Fall 2022 | CSC 22100 | Assignment 3

Alhamza Muswara

Problem/Task:

1. Implement a Java class HistogramAlphaBet.
2. Inner class MyPieChart which displays a pie chart of the probabilities
   a. utilizes a Map collection
3. Create class Slice which utilizes the MyArc class in the MyShape
4. Using JavaFX display chart from reading file “WarandPeace.txt”.

[2] Solution Methods

The MySlice class mainly utilizes the MyArc class to draw, with the appropriate constructors.

MySlice

```
package application;

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


public class Slice {
	MyArc slice;
	Double angleSlice;
	Double startAngle;
	Double endAngle;

	Slice(double x, double y, double w, double h, double angle1, double angle2, 
ArcType closure) {
		slice = new MyArc(x, y, w, h, angle1, angle2, closure);
		this.endAngle=angle2;
		this.angleSlice=angle2-angle1;
	}

	void setFill(MyColor c) {
		slice.fill = c;
	}

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

	void draw(GraphicsContext gc) {
		slice.draw(gc);
	}
}
```

The class has the getFreq() function that replace non alphabetical literals with empty character ‘’, and creates entries for each letter present with the frequency. The setFract sets each letter present as probability to another collection. The getSlices() creates another collection but with letters as keys and there corresponding slice object. The HistoframAphaBet class has a draw function that draw a slice of a pie that represents with its probability with different colors.

Class HistogramAlphaBet + MyPieChart(Inner Class)

```
package application;

import java.util.*;
import java.util.Map;
import java.util.Map.Entry;

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

import java.io.File; // Import the File class
import java.util.Scanner; // Import the Scanner class to read text files
import java.net.URL;
import java.text.DecimalFormat;

public class HistogramAlphaBet {
    Map<Character, Integer> characterFreq = new HashMap<Character, Integer>();
    Map<Character, Double> characterFract = new HashMap<Character, Double>();
    String finalS = "";
    Integer sum = 0;

    class MyPieChart {
        Map<Character, Slice> slices = new HashMap<Character, Slice>();
        int topN = 0;
        double startAngle = 0;
        Slice s;

        void getSlices(int n) {
            Map.Entry<Character, Double> entryWithMaxPercentage = null;
            for (MyColor color : MyColor.values()) {
                //
                entryWithMaxPercentage = null;
                for (Entry<Character, Double> entry :
characterFract.entrySet()) {
                    if (entryWithMaxPercentage == null
                                          ||
entry.getValue().compareTo(entryWithMaxPercentage.getValue()) > 0) {
                        entryWithMaxPercentage = entry;
                    }
                }
                //
System.out.println(entryWithMaxPercentage.getValue()*sum);
                System.out.println(entryWithMaxPercentage.getKey());

                characterFract.remove(entryWithMaxPercentage.getKey());
                s = new Slice(300, 300, 300, 300, startAngle, startAngle +
(360 * entryWithMaxPercentage.getValue()),
                        ArcType.ROUND);
                s.setFill(color);
                slices.put(entryWithMaxPercentage.getKey(), s);
                startAngle += (360 * entryWithMaxPercentage.getValue());
                topN++;
                if (topN == n || characterFract.size()==0) {
                    s = new Slice(300, 300, 300, 300, startAngle, 360,
ArcType.ROUND);
                    s.setFill(MyColor.AliceBlue);
                    slices.put('*', s);
                    break;
                }
            }

        }

        void draw(GraphicsContext gc) {
            DecimalFormat f = new DecimalFormat("#.###");
            Double pointSlice = (double) 0;
            for (Entry<Character, Slice> entry : slices.entrySet()) {
                pointSlice = entry.getValue().endAngle -
(entry.getValue().angleSlice / 2);
                entry.getValue().draw(gc);
                System.out.println(entry.getValue().angleSlice / 360);
                System.out.println(entry.getKey());
                if (pointSlice <= 90) {
                    gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                            (150 *
Math.cos(Math.toRadians(pointSlice))) + 450,
                            -(150 *
Math.sin(Math.toRadians(pointSlice))) + 450);

                } else if (pointSlice <= 180) {
                    gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                            -(150 *
Math.sin(Math.toRadians(pointSlice - 90))) + 450 - 30,
                            -(150 *
Math.cos(Math.toRadians(pointSlice - 90))) + 450 - 20);

                }

                else if (pointSlice <= 270) {
                    gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                            -(150 *
Math.cos(Math.toRadians(pointSlice - 180))) + 450 - 20,
                            (150 *
Math.sin(Math.toRadians(pointSlice - 180))) + 450 + 20);

                } else if (pointSlice <= 360) {
```

```java
                    gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                            (150 *
Math.sin(Math.toRadians(pointSlice - 270))) + 450 + 10,
                            (150 *
Math.cos(Math.toRadians(pointSlice - 270))) + 450 + 10);

                }
            }

        }
    }

    public void setFreq(String filename) throws Exception {

        URL path = HistogramAlphaBet.class.getResource(filename);
        File myObj = new File(path.getFile());
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
            finalS += myReader.nextLine().replaceAll("[^a-zA-Z]",
"").toLowerCase();
        }
        myReader.close();
        for (int i = 0; i < finalS.length(); i++) {
            if (characterFreq.containsKey(finalS.charAt(i))) {
                characterFreq.put(finalS.charAt(i),
characterFreq.get(finalS.charAt(i)) + 1);
            } else {
                characterFreq.put(finalS.charAt(i), 1);
            }
            // Print current character finalS.charAt(i)

        }
    }

    public void setFract() {
        for (Entry<Character, Integer> entry : characterFreq.entrySet()) {
            sum += entry.getValue();
        }

        for (Entry<Character, Integer> entry : characterFreq.entrySet()) {
            characterFract.put(entry.getKey(), (double) entry.getValue() /
sum);
        }

    }

    public void printFinal() {
        System.out.println(finalS);
    }
}
```

[3] Codes Developed

Classes Imported

```
import java.lang.Math;
import java.util.ArrayList;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import java.io.File;
import java.util.Scanner;
import java.net.URL;
import java.text.DecimalFormat;

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

Class MyPoint:

```
package application;

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

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

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

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

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

       void shiftX(double x) {
           this.x+=x;
       }
       void shiftY(double y) {
              this.y+=y;
```

```
    }
    public String toString() {
        return "MyPoint [x = "+x+", y = "+y+"]";
    }


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

}
```

Class MyShape:

```
package application;

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

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

    // area, parameter

    abstract double getArea();

    void draw() {

    }

    abstract double getWidth();

    abstract double getHeight();

    abstract double getPerimeter();

    // toString

    public String toString() {
        return "";
    }

    // draw
    abstract void draw(GraphicsContext gc);

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

}
```

Class Slice

```
package application;

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


public class Slice {
    MyArc slice;
    Double angleSlice;
    Double startAngle;
    Double endAngle;

    Slice(double x, double y, double w, double h, double angle1, double angle2, 
ArcType closure) {
        slice = new MyArc(x, y, w, h, angle1, angle2, closure);
        this.endAngle=angle2;
        this.angleSlice=angle2-angle1;
    }

    void setFill(MyColor c) {
        slice.fill = c;
    }

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

    void draw(GraphicsContext gc) {
        slice.draw(gc);
    }
}
```

Class MyArc

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

public class MyArc extends MyShape {
    MyPoint p1, p2;
    private double width;
    private double height;
    private double startAngle;
    private double arcExtent;
    private ArcType closuretype;
    MyArc(double x, double y, double w, double h, double angle1, double angle2,
ArcType closure) {
        p.setX(x);
        p.setY(y);
        this.width = w;
        this.height = h;
        this.startAngle = angle1;
        this.arcExtent = angle2 - angle1;
        this.closuretype = closure;
    }

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

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

    @Override
    double getArea() {
        return 1;
    }

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

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

    @Override
    double getWidth() {
        return width;
    }
    @Override
    double getHeight() {
        return height;
    }

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

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

        }
        return false;
    }

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

}
```

Class HistogramAlphaBet

```
package application;

import java.util.*;
import java.util.Map;
import java.util.Map.Entry;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;
import java.io.File; // Import the File class
import java.util.Scanner; // Import the Scanner class to read text files
import java.net.URL;
import java.text.DecimalFormat;

public class HistogramAlphaBet {
    Map<Character, Integer> characterFreq = new HashMap<Character, Integer>();
    Map<Character, Double> characterFract = new HashMap<Character, Double>();
    String finalS = "";
    Integer sum = 0;

    class MyPieChart {
        Map<Character, Slice> slices = new HashMap<Character, Slice>();
        int topN = 0;
        double startAngle = 0;
        Slice s;
        void getSlices(int n) {
            Map.Entry<Character, Double> entryWithMaxPercentage = null;
            for (MyColor color : MyColor.values()) {
                //
                entryWithMaxPercentage = null;
                for (Entry<Character, Double> entry :
characterFract.entrySet()) {
                    if (entryWithMaxPercentage == null
                            ||
entry.getValue().compareTo(entryWithMaxPercentage.getValue()) > 0) {
                        entryWithMaxPercentage = entry;
                    }
                }
                //
System.out.println(entryWithMaxPercentage.getValue()*sum);
                System.out.println(entryWithMaxPercentage.getKey());

                characterFract.remove(entryWithMaxPercentage.getKey());
                s = new Slice(300, 300, 300, 300, startAngle, startAngle +
(360 * entryWithMaxPercentage.getValue()),
                        ArcType.ROUND);
                s.setFill(color);
                slices.put(entryWithMaxPercentage.getKey(), s);
                startAngle += (360 * entryWithMaxPercentage.getValue());
                topN++;
                if (topN == n || characterFract.size()==0) {
                    s = new Slice(300, 300, 300, 300, startAngle, 360,
ArcType.ROUND);
                    s.setFill(MyColor.AliceBlue);
                    slices.put('*', s);
                    break;
                }
            }

        }

        void draw(GraphicsContext gc) {
            DecimalFormat f = new DecimalFormat("#.###");
            Double pointSlice = (double) 0;
            for (Entry<Character, Slice> entry : slices.entrySet()) {
                pointSlice = entry.getValue().endAngle -
(entry.getValue().angleSlice / 2);
                entry.getValue().draw(gc);
                System.out.println(entry.getValue().angleSlice / 360);
                System.out.println(entry.getKey());
                if (pointSlice <= 90) {
                    gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                            (150 *
Math.cos(Math.toRadians(pointSlice))) + 450,
                            -(150 *
Math.sin(Math.toRadians(pointSlice))) + 450);

                } else if (pointSlice <= 180) {
```

```
                                 gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                                              -(150 *
Math.sin(Math.toRadians(pointSlice - 90))) + 450 - 30,
                                              -(150 *
Math.cos(Math.toRadians(pointSlice - 90))) + 450 - 20);

                          }

                          else if (pointSlice <= 270) {
                                 gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                                              -(150 *
Math.cos(Math.toRadians(pointSlice - 180))) + 450 - 20,
                                              (150 *
Math.sin(Math.toRadians(pointSlice - 180))) + 450 + 20);

                          } else if (pointSlice <= 360) {
                                 gc.strokeText(entry.getKey() + ", " +
f.format(entry.getValue().angleSlice / 360),
                                              (150 *
Math.sin(Math.toRadians(pointSlice - 270))) + 450 + 10,
                                              (150 *
Math.cos(Math.toRadians(pointSlice - 270))) + 450 + 10);

                          }
                    }

             }
       }

       public void setFreq(String filename) throws Exception {

             URL path = HistogramAlphaBet.class.getResource(filename);
             File myObj = new File(path.getFile());
             Scanner myReader = new Scanner(myObj);
             while (myReader.hasNextLine()) {
                    finalS += myReader.nextLine().replaceAll("[^a-zA-Z]",
"").toLowerCase();
             }
             myReader.close();
             for (int i = 0; i < finalS.length(); i++) {
                    if (characterFreq.containsKey(finalS.charAt(i))) {
                          characterFreq.put(finalS.charAt(i),
characterFreq.get(finalS.charAt(i)) + 1);
                    } else {
                          characterFreq.put(finalS.charAt(i), 1);
                    }
                    // Print current character finalS.charAt(i)

             }
       }

       public void setFract() {
             for (Entry<Character, Integer> entry : characterFreq.entrySet()) {
```

```java
            sum += entry.getValue();
        }

        for (Entry<Character, Integer> entry : characterFreq.entrySet()) {
            characterFract.put(entry.getKey(), (double) entry.getValue() /
sum);
        }

    }

    public void printFinal() {
        System.out.println(finalS);
    }
}
```

Class MyRectangle

```java
package application;

import javafx.scene.canvas.GraphicsContext;

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

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

    @Override
    double getWidth() {
        return width;
    }

    @Override
    double getHeight() {
        return height;
    }

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

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

    @Override
    void draw(GraphicsContext gc) {
        // TODO
        gc.strokeRect(p.getX(), p.getY(), this.width, this.height);
        gc.setFill(fill.myColor());
        gc.fillRect(p.getX(), p.getY(), width, height);
    }

    public boolean pointInMyShape(double x, double y) {

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

    }


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

}


/*
```

Interface MyShapeInterface

```
package application;

import java.util.ArrayList;


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


public interface MyShapeInterface {

    abstract MyRectangle getMyBoundingRectangle();

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

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

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

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

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

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

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

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

    }

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

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

    abstract boolean pointInMyShape(double x, double y);
    static ArrayList<MyPoint> intersectMyShapes(MyRectangle R, MyShape S1, MyShape
S2) {
        ArrayList<MyPoint> List = new ArrayList<MyPoint>();

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

        return List;
    }

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

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

Class MyOval

```
package application;

import javafx.scene.canvas.GraphicsContext;

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

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

    @Override
    double getWidth() {
        return width;
    }
```

```
       @Override
       double getHeight() {
             return height;
       }

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

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

       // getX, getY, getA, getB

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

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

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

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

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

       @Override
```

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

Class MyCircle

```
package application;

public class MyCircle extends MyOval {

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

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

	public boolean pointInMyShape(double x, double y) {

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

	}

}
```

Class MyColor

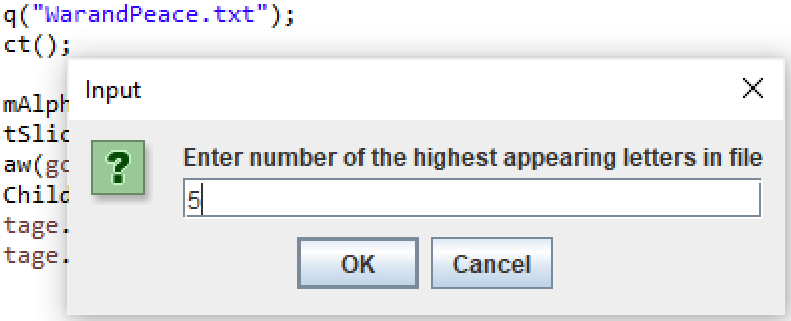
```
package application;

import javafx.scene.paint.Color;

public enum MyColor {
	RED(255, 0, 0, 1),
	GREEN(0, 255, 0, 1),
	BLUE(0, 0, 255, 1),
	DARKKHAKI(189, 183, 107, 1),
	DARKORANGE(255, 140, 0, 1),
	DARKORCHID(99,32,204,1),
	FUCHSIA(255, 0, 255, 1),
	GOLD(255, 215, 0, 1),
	GoldenRod(218, 165, 32, 1),
	Lavender(230, 230, 250, 1),
	LightBlue(173, 216, 230, 1),
	LightCoral(240, 128, 128, 1),
	MediumAquaMarine(102, 205, 170, 1),
```

```java
        Olive(128, 128, 0, 1),
        PaleGreen(152, 251, 152, 1),
        PALEVIOLETRED(219, 112, 147,1),
        YELLOWGREEN(154, 205, 50,1),
        LIGHTSALMON(255, 160, 122,1),
        LIGHTGOLDENRODYELLOW(250, 250, 210,1),
        DEEPPINK(255, 20, 147,1),
        DARKSALMON(233, 150, 122,1),
        CYAN(0, 255, 255,1),
        CORNFLOWERBLUE(100, 149, 237,1),
        BEIGE(245, 245, 220,1),
        AZURE(240, 255, 255,1),
        AQUAMARINE(127, 255, 212,1),
        AliceBlue(240, 248, 255, 1),
        ;

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

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

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

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

Class Main

```
package application;

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

import javax.swing.JOptionPane;

public class Main extends Application {

        @Override
        public void start(Stage primaryStage) throws Exception {
                JFrame g = new JFrame();
                String num = JOptionPane.showInputDialog(g, "Enter number of the highest
appearing letters in file");
                int n = Integer.parseInt(num);
                primaryStage.setTitle("Drawing Operations Test");
                Group root = new Group();
                Canvas canvas = new Canvas(700, 700);
                GraphicsContext gc = canvas.getGraphicsContext2D();
                gc.setLineWidth(1);
                HistogramAlphaBet f = new HistogramAlphaBet();
                f.setFreq("WarandPeace.txt");
                f.setFract();

                HistogramAlphaBet.MyPieChart chart = f.new MyPieChart();
                chart.getSlices(n);
                chart.draw(gc);
                root.getChildren().add(canvas);
                primaryStage.setScene(new Scene(root));
                primaryStage.show();

        }

}
```

[4] Outputs produced for the tasks indicated

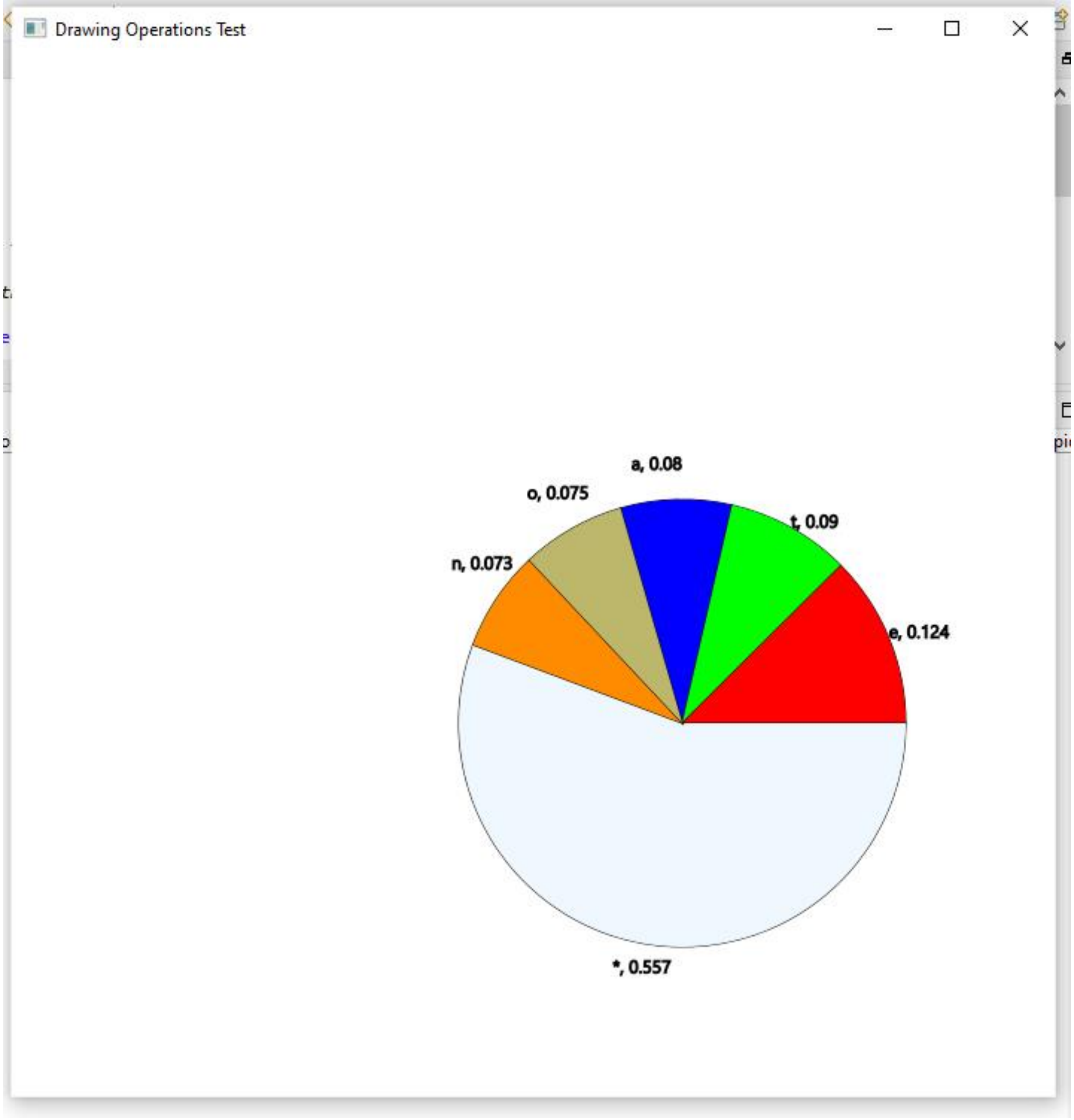


```
q("WarandPeace.txt");
ct();

mAlph
tSlic
aw(gc
Chilc
tage.
tage.
```

Input

Enter number of the highest appearing letters in file

5



Drawing Operations Test
a, 0.08
o, 0.075
t, 0.09
n, 0.073
e, 0.124
*, 0.557