CSC 22100: Exercise 3

July 19, 2018

Contents

List of Figures

1 Part 1

1.1 Instructions

Implement a Java class PieChart that displays a pie chart of the probabilities of the n most frequent occurrences of an event to be specified in part 3 of the exercise. The probability of event is given by the equation:

$$Probability\ of\ event = \frac{Frequency\ of\ event}{\Sigma Frequencies\ of\ all\ events} \tag{1}$$

In the pie chart:
i. The area of each segment is proportional to the probability of the corresponding event:

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

ii. Each segment has a different color;
iii. Each segment has a legend showing the event and its probability;
iv. The last segment represents ?All Other Events? and their cumulative probability. For 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;

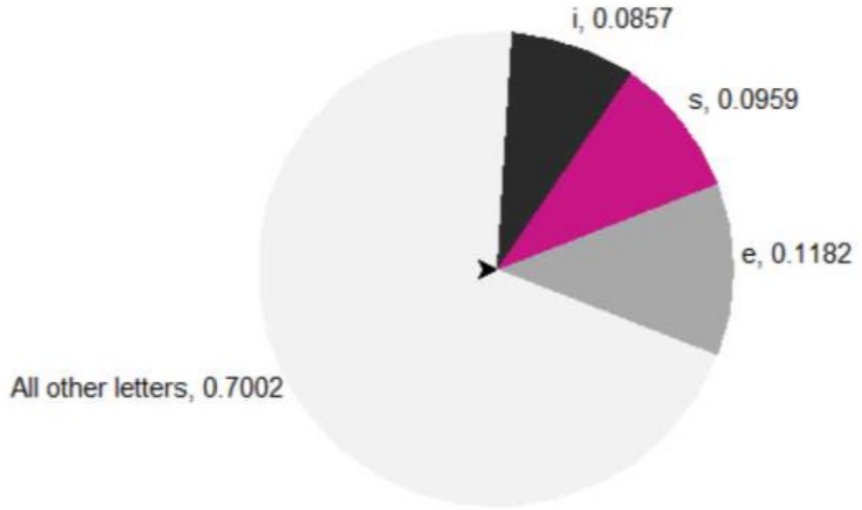

Figure 1: ShapePositionInterface Part2

1.2 Solution Method

The implementation for the PieChart class is shown in figures 2 to 5. Lines 6 - 17 the private members n, probabilityOfEvents, centralAngleOfSegments, typesOfEvents, and colorList are declared. n refers to the number of different events that the pie chart will include, probabilityOfEvents is an array of doubles containing the probability for each event as defined in equation 1, and centralAngleOfSegments is an array containing the angles which will be used to draw the pie chart. typesOfEvents is the array containing the different events will might occur, in the case of the Histogram, it includes the 26 letters of the english alphabet.

It is important to realize that the PieChart class extends the class Circle, which extends the class Shape. That's why in the constructor in lines 20 - 26 the keyword super is used to properly initiate the member variables of its superclasses. Method calculateCentralAngleOfSegment() is included in lines 28 - 34, this uses the equation 2 to calculate the angle of each event.

Finally, the draw() and addLegend() methods were implemented in lines 36 - 66, and 68 - 118, respectively. The draw method starts by calling the method calculateCentralAngleOfSegment to calculate the angles for each event. First, a for loop draw the arcs for the first 20th most frequent letters with different colors. The six less frequent letters are all drawn with the same color to avoid crowding the legends on the pie chart drawing. The addLegend() process all the events in a similar fashion than the draw method. This however uses a set of if - else statements to properly locate the place of the legends regarding their quadrant on the canvas. At the end of the PieChar class, a sort method is included which was used to sort the probabilities of letters in decreasing order.

1.3 Code Developed

```
1   import javafx.scene.canvas.GraphicsContext;
2   import javafx.scene.paint.Color;
3   import javafx.scene.shape.ArcType;
4
5   public class PieChart extends Circle {
6       private int n;    // number of events
7       private double[] probabilityOfEvents;
8       private double[] centralAngleOfSegments;
9       private char[] typesOfEvents;
10      private Color[] colorList = {Color.BLUE, Color.BROWN,Color.PURPLE,
11              Color.ORANGE,Color.STEELBLUE,Color.DARKVIOLET,Color.LIGHTCORAL,
12              Color.DARKSALMON,Color.DARKGOLDENROD, Color.DARKOLIVEGREEN,
13              Color.DARKBLUE,Color.DARKSLATEGRAY,Color.LAWNGREEN,
14              Color.DARKORANGE,Color.VIOLET,Color.BLUEVIOLET,Color.PURPLE,
15              Color.BLACK,Color.CHOCOLATE,Color.DARKGREY, Color.DARKOLIVEGREEN,
16              Color.DARKKHAKI,Color.DARKORANGE,Color.CORNSILK,Color.DARKMAGENTA,
17              Color.BLANCHEDALMOND};
18
19      // constructor
20      public PieChart(double x, double y, Color strokeColor, double radius,int n,
21                      double[] probabilityOfEvents, char[] typesOfEvents){
22          super(x,y,strokeColor,radius);
23          this.n = n;
24          this.probabilityOfEvents = probabilityOfEvents;
25          this.typesOfEvents = typesOfEvents;
26      }
27
28      private void calculateCentralAngleOfSegment(){
```

Figure 2: pieChartPart1

```
28      private void calculateCentralAngleOfSegment(){
29          centralAngleOfSegments = new double[n];
30          sort(probabilityOfEvents);
31          for(int i = 0; i < n; i++) {
32              centralAngleOfSegments[i] = Math.toDegrees(2 * Math.PI * probabilityOfEvents[i]);
33          }
34      }
35
36      // draw method
37      @Override
38      public void draw(GraphicsContext gc, double width, double height) {
39          calculateCentralAngleOfSegment();
40
41          double startAngle = 90;
42          for (int i = 0; i < 20; i++){
43              // Fill arc
44              gc.setFill(colorList[i]);
45              gc.strokeArc( x: getX() - getRadius(), y: getY()-getRadius(), w: getRadius()*2,
46                      h: getRadius()*2,startAngle,-centralAngleOfSegments[i],ArcType.ROUND);
47              gc.fillArc( x: getX() - getRadius(), y: getY()-getRadius(), w: getRadius()*2,
48                      h: getRadius()*2,startAngle,-centralAngleOfSegments[i],ArcType.ROUND);
49
50
51              // update startAngle for next event
52              startAngle = startAngle -  centralAngleOfSegments[i];
53          }
54
55          for (int i = 20; i < 26; i++) {
56              // Fill arc
57              gc.setFill(colorList[20]);
58              gc.fillArc( x: getX() - getRadius(),  y: getY() - getRadius(),  w: getRadius() * 2,
59                      h: getRadius() * 2, startAngle, -centralAngleOfSegments[i], ArcType.ROUND);
60
61              // update startAngle for next event
62              startAngle = startAngle - centralAngleOfSegments[i];
63          }
64
65
66      }
```

Figure 3: pieChartPart2

```
68      public  void addLegend(GraphicsContext gc, double width, double height){
69          calculateCentralAngleOfSegment();
70
71          gc.setFont(new javafx.scene.text.Font( size: 7));
72
73          double startAngle = 90;
74
75          for (int i = 0; i < 20; i++){
76
77              if(probabilityOfEvents[i] < 0.01 ){
78                  continue;
79              }
80              gc.setFill(colorList[i]);
81              if(startAngle >= 0 && startAngle <=90){
82                  gc.fillText( text: String.valueOf(typesOfEvents[i]) +": " +
83                                  String.valueOf(Math.round(probabilityOfEvents[i]*10000.0)/10000.0),
84                          x: width/2 + 1.1*getRadius()*Math.cos(Math.toRadians(startAngle)) + 2 ,
85                          y: height/2 - 1.1*getRadius()*Math.sin(Math.toRadians(startAngle))  - 2);
86              }
87              else if(startAngle >= -90 && startAngle <= 0){
88                  gc.fillText( text: String.valueOf(typesOfEvents[i]) +": " +
89                                  String.valueOf(Math.round(probabilityOfEvents[i]*10000.0)/10000.0),
90                          x: width/2 + 1.1*getRadius()*Math.cos(Math.toRadians(startAngle)) + 2 ,
91                          y: height/2 - 1.1*getRadius()*Math.sin(Math.toRadians(startAngle)) + 2);
92              }
93              else if(startAngle >= -180 && startAngle <=-90){
94                  gc.fillText( text: String.valueOf(typesOfEvents[i]) +": " +
95                                  String.valueOf(Math.round(probabilityOfEvents[i]*10000.0)/10000.0),
96                          x: width/2 + 1.2*getRadius()*Math.cos(Math.toRadians(startAngle ) )  ,
97                          y: height/2 - 1.2*getRadius()*Math.sin(Math.toRadians(startAngle )) );
98              }
99              else if (startAngle >= -270 && startAngle <=-180){
100                 gc.fillText( text: String.valueOf(typesOfEvents[i]) +": " +
101                                 String.valueOf(Math.round(probabilityOfEvents[i]*10000.0)/10000.0),
102                         x: width/2 + 1.2*getRadius()*Math.cos(Math.toRadians(startAngle ) )  ,
103                         y: height/2 - 1.2*getRadius()*Math.sin(Math.toRadians(startAngle )) );
104             }
105             startAngle = startAngle -  (centralAngleOfSegments[i]);
```

Figure 4: pieChartPart3

```
106         }
107
108         gc.setFill(colorList[20]);
109         double probabilityOfEventsRestOfEvents = 0;
110         for(int i = 20; i <= 25;i++) {
111             probabilityOfEventsRestOfEvents += probabilityOfEvents[i];
112         }
113         gc.fillText( text: "Rest of Events" + ": " +
114                         String.valueOf(Math.round(probabilityOfEventsRestOfEvents * 10000.0) / 10000.0),
115                 x: width / 2 + 1.2 * getRadius() * Math.cos(Math.toRadians(startAngle)),
116                 y: height / 2 - 1.2 * getRadius() * Math.sin(Math.toRadians(startAngle)));
117
118     }
119
120     public void sort(double A[]) {
121         int n = A.length;
122         for (int i=1; i<n; ++i)
123         {
124             double key = A[i];
125             int j = i-1;
126
127             while (j>=0 && A[j] < key)
128             {
129                 A[j+1] = A[j];
130                 j = j-1;
131             }
132             A[j+1] = key;
133         }
134     }
135
136 }
137
```

Figure 5: pieChartPart4

2 Part 2

2.1 Instructions

The PieChart class includes appropriate constructors and a method draw that draws the pie chart. The drawing panel should include appropriate GUI components to input the number of events, n, and display the pie chart together with the events probabilities. You may amend and use the class hierarchy in previous exercises, but in any case you may only use your own classes and methods for the operations included.

2.2 Solution Method

As mentioned previously, the Shape, and Circle classes are the super classes of PieChart. Figures 6 - 8 illustrate the code for both classes, including the declaration of their private members, constructors, set and get methods, as well as the toString and draw methods.

Two additional classes were required to incorporate the drawing panel with the GUI components. First, figures 9 - 10 shows the DrawPieChartController class which included all of the fxml components that allow the user to input the number of events to be drawn in the pie chat. The three components were the TextField numberEventsTextField which is used to obtain the value of n from the user, the canvas, where the shapes are drawn, and the drawShapesButtonPressed, which is the method with an ActionEvent as an arguments allowing the statements in its body to have any effect on the canvas. The rest of this class will be discussed in part 3. Finally, the class DrawPieChart in figure 11 loads the PieChart.fxml file and launches the application.

2.3 Code Developed

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public class Shape {

    // instance variable
    private double x;
    private double y;
    private Color strokeColor;

    //constructor
    Shape(double x, double y, Color strokeColor){
        this.x = x;
        this.y = y;
        this.strokeColor = strokeColor;
    }

    // get methods
    double getX(){ return this.x; }

    double getY() { return this.y; }

    Color getColor() { return this.strokeColor; }

    // set methods
    public void setX(double x){ this.x  = x;
    }

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

    public void setColor(Color strokeColor) { this.strokeColor  = strokeColor; }


    // shift methods
    public void shiftX(double deltaX) { this.x = this.x + deltaX; }

    public void shiftY(double deltaY) { this.y = this.y + deltaY; }
```

Figure 6: shape1

```java
    // toString method
    @Override
    public String toString(){
        return String.format("%s: %n%s %.1f%n%s %.1f%n%s%s",
                "shape","x:",getX(),"y:",getY(),"color: ",getColor());
    }

    // draw method
    public void draw(GraphicsContext gc, double canvasWidth, double canvasHeight){
        // set the color for the current arc
        gc.setFill(strokeColor);

        gc.fillRect( x: x - canvasWidth/2, y: y - canvasHeight/2,canvasWidth,canvasHeight);
    }
}
```

Figure 7: shape2

```java
import javafx.scene.paint.Color;
import javafx.scene.canvas.GraphicsContext;

public class Circle extends Shape {
    private double radius;
    // constructor
    public Circle(double x, double y, Color strokeColor, double radius){
        super(x,y,strokeColor);
        this.radius = radius;
    }
    // area method
    public double getArea() {
        return Math.PI * radius * radius;
    }
    // perimeter method
    public double getPerimeter(){
        return 2*Math.PI*radius;
    }

    // return radius method
    public double getRadius(){ return this.radius; }

    // toString method
    @Override
    public String toString(){
       return String.format("%s %s%n%s%.1f%n%s%.1f", "circle",
               super.toString(),"radius: ",radius,"area: ", getArea());
    }
    // draw method
    @Override
    public void draw(GraphicsContext gc,double width, double height) {
        // set the color for the current arc
        gc.setFill(super.getColor());

        // Draw circle
        gc.strokeOval( x: super.getX() - width/2, y: super.getY()- height/2, w: 2*radius, h: 2*radius);
    }
}
```

Figure 8: circle1

```java
import javafx.fxml.FXML;
import javafx.scene.canvas.Canvas;
import javafx.event.ActionEvent;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.control.TextField;


public class DrawPieChartController {

    @FXML
    private TextField numberEventsTextField;
    @FXML
    private Canvas canvas;
    @FXML
    void drawShapesButtonPressed(ActionEvent event) {
        GraphicsContext gc = canvas.getGraphicsContext2D();

        double centerY = canvas.getWidth()/2;
        double centerX = canvas.getHeight()/2;
        double canvasWidth = canvas.getWidth();
        double canvasHeight = canvas.getHeight();


            int n = Integer.parseInt(numberEventsTextField.getText());
            // Polygon and circle 1
            Color color1 = Color.LIGHTGREEN;
            Color color2 = Color.LIGHTBLUE;
            double radius1 = canvasHeight / 3;

            HistogramLetters EmmaHL = new HistogramLetters();
```

Figure 9: DrawPieChartControllerPart1

```
32
33              EmmaHL.fileProcessor();
34              EmmaHL.printLettersCount();
35              EmmaHL.calculateProbabilityOfEvents();
36
37              EmmaHL.printProbabilityOfEvents();
38
39              Circle circle1 = new Circle(centerX, centerY, color2, radius1);
40
41              System.out.println("Number of events input from GUI: " + numberEventsTextField.getText());
42              circle1.draw(gc,  width: 2 * radius1,  height: 2 * radius1);
43
44              PieChart PieChart1 = new PieChart(centerX, centerY, Color.BLUE, radius1, n,
45                      EmmaHL.calculateProbabilityOfEvents(), EmmaHL.getTypesOfEvents());
46              PieChart1.draw(gc, canvasWidth, canvasHeight);
47              PieChart1.addLegend(gc, canvasWidth, canvasHeight);
48
49
50      }
51  }
52
```

Figure 10: DrawPieChartControllerPart2

```
1   /.../
3   import ...
8
9   public class DrawPieChart extends Application {
10      @Override
11      public void start(Stage stage) throws Exception {
12          // loads Welcome.fxml and configures the DrawRandomLinesController
13          Parent root =
14            FXMLLoader.load(getClass().getResource( name: "PieChart.fxml"));
15
16          Scene scene = new Scene(root); // attach scene graph to scene
17          stage.setTitle("Draw Pie Chart"); // displayed in window's title bar
18          stage.setScene(scene); // attach scene to stage
19          stage.show(); // display the stage
20      }
21
22      public static void main(String[] args) {
23          launch(args);
24      }
25  }
26
27  4
```

Figure 11: DrawPieChart

3 Part 3

3.1 Instructions

Implement a Java class HistogramLetters that calculates the n most frequent letters in the file ?Emma.txt? and their probabilities. The HistogramLetters class utilizes the drawing panel above to draw a pie chart of the letter probabilities.

3.2 Solution Method

The HistogramLetters class is shown in figures 12 - 14. This class has the private members lettersCounter, probabilityOfEvents, and the englishAlphabet, and englishAlphabetCapital. The lettersCounter which is an array of integers representing the frequency of each letter in the book

”Emma”, and the probabilityOfEvents is an array of doubles with the probability of each letter calculated with the method getFrequencyOfAllEvents.

The method fileProcessor uses a while loop to read the text file ”Emma” character-by-character and employees the method identifyLetter which identifies a letter increments and increments it by 1. This process repeats itself until the end of the file. Methods printLettersCount and printProbabilityOfEvents are used to check the frequency and probability of each letter, respectively.

Finally, the HistogramLetters is used to instantiate an object ”EmmaHL” in the DrawPieChartController class in figure 10, which calls the methods fileProcessor, printLettersCount, calculateProbabilityOfEvents, and PrintProbability of events to double check the accuracy of the program. EmmaHL is also used to pass the probability of each event and the types of events to an object of PieChart class. The object ”PieChart1” is used to draw the PieChart illustrated in figure 15.

3.3 Code Developed

```
1   import java.io.FileNotFoundException;
2   import java.io.FileReader;
3   import java.io.IOException;
4
5   public class HistogramLetters {
6
7       private static int[] lettersCounter = new int[26];
8       private static double[] probabilityOfEvents  = new double[26];
9       private static char[] englishAlphabet = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
10              'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
11              's', 't', 'u', 'v', 'w', 'x', 'y', 'z'};
12      private static char[] englishAlphabetCapital = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
13              'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
14              'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};
15      private static double[] centralAngleOfSegments = new double[26];
16
17      public void fileProcessor() {
18          try {
19              FileReader myFileReader = new FileReader( fileName: "Emma.txt");
20              int singleCharacter = myFileReader.read();
21
22              while (singleCharacter != -1) {
23                  identifyLetter(singleCharacter);
24  //                System.out.print((char) singleCharacter);
25                  singleCharacter = myFileReader.read();
26              }
27          }
28          catch (FileNotFoundException exception1){
29              System.out.println("File was not found ");
30          }
31          catch(IOException exception2){
32              System.out.println("I/O error occured");
33          }
34      }
35
```

Figure 12: histogramLettersPart1

```
35
36      public void identifyLetter(int intCharacter){
37          char charCharacter = (char) intCharacter;
38
39          for(int i = 0; i < 26; i++){
40              if(charCharacter == englishAlphabet[i] ||
41                      charCharacter == englishAlphabetCapital[i]){
42                  lettersCounter[i] += 1;
43              }
44          }
45
46      }
47
48  @   private double getFrequencyOfAllEvents(){
49          int frequencyOfAllEvents = 0;
50
51          for(int i = 0 ; i < 26; i++){
52              frequencyOfAllEvents += lettersCounter[i];
53          }
54
55          return frequencyOfAllEvents;
56      }
57
58      public double[] calculateProbabilityOfEvents(){
59          for(int i = 0; i < 26; i++) {
60              probabilityOfEvents[i] = lettersCounter[i]/getFrequencyOfAllEvents();
61          }
62
63          return probabilityOfEvents;
64      }
65
66      public char[] getTypesOfEvents(){
67          return englishAlphabet;
68      }
69
```

Figure 13: histogramLettersPart2t

```
69
70
71      public void printLettersCount(){
72          System.out.println();
73          System.out.println();
74          for(int i = 0; i < 26;i++){
75              System.out.print(englishAlphabet[i] + ": #");
76              System.out.println(lettersCounter[i]);
77          }
78      }
79
80      public void printProbabilityOfEvents(){
81          System.out.println();
82          System.out.println();
83          for(int i = 0; i < 26;i++){
84              System.out.print(englishAlphabet[i] + ": #");
85              System.out.println(probabilityOfEvents[i]);
86          }
87      }
88
89
90  }
91
```

Figure 14: histogramLettersPart3

4 Results

4.1 Pie Chart for letters of Emma Book

Figures 15 , 16 , and 17 show the output results for the pieChart, the number of frequencies and probabilities of each events, respectively. The output pie chart for the histogram of Emma book in figure 15 shows the probability of the letters in this book. They are alphabetically order in the clockwise direction, starting with the probability for the letter a 90 degrees and finishing with the probability for the letter t, the rest probability for the rest of letters are represented in the region with the legend ”Rest of Events: 0.0905.

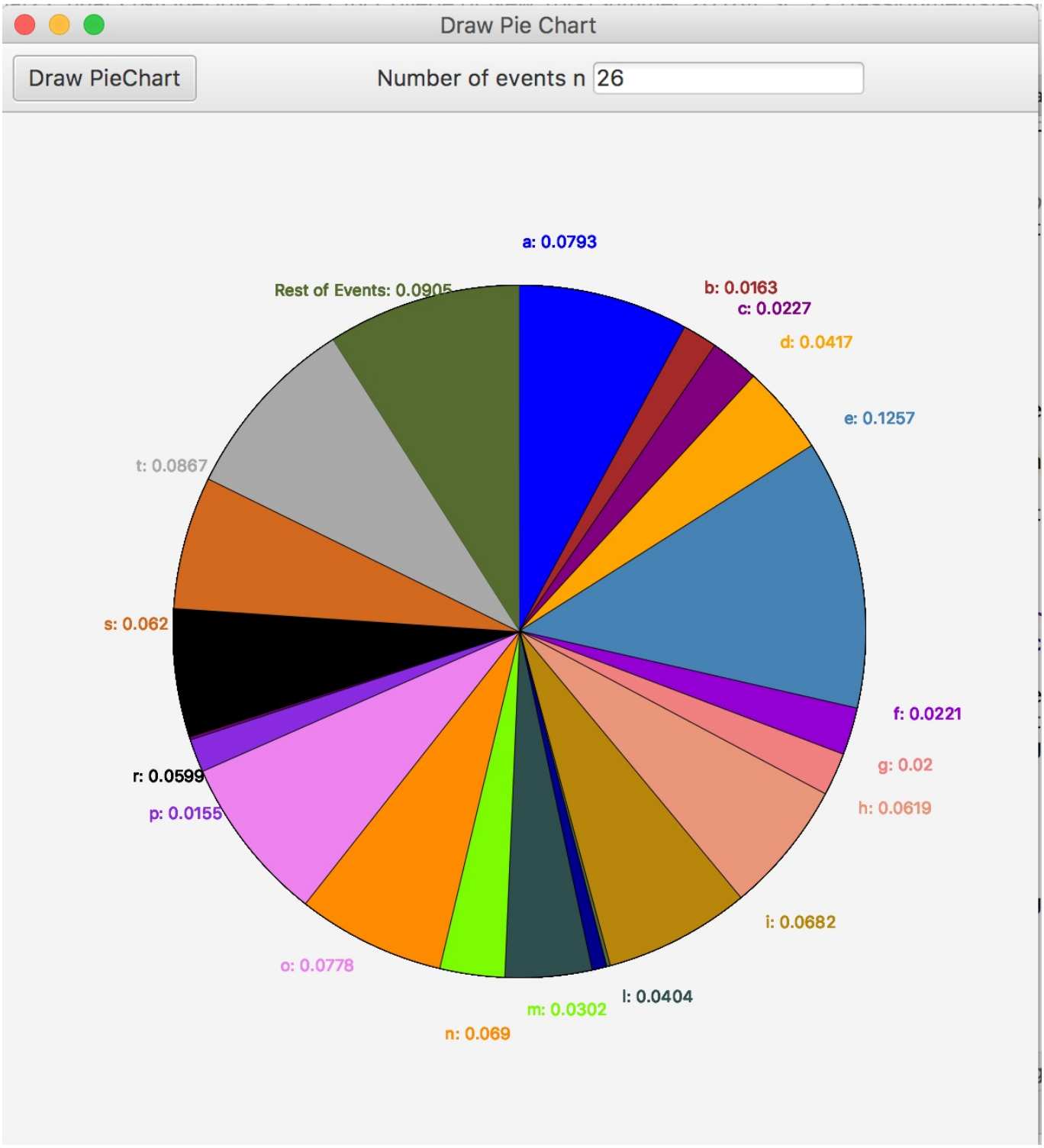

Figure 15: resultPieChart

4.2 Output for letters of Emma Book

The frequency for each of the letters of the english alphabet in the book Emma are shown in figure 16, this was the output from the program when the Histogram method printLettersCount() was used.

```
Number of events input from GUI: 26


a: #384188
b: #78841
c: #110096
d: #202090
e: #609203
f: #107261
g: #96824
h: #299733
i: #330204
j: #8092
k: #33572
l: #195797
m: #146419
n: #334411
o: #377139
p: #75145
q: #6447
r: #290171
s: #300440
t: #420245
u: #146370
v: #54698
w: #114639
x: #10276
y: #111398
z: #1260
```

Figure 16: Number of Repetition of each letter

4.3 Output for possibilities of letters of Emma Book

The possibility for each of the letters of the english alphabet in the book Emma are shown in figure 17 , this was the output from the program when the Histogram method printProbabilityOfEvents was used. As one can observe, these probability values are indeed the same than the ones presented in the PieChart, showing the correctness of he addLegend() methods of the PieChart class.

```
a: #0.0792964398666738
b: #0.01627278992453806
c: #0.022723824907496637
d: #0.04171139528734918
e: #0.12573955734197131
f: #0.022138680636925926
g: #0.019984482840824866
h: #0.06186491980633892
i: #0.068154137114473
j: #0.0016701895722956582
k: #0.006929264004091676
l: #0.04041251948674901
m: #0.030220895574142113
n: #0.06902246231598658
o: #0.07784152559392143
p: #0.015509935171794024
q: #0.0013306614152978383
r: #0.05989132209374733
s: #0.062010844673814576
t: #0.0867386081079324
u: #0.030210781969465583
v: #0.011289672420344527
w: #0.02366150054107785
x: #0.002120967380735317
y: #0.022992557831758742
z: #2.600641202536492E-4
Number of events input from GUI: 26
```

Figure 17: Probabilities of each Letter

5 References

[1] Deitel, Paul J., and Harvey M. Deitel. Java: How to Program Early Objects. Pearson, 2018.