Double**> and *stream* operations, for statistical calculations and sorting of the frequencies and probabilities. It also includes a **MyPieChart** class as an inner class.

2- Class **MyPieChart** displays a pie chart of the probabilities of the ***n*** most frequent occurrences of an event – the frequency of characters in a document. The probability of event is given by:

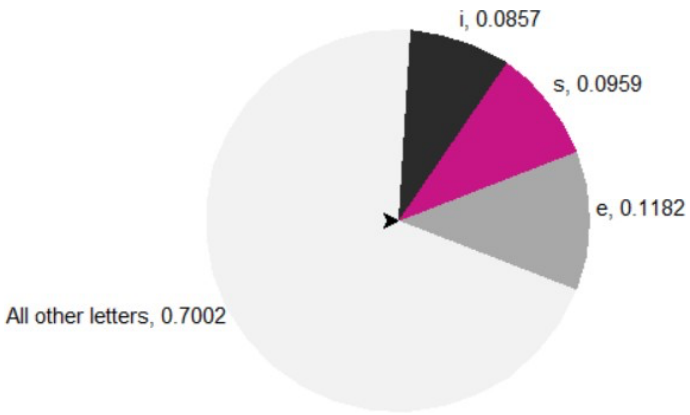
$$Probability\ of\ event = \frac{Frequency\ of\ event}{\sum\ Frequencies\ of\ all\ events}$$

In the pie chart:

i. Each event is represented by a slice of the pie chart. The size of the slice is proportional to the probability of the corresponding event:

$$Probability\ of\ event = \frac{Central\ angle\ of\ slice}{2\pi}$$

ii. Each slice has a different color of your choice of type *enum* **MyColor**;
iii. Each slice has a legend showing the corresponding event and its probability;
iv. The slices are displayed in order of *decreasing* probability;
v. The last slice represents "*All Other Events*" and their cumulative probability. *As an example*, in the graph below where the event is the occurrence of a letter in a text: ***n*** = 3, and the probability of All Other Events is *one* minus the sum of the probabilities of events *e*, *s*, and *i*.



3- Class **MyPieChart** utilizes a **Map** collection *Map*<**Character**, **Slice**>, and includes appropriate constructors and a method *draw* that draws the pie chart. The drawing canvas includes appropriate GUI components to input the number of events, ***n*** (variable), and display the pie chart together with the characters and their probabilities.

4- Class **Slice** utilizes the **MyArc** class in the **MyShape** class hierarchy, and includes appropriate constructors and methods, including methods that perform the following operations:

a. *toString*— returns a string representation of a **Slice** object;
b. *draw*— draws a **Slice** object.

5- You may only use JavaFX graphics and your own classes and methods for the operations included. Further,

a. The code is applicable to canvases of variable height and width;
b. The size of the pie chart is proportional to the smallest dimension of the canvas.

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

Hesham A Auda 9
July 2022

Table of Contents

Solution:

This project contains nine classes, an enum class, an interface, an inner class MyPieChart and a Main class. Main class contains everything together to generate the pie chart as output using all the classes created in Assignment-3. 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. Slice class extends MyArc and used here to draw each slice of the pie chart with details. Another class called HistogramAlphaBet reads the provided file "War and Peace.txt" and calculates frequencies of each alphabet character from the text file and finds their probabilities. Additionally, it also uses two separate java Map objects and has an inner class called MyPieChart, which helps us draw the pie chart. Main class uses HistogramAlphaBet and MyPieChart object to draw the pie chart. MyPieChart itself creates all the slices and draws them.

Classes and Methods:

- **Main.java:**
  Uses the JavaFx library to draw pie chart 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 pie chart depend on canvas' smallest dimension. At first main class outputs an Input GUI that asks for an input number to take value for n (number of slices). After pressing ok, a JavaFx window pops up containing a colorful pie chart of n largest slices with all other slices if exists. Additionally, the window draws the probabilities of the slices at the top left corner.

  - ***Imports:***

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

  - ***Methods:***

    - **start();** This method creates a HistogramAlphaBet object and a MyPieChart object and then draws the pie chart using MyPieChart's draw() function. It creates necessary Panes, canvas etc for drawing.
    - **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 its 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.

```
// 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.

```
// 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.

```
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.

```java
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

```java
// 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.

```
// 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/

```
    @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.

```
@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.

```
@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

```
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;
    }
}
```

- **Slice.java:**
  This class is made to draw each slice of the pie chart along with details. Slice.java extends the class **MyArc**. It adds some additional attributes to that helps it better describe each slice of the pie chart. This class is used by MyPieChart class in order to draw the complete pie chart. This class has two constructors that use super constructor to initialize the attribute values. Class slice's implementation details has been described thoroughly below:

  - ***Imports:***

    - Javafx.scene.canvas.GraphicsContext;

  - ***Attributes:***

    - **MyColor color:** This attribute is passed on to class Slice by its superclass MyArc. It describes a Slice's color in the pie chart.
    - **MyOval o:** This attribute is also obtained from superclass MyArc. This stores the Oval the MyArc object lies within.
    - **double startAngle, angle:** Obtained from superclass MyArc, startAngle and angle describe at which angle the slice starts and the body angle of the slice respectively.
    - **String c:** This string stores the character that the slice is representing. A string has been taken here instead of Character because a slice might represent "All other characters", which cannot be stored inside a character.
    - **double yPos:** This variable is kept to draw description of the slice in appropriate height of canvas. It is used in draw() method.

```
private String c;
private double yPos;
```

  - ***Constructors:***

    - Class Slice has two constructors that initialize the value of the attributes belonging to Slice.

```
public Slice(MyColor color, MyOval o, double startAngle, double angle, String c) {
    super(color, o, startAngle, angle);
    this.c = c;
    this.yPos = 0;
}

public Slice(MyColor color, MyPoint center, double h, double w, double startAngle, double angle, String c) {
    this(color, new MyOval(center, h, w), startAngle, angle, c);
}
```

- ***Methods:***

  - **getyPos():** returns value of yPos attribute of Slice object.
  - **setyPos():** sets value of yPos attribute according to given value.
  - **getC():** Returns the character the Slice object is representing.

```
public double getyPos() { return yPos; }

public void setyPos(double yPos) { this.yPos = yPos; }

public String getC() { return c; }
```

  - **draw():** Draws the slice and the description of the slice object in the given GraphicsContext. This method uses super class MyArc's draw method and also adds some more to draw the texts describing the slice.
  - **toString():** Returns a String object containing a few details of the Slice object such as: the character a slice represents, probability of getting that character in the given file, and the color of the slice.

```
@Override
void draw(GraphicsContext graphicsContext) {
    super.draw(graphicsContext);
    graphicsContext.fillText( s: c+":\t"+angle/360.0,  v: 40, yPos);
    graphicsContext.fillOval( v: 10,  v1: yPos-15,  v2: 20,  v3: 20);
}

@Override
public String toString() {
    return "Slice{" +
            "\n\tCharacter: " + this.c +
            "\n\tProbability: " + this.angle*360.0 +
            "\n\tColor: "+ this.color.getHexRepresentation() +
            "\n}";
}
```

- **HistogramAlphaBet.java:**
  This class reads from provided file "War and Peace.txt" and calculates frequencies of each alphabet character in the file along with the probability of getting each character. It stores the frequencies and the probability in two separate java Map objects. HistpgramAlphaBet class has a constructor which initiates the calculation of the frequency and file read by calling another method calculateFreq which is private. So, upon constructing an object of

HistogramAlphaBet, all the frequencies and probabilities are already calculated. This class also has an **inner class**, which is called **MyPieChart.**

- ***Imports:***
  - Javafx.scene.canvas.GraphicsContext;
  - Javafx.scene.control.TextInputDialog **:** to take input value n.
  - Java.io.FileInputStream: helps to read a file.
  - Java.io.InputStream;
  - Java.util.*
  - Java.lang.Math.min;
- ***Constructors:***
  - This class has a constructor which initialize the value of frequency and probability map by calling **calculateFreq()** function.

```
public HistogramAlphaBet() throws Exception {
    for(int tmp = 0; tmp<=255; tmp++) {
        freq.put((char)tmp, 0);
    }
    calculateFreq();
}
```

- ***Attributes:***
  - **Map<Character, Integer> freq:** this is a HashMap that stores a character as key and the number of occurrences of that character in provided input file.
  - **Map<Character, Double> prob:** this is a HashMap that stores a character as key and the probability of getting that character from provided input file.

```
private Map<Character, Integer> freq = new HashMap<>();
private Map<Character, Double> prob = new HashMap<>();
```

- ***Methods:***
  - **getFreq():** returns frequency Map of the object.
  - **getProb():** returns probability Map of the object.

```
public Map<Character, Integer> getFreq() { return freq; }

public Map<Character, Double> getProb() { return prob; }
```

- **calculateFreq():** This method is private and it reads the provided file "War and Peace.txt" into an inputStream. Then it counts occurrence of each character in the text file and stores it in **Map freq.** After calculating frequencies, it calculates probability and stores it in the **Map prob.**

```
private void calculateFreq() throws Exception{
    InputStream inputStream = new FileInputStream( name: "War and Peace.txt");
    byte array[] = inputStream.readAllBytes();
    String s = new String(array);
    double cc = 0;
    Character c;
    System.out.println(s.length());
    for(int i=0; i<s.length(); i++){
        c = Character.toLowerCase(s.charAt(i));
        if(c<'a' || c>'z') {
            continue;
        }
        freq.put(c, freq.get(c)+1);
        c = (char)inputStream.read();
        cc++;
    }
    for(Character tmp = 'a'; tmp<='z'; tmp++) {
        prob.put(tmp, (double) (freq.get(tmp) / cc));
    }
}
```

- **Inner Class: MyPieChart**
  MyPieChart is an inner class of HistogramAlphaBet**.** That means, it is declared and defined inside the latter mentioned class. Purpose of this class is to get the number of slices to be drawn and draw the pie chart containing that number of slices. This class has a constructor which take height and width of the canvas as input and it creates necessary Slice objects to fill the **Map slices.**

  - ***Attributes:***

    - **Map<Character, Slice> slices:** This attribute stores each slice of the full pie chart as value and the character themselves as keys.
    - **int n:** This attribute stores the number of slices to be drawn.

- ***Constructors:***

  - This inner class MyPieChart has a constructor which takes height and width of the canvas. It takes the probability from the **HistogramAlphaBet** class and calculates the angle each character creates in the pie chart. Then it constructs each Slice based on that angle. Here, the angle is proportional to the probability of a character. ***Angle = probability*360***. This class also gives each slice arbitrary colors.

```
public class MyPieChart {
    private Map<Character, Slice> slices = new HashMap<>();
    private int n;

    public MyPieChart(double w, double h) {
        MyCircle pieChart = new MyCircle( x: w/2,  y: h/2,  r: min(h, w)*0.5);
        double tot_ang = 0;
        MyColor[] colors = new MyColor[49];
        Slice slice;
        int i = 0;
        for (MyColor color : MyColor.values()) {
            colors[i++] = color;
        }
        i = 15;
        for(Character key: prob.keySet()) {
            double angle = 360.0*prob.get(key);
            slice = new Slice(colors[i], pieChart, tot_ang, angle,  c: key+"");
            slices.put(key, slice);
            i++;
            tot_ang += angle;
        }
    }
```

- ***Methods:***

  - **takeInput():** This is a private method which is used by draw method to get the value of n before drawing the pie chart. It uses javafx TextInputDialogue to take input.

```
private void takeInput() {
    TextInputDialog td = new TextInputDialog();
    td.setTitle("Number of Slices");
    td.setContentText("Enter value of n:");
    td.setHeaderText("Input");
    td.showAndWait();
    String s = td.getEditor().getText();
    n = Integer.parseInt(s);
}
```

- **draw():** Draw method is responsible for drawing the whole pie chart. This method first takes n as input using **takeInput()** function. Then it calculates all other character's probability slice. It stores all the Slices to be drawn in a TreeMap which sorts all the slices to be drawn based on their body angle or probability. After it sorts the slices, all the slices are drawn into the given GraphicsContext using **Slice** class's **draw()** method.

```
public void draw(GraphicsContext graphicsContext) {
    takeInput();

    double h = graphicsContext.getCanvas().getHeight();
    double w = graphicsContext.getCanvas().getWidth();

    MyCircle pieChart = new MyCircle( x: w/2,  y: h/2,  r: min(h, w)*0.5);
    Slice slice;

    Map<Double, Slice> drawableSlices = new TreeMap<>(Collections.reverseOrder());
    for(Slice s:slices.values()) {
        drawableSlices.put(s.angle, s);
    }

    int j=0;
    double tot_ang = 0;
    for (Slice s:drawableSlices.values()){
        tot_ang += s.angle;
        j++;
        if(j==n) break;
    }

    slice = new Slice(MyColor.DARKVIOLET, pieChart,  startAngle: 0,  angle: 360-tot_ang,  c: "Others");
    drawableSlices.put(slice.angle, slice);
    j=0;
    tot_ang=90;
    int textYPos=25;
    for (Double key:drawableSlices.keySet()){
        drawableSlices.get(key).startAngle=tot_ang;
        tot_ang += key;
        j++;
        drawableSlices.get(key).setyPos(textYPos);
        drawableSlices.get(key).draw(graphicsContext);
        System.out.println(drawableSlices.get(key).toString());
        textYPos += 25;
        if(j==n+1) break;
    }
}
```

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:**

```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);
    }
```

```
    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) {
        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);
}
```

```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:**

```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;
    }
}
```

Classes Added in Assignment-3

- **Slice.java**

```java
package com.example.assignment2;

import javafx.scene.canvas.GraphicsContext;

public class Slice extends MyArc{

    private String c;
    private double yPos;

    public Slice(MyColor color, MyOval o, double startAngle, double angle, String c) {
        super(color, o, startAngle, angle);
        this.c = c;
        this.yPos = 0;
    }

    public Slice(MyColor color, MyPoint center, double h, double w, double startAngle, double angle, String c) {
        this(color, new MyOval(center, h, w), startAngle, angle, c);
    }

    public double getyPos() { return yPos; }

    public void setyPos(double yPos) { this.yPos = yPos; }

    public String getC() { return c; }

    @Override
    void draw(GraphicsContext graphicsContext) {
        super.draw(graphicsContext);
        graphicsContext.fillText( s: c+":\t"+angle/360.0,  v: 40, yPos);
        graphicsContext.fillOval( v: 10,  v1: yPos-15,  v2: 20,  v3: 20);
    }
```

```java
@Override
public String toString() {
    return "Slice{" +
            "\n\tCharacter: " + this.c +
            "\n\tProbability: " + this.angle*360.0 +
            "\n\tColor: "+ this.color.getHexRepresentation() +
            "\n}";
}
```

- **HistogramAlphaBet.java**

```java
package com.example.assignment2;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.TextInputDialog;
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.*;
import static java.lang.Math.min;
```

```java
public class HistogramAlphaBet {

    private Map<Character, Integer> freq = new HashMap<>();
    private Map<Character, Double> prob = new HashMap<>();

    public HistogramAlphaBet() throws Exception {
        for(int tmp = 0; tmp<=255; tmp++) {
            freq.put((char)tmp, 0);
        }
        calculateFreq();
    }

    public Map<Character, Integer> getFreq() { return freq; }

    public Map<Character, Double> getProb() { return prob; }
```

```java
private void calculateFreq() throws Exception{
    InputStream inputStream = new FileInputStream( name: "War and Peace.txt");
    byte array[] = inputStream.readAllBytes();
    String s = new String(array);
    double cc = 0;
    Character c;
    System.out.println(s.length());
    for(int i=0; i<s.length(); i++){
        c = Character.toLowerCase(s.charAt(i));
        if(c<'a' || c>'z') {
            continue;
        }
        freq.put(c, freq.get(c)+1);
        c = (char)inputStream.read();
        cc++;
    }
    for(Character tmp = 'a'; tmp<='z'; tmp++) {
        prob.put(tmp, (double) (freq.get(tmp) / cc));
    }
}
```

- **MyPieChart**

```
package com.example.assignment2;

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

```
public class MyPieChart {
    private Map<Character, Slice> slices = new HashMap<>();
    private int n;

    public MyPieChart(double w, double h) {
        MyCircle pieChart = new MyCircle( x: w/2,  y: h/2,  r: min(h, w)*0.5);
        double tot_ang = 0;
        MyColor[] colors = new MyColor[49];
        Slice slice;
        int i = 0;
        for (MyColor color : MyColor.values()) {
            colors[i++] = color;
        }
        i = 15;
        for(Character key: prob.keySet()) {
            double angle = 360.0*prob.get(key);
            slice = new Slice(colors[i], pieChart, tot_ang, angle,  c: key+"");
            slices.put(key, slice);
            i++;
            tot_ang += angle;
        }
    }
```

```
public void draw(GraphicsContext graphicsContext) {
    takeInput();
    double h = graphicsContext.getCanvas().getHeight();
    double w = graphicsContext.getCanvas().getWidth();
    MyCircle pieChart = new MyCircle( x: w/2,  y: h/2,  r: min(h, w)*0.5);
    Slice slice;

    Map<Double, Slice> drawableSlices = new TreeMap<>(Collections.reverseOrder());
    for(Slice s:slices.values()) {
        drawableSlices.put(s.angle, s);
    }
    int j=0;
    double tot_ang = 0;
    for (Slice s:drawableSlices.values()){
        tot_ang += s.angle;
        j++;
        if(j==n) break;
    }
```

```
    slice = new Slice(MyColor.DARKVIOLET, pieChart, startAngle: 0, angle: 360-tot_ang, c: "Others");
    drawableSlices.put(slice.angle, slice);
    j=0;
    tot_ang=90;
    int textYPos=25;
    for (Double key:drawableSlices.keySet()){
        drawableSlices.get(key).startAngle=tot_ang;
        tot_ang += key;
        j++;
        drawableSlices.get(key).setyPos(textYPos);
        drawableSlices.get(key).draw(graphicsContext);
        System.out.println(drawableSlices.get(key).toString());
        textYPos += 25;
        if(j==n+1) break;
    }
}
```

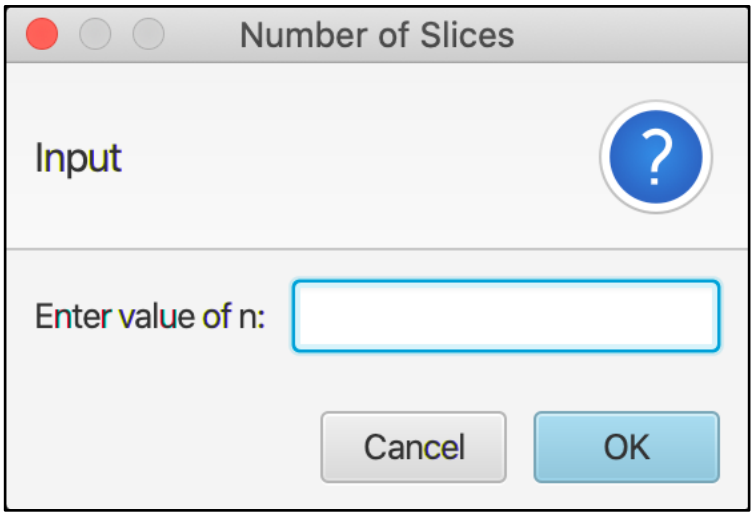
- **Main.java**

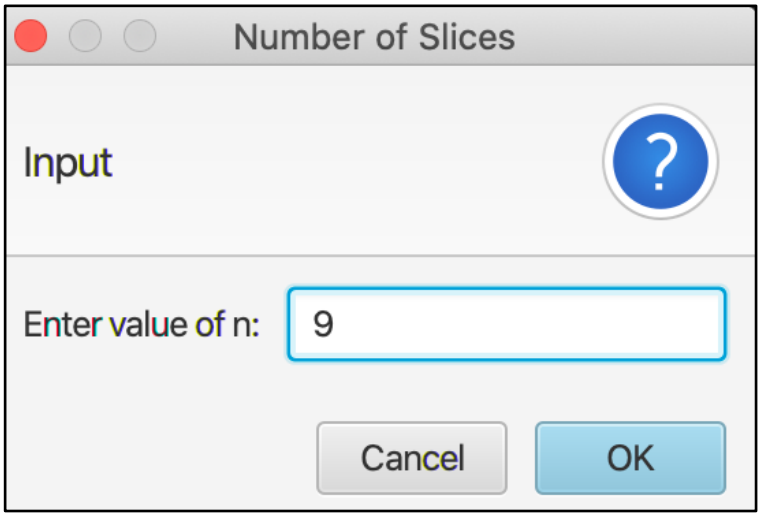
```
package com.example.assignment2;

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;
```

```
public class Main extends Application {
    @Override
    public void start(Stage stage) throws Exception {

        int w = 1000;
        int h = 600;

        HistogramAlphaBet ha = new HistogramAlphaBet();
        HistogramAlphaBet.MyPieChart pi = ha.new MyPieChart(w, h);

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

        pi.draw(graphicsContext);

        root.getChildren().add(canvas);
        stage.setScene(scene);
        stage.setTitle("Assignment-3");
        stage.show();
    }

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

}
```

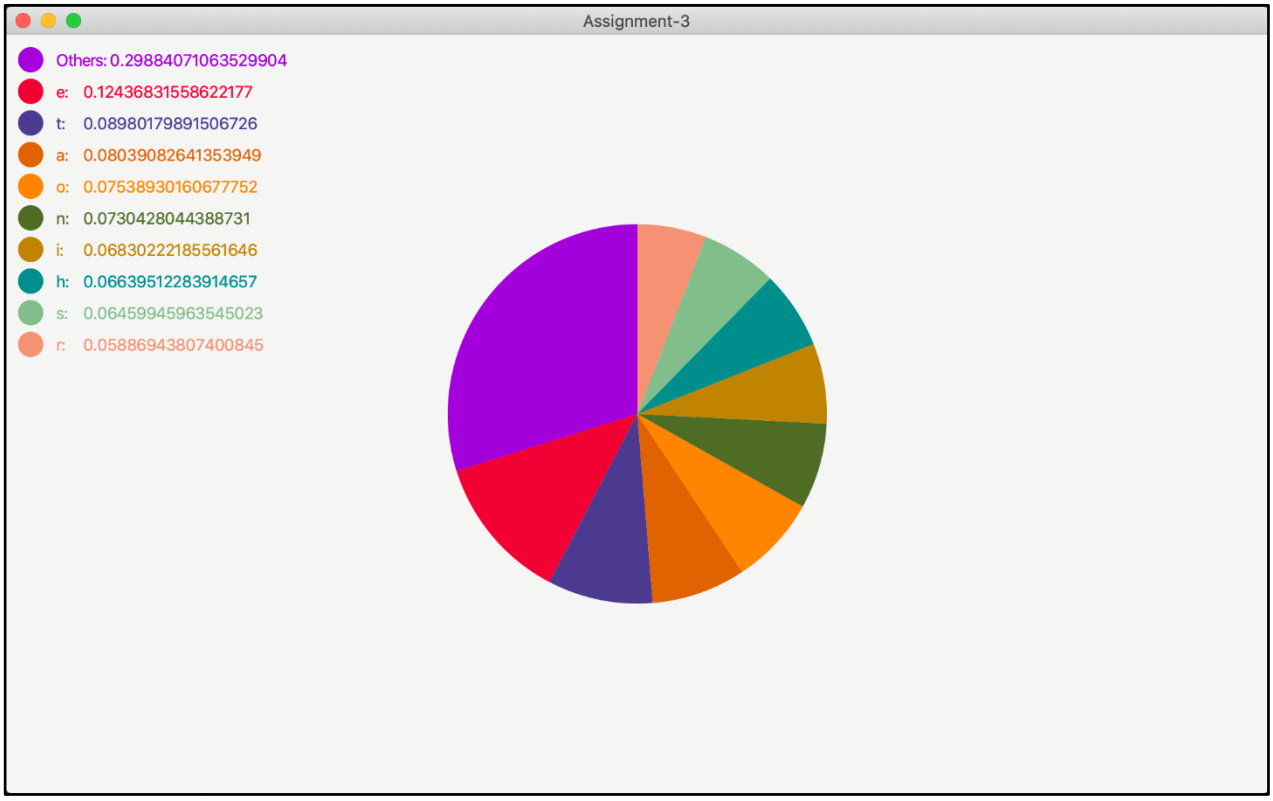
Screenshots of Output:

- **JavaFx Windows:**
    - ***Input GUI:***
      This graphical interface takes the value of n, upon pressing ok, n largest slices of the pie chart will be drawn along with all other's slice if it exists.





- ***Output window:***
  This window shows all the slices in the pie chart, at the top left corner side it draws probability of each slice



- **Console Output:**
  toString of all the Slices have been printed on console to get better understanding and testing.

```
Slice{
    Character: Others
    Probability: 0.25192940600746094
    Color: #940d3
}
Slice{
    Character: e
    Probability: 0.12436831558622177
    Color: #dc143c
}
Slice{
    Character: t
    Probability: 0.08980179891506726
    Color: #483d8b
}
Slice{
    Character: a
    Probability: 0.08039082641353949
    Color: #d2691e
}
Slice{
    Character: o
    Probability: 0.07538930160677752
    Color: #ff8c0
}
Slice{
    Character: n
    Probability: 0.0730428044388731
    Color: #556b2f
}
Slice{
    Character: i
    Probability: 0.06830222185561646
    Color: #b886b
}
Slice{
    Character: h
    Probability: 0.06639512283914657
    Color: #08b8b
}
Slice{
    Character: s
    Probability: 0.06459945963545023
    Color: #8fbc8f
}
Slice{
    Character: r
    Probability: 0.05886943807400845
    Color: #e9967a
}
Slice{
    Character: d
    Probability: 0.046911304627838095
    Color: #fff8dc
}

Process finished with exit code 0
```