

# CSC 22100: Exercise 3

July 19, 2018

## Contents

<b>1</b>	<b>Part 1</b>	<b>3</b>
1.1	Instructions . . . . .	3
1.2	Solution Method . . . . .	3
1.3	Code Developed . . . . .	4
<b>2</b>	<b>Part 2</b>	<b>7</b>
2.1	Instructions . . . . .	7
2.2	Solution Method . . . . .	7
2.3	Code Developed . . . . .	8
<b>3</b>	<b>Part 3</b>	<b>10</b>
3.1	Instructions . . . . .	10
3.2	Solution Method . . . . .	10
3.3	Code Developed . . . . .	11
<b>4</b>	<b>Results</b>	<b>13</b>
4.1	Pie Chart for letters of Emma Book . . . . .	13
4.2	Output for letters of Emma Book . . . . .	14
4.3	Output for possibilities of letters of Emma Book . . . . .	15
<b>5</b>	<b>References</b>	<b>17</b>

## List of Figures

1	ShapePositionInterface Part2 . . . . .	3
2	pieChartPart1 . . . . .	4
3	pieChartPart2 . . . . .	5
4	pieChartPart3 . . . . .	6
5	pieChartPart4 . . . . .	6
6	shape1 . . . . .	8
7	shape2 . . . . .	8
8	circle1 . . . . .	9
9	DrawPieChartControllerPart1 . . . . .	9
10	DrawPieChartControllerPart2 . . . . .	10
11	DrawPieChart . . . . .	10
12	histogramLettersPart1 . . . . .	11
13	histogramLettersPart2t . . . . .	12
14	histogramLettersPart3 . . . . .	12
15	resultPieChart . . . . .	14
16	Number of Repetition of each letter . . . . .	15
17	Probabilities of each Letter . . . . .	16

# 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:

$$\text{Probability of event} = \frac{\text{Frequency of event}}{\sum \text{Frequencies of all events}} \quad (1)$$

In the pie chart:

- i. The area of each segment is proportional to the probability of the corresponding event:

$$\text{Probability of event} = \frac{\text{Central angle of segment}}{2\pi} \quad (2)$$

- 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;

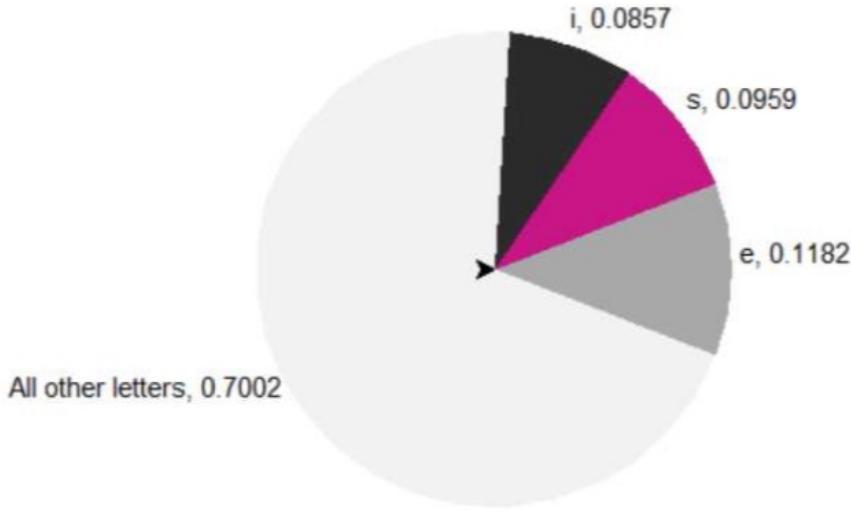


Figure 1: ShapePositionInterface Part2

## 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 initialize 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.CORN SILK, 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    /**
29     * @param probabilityOfEvents the probability of events
30     */
31    public void setProbabilityOfEvents(double[] probabilityOfEvents) {
32        this.probabilityOfEvents = probabilityOfEvents;
33    }
34
35    /**
36     * @param typesOfEvents the types of events
37     */
38    public void setTypesOfEvents(char[] typesOfEvents) {
39        this.typesOfEvents = typesOfEvents;
40    }
41
42    /**
43     * @param radius the radius of the pie chart
44     */
45    public void setRadius(double radius) {
46        this.radius = radius;
47    }
48
49    /**
50     * @param strokeColor the stroke color of the pie chart
51     */
52    public void setStrokeColor(Color strokeColor) {
53        this.strokeColor = strokeColor;
54    }
55
56    /**
57     * @param x the x coordinate of the center of the pie chart
58     */
59    public void setX(double x) {
60        this.x = x;
61    }
62
63    /**
64     * @param y the y coordinate of the center of the pie chart
65     */
66    public void setY(double y) {
67        this.y = y;
68    }
69
70    /**
71     * @param arcType the arc type of the pie chart
72     */
73    public void setArcType(ArcType arcType) {
74        this.arcType = arcType;
75    }
76
77    /**
78     * @param centralAngleOfSegments the central angle of segments
79     */
80    public void setCentralAngleOfSegments(double[] centralAngleOfSegments) {
81        this.centralAngleOfSegments = centralAngleOfSegments;
82    }
83
84    /**
85     * @param n the number of events
86     */
87    public void setN(int n) {
88        this.n = n;
89    }
90
91    /**
92     * @param colorList the color list of the pie chart
93     */
94    public void setColorList(Color[] colorList) {
95        this.colorList = colorList;
96    }
97
98    /**
99     * @param strokeDash the stroke dash of the pie chart
100    */
101    public void setStrokeDash(double[] strokeDash) {
102        this.strokeDash = strokeDash;
103    }
104
105    /**
106     * @param fill the fill of the pie chart
107     */
108    public void setFill(Color fill) {
109        this.fill = fill;
110    }
111
112    /**
113     * @param radius the radius of the pie chart
114     */
115    public void setRadius() {
116        this.radius = radius;
117    }
118
119    /**
120     * @param strokeColor the stroke color of the pie chart
121     */
122    public void setStrokeColor() {
123        this.strokeColor = strokeColor;
124    }
125
126    /**
127     * @param x the x coordinate of the center of the pie chart
128     */
129    public void setX() {
130        this.x = x;
131    }
132
133    /**
134     * @param y the y coordinate of the center of the pie chart
135     */
136    public void setY() {
137        this.y = y;
138    }
139
140    /**
141     * @param arcType the arc type of the pie chart
142     */
143    public void setArcType() {
144        this.arcType = arcType;
145    }
146
147    /**
148     * @param centralAngleOfSegments the central angle of segments
149     */
150    public void setCentralAngleOfSegments() {
151        this.centralAngleOfSegments = centralAngleOfSegments;
152    }
153
154    /**
155     * @param n the number of events
156     */
157    public void setN() {
158        this.n = n;
159    }
160
161    /**
162     * @param colorList the color list of the pie chart
163     */
164    public void setColorList() {
165        this.colorList = colorList;
166    }
167
168    /**
169     * @param strokeDash the stroke dash of the pie chart
170     */
171    public void setStrokeDash() {
172        this.strokeDash = strokeDash;
173    }
174
175    /**
176     * @param fill the fill of the pie chart
177     */
178    public void setFill() {
179        this.fill = fill;
180    }
181
182    /**
183     * @param radius the radius of the pie chart
184     */
185    public void setRadius() {
186        this.radius = radius;
187    }
188
189    /**
190     * @param strokeColor the stroke color of the pie chart
191     */
192    public void setStrokeColor() {
193        this.strokeColor = strokeColor;
194    }
195
196    /**
197     * @param x the x coordinate of the center of the pie chart
198     */
199    public void setX() {
200        this.x = x;
201    }
202
203    /**
204     * @param y the y coordinate of the center of the pie chart
205     */
206    public void setY() {
207        this.y = y;
208    }
209
210    /**
211     * @param arcType the arc type of the pie chart
212     */
213    public void setArcType() {
214        this.arcType = arcType;
215    }
216
217    /**
218     * @param centralAngleOfSegments the central angle of segments
219     */
220    public void setCentralAngleOfSegments() {
221        this.centralAngleOfSegments = centralAngleOfSegments;
222    }
223
224    /**
225     * @param n the number of events
226     */
227    public void setN() {
228        this.n = n;
229    }
230
231    /**
232     * @param colorList the color list of the pie chart
233     */
234    public void setColorList() {
235        this.colorList = colorList;
236    }
237
238    /**
239     * @param strokeDash the stroke dash of the pie chart
240     */
241    public void setStrokeDash() {
242        this.strokeDash = strokeDash;
243    }
244
245    /**
246     * @param fill the fill of the pie chart
247     */
248    public void setFill() {
249        this.fill = fill;
250    }
251
252    /**
253     * @param radius the radius of the pie chart
254     */
255    public void setRadius() {
256        this.radius = radius;
257    }
258
259    /**
260     * @param strokeColor the stroke color of the pie chart
261     */
262    public void setStrokeColor() {
263        this.strokeColor = strokeColor;
264    }
265
266    /**
267     * @param x the x coordinate of the center of the pie chart
268     */
269    public void setX() {
270        this.x = x;
271    }
272
273    /**
274     * @param y the y coordinate of the center of the pie chart
275     */
276    public void setY() {
277        this.y = y;
278    }
279
280    /**
281     * @param arcType the arc type of the pie chart
282     */
283    public void setArcType() {
284        this.arcType = arcType;
285    }
286
287    /**
288     * @param centralAngleOfSegments the central angle of segments
289     */
290    public void setCentralAngleOfSegments() {
291        this.centralAngleOfSegments = centralAngleOfSegments;
292    }
293
294    /**
295     * @param n the number of events
296     */
297    public void setN() {
298        this.n = n;
299    }
300
301    /**
302     * @param colorList the color list of the pie chart
303     */
304    public void setColorList() {
305        this.colorList = colorList;
306    }
307
308    /**
309     * @param strokeDash the stroke dash of the pie chart
310     */
311    public void setStrokeDash() {
312        this.strokeDash = strokeDash;
313    }
314
315    /**
316     * @param fill the fill of the pie chart
317     */
318    public void setFill() {
319        this.fill = fill;
320    }
321
322    /**
323     * @param radius the radius of the pie chart
324     */
325    public void setRadius() {
326        this.radius = radius;
327    }
328
329    /**
330     * @param strokeColor the stroke color of the pie chart
331     */
332    public void setStrokeColor() {
333        this.strokeColor = strokeColor;
334    }
335
336    /**
337     * @param x the x coordinate of the center of the pie chart
338     */
339    public void setX() {
340        this.x = x;
341    }
342
343    /**
344     * @param y the y coordinate of the center of the pie chart
345     */
346    public void setY() {
347        this.y = y;
348    }
349
350    /**
351     * @param arcType the arc type of the pie chart
352     */
353    public void setArcType() {
354        this.arcType = arcType;
355    }
356
357    /**
358     * @param centralAngleOfSegments the central angle of segments
359     */
360    public void setCentralAngleOfSegments() {
361        this.centralAngleOfSegments = centralAngleOfSegments;
362    }
363
364    /**
365     * @param n the number of events
366     */
367    public void setN() {
368        this.n = n;
369    }
370
371    /**
372     * @param colorList the color list of the pie chart
373     */
374    public void setColorList() {
375        this.colorList = colorList;
376    }
377
378    /**
379     * @param strokeDash the stroke dash of the pie chart
380     */
381    public void setStrokeDash() {
382        this.strokeDash = strokeDash;
383    }
384
385    /**
386     * @param fill the fill of the pie chart
387     */
388    public void setFill() {
389        this.fill = fill;
390    }
391
392    /**
393     * @param radius the radius of the pie chart
394     */
395    public void setRadius() {
396        this.radius = radius;
397    }
398
399    /**
400     * @param strokeColor the stroke color of the pie chart
401     */
402    public void setStrokeColor() {
403        this.strokeColor = strokeColor;
404    }
405
406    /**
407     * @param x the x coordinate of the center of the pie chart
408     */
409    public void setX() {
410        this.x = x;
411    }
412
413    /**
414     * @param y the y coordinate of the center of the pie chart
415     */
416    public void setY() {
417        this.y = y;
418    }
419
420    /**
421     * @param arcType the arc type of the pie chart
422     */
423    public void setArcType() {
424        this.arcType = arcType;
425    }
426
427    /**
428     * @param centralAngleOfSegments the central angle of segments
429     */
430    public void setCentralAngleOfSegments() {
431        this.centralAngleOfSegments = centralAngleOfSegments;
432    }
433
434    /**
435     * @param n the number of events
436     */
437    public void setN() {
438        this.n = n;
439    }
440
441    /**
442     * @param colorList the color list of the pie chart
443     */
444    public void setColorList() {
445        this.colorList = colorList;
446    }
447
448    /**
449     * @param strokeDash the stroke dash of the pie chart
450     */
451    public void setStrokeDash() {
452        this.strokeDash = strokeDash;
453    }
454
455    /**
456     * @param fill the fill of the pie chart
457     */
458    public void setFill() {
459        this.fill = fill;
460    }
461
462    /**
463     * @param radius the radius of the pie chart
464     */
465    public void setRadius() {
466        this.radius = radius;
467    }
468
469    /**
470     * @param strokeColor the stroke color of the pie chart
471     */
472    public void setStrokeColor() {
473        this.strokeColor = strokeColor;
474    }
475
476    /**
477     * @param x the x coordinate of the center of the pie chart
478     */
479    public void setX() {
480        this.x = x;
481    }
482
483    /**
484     * @param y the y coordinate of the center of the pie chart
485     */
486    public void setY() {
487        this.y = y;
488    }
489
490    /**
491     * @param arcType the arc type of the pie chart
492     */
493    public void setArcType() {
494        this.arcType = arcType;
495    }
496
497    /**
498     * @param centralAngleOfSegments the central angle of segments
499     */
500    public void setCentralAngleOfSegments() {
501        this.centralAngleOfSegments = centralAngleOfSegments;
502    }
503
504    /**
505     * @param n the number of events
506     */
507    public void setN() {
508        this.n = n;
509    }
510
511    /**
512     * @param colorList the color list of the pie chart
513     */
514    public void setColorList() {
515        this.colorList = colorList;
516    }
517
518    /**
519     * @param strokeDash the stroke dash of the pie chart
520     */
521    public void setStrokeDash() {
522        this.strokeDash = strokeDash;
523    }
524
525    /**
526     * @param fill the fill of the pie chart
527     */
528    public void setFill() {
529        this.fill = fill;
530    }
531
532    /**
533     * @param radius the radius of the pie chart
534     */
535    public void setRadius() {
536        this.radius = radius;
537    }
538
539    /**
540     * @param strokeColor the stroke color of the pie chart
541     */
542    public void setStrokeColor() {
543        this.strokeColor = strokeColor;
544    }
545
546    /**
547     * @param x the x coordinate of the center of the pie chart
548     */
549    public void setX() {
550        this.x = x;
551    }
552
553    /**
554     * @param y the y coordinate of the center of the pie chart
555     */
556    public void setY() {
557        this.y = y;
558    }
559
560    /**
561     * @param arcType the arc type of the pie chart
562     */
563    public void setArcType() {
564        this.arcType = arcType;
565    }
566
567    /**
568     * @param centralAngleOfSegments the central angle of segments
569     */
570    public void setCentralAngleOfSegments() {
571        this.centralAngleOfSegments = centralAngleOfSegments;
572    }
573
574    /**
575     * @param n the number of events
576     */
577    public void setN() {
578        this.n = n;
579    }
580
581    /**
582     * @param colorList the color list of the pie chart
583     */
584    public void setColorList() {
585        this.colorList = colorList;
586    }
587
588    /**
589     * @param strokeDash the stroke dash of the pie chart
590     */
591    public void setStrokeDash() {
592        this.strokeDash = strokeDash;
593    }
594
595    /**
596     * @param fill the fill of the pie chart
597     */
598    public void setFill() {
599        this.fill = fill;
600    }
601
602    /**
603     * @param radius the radius of the pie chart
604     */
605    public void setRadius() {
606        this.radius = radius;
607    }
608
609    /**
610     * @param strokeColor the stroke color of the pie chart
611     */
612    public void setStrokeColor() {
613        this.strokeColor = strokeColor;
614    }
615
616    /**
617     * @param x the x coordinate of the center of the pie chart
618     */
619    public void setX() {
620        this.x = x;
621    }
622
623    /**
624     * @param y the y coordinate of the center of the pie chart
625     */
626    public void setY() {
627        this.y = y;
628    }
629
630    /**
631     * @param arcType the arc type of the pie chart
632     */
633    public void setArcType() {
634        this.arcType = arcType;
635    }
636
637    /**
638     * @param centralAngleOfSegments the central angle of segments
639     */
640    public void setCentralAngleOfSegments() {
641        this.centralAngleOfSegments = centralAngleOfSegments;
642    }
643
644    /**
645     * @param n the number of events
646     */
647    public void setN() {
648        this.n = n;
649    }
650
651    /**
652     * @param colorList the color list of the pie chart
653     */
654    public void setColorList() {
655        this.colorList = colorList;
656    }
657
658    /**
659     * @param strokeDash the stroke dash of the pie chart
660     */
661    public void setStrokeDash() {
662        this.strokeDash = strokeDash;
663    }
664
665    /**
666     * @param fill the fill of the pie chart
667     */
668    public void setFill() {
669        this.fill = fill;
670    }
671
672    /**
673     * @param radius the radius of the pie chart
674     */
675    public void setRadius() {
676        this.radius = radius;
677    }
678
679    /**
680     * @param strokeColor the stroke color of the pie chart
681     */
682    public void setStrokeColor() {
683        this.strokeColor = strokeColor;
684    }
685
686    /**
687     * @param x the x coordinate of the center of the pie chart
688     */
689    public void setX() {
690        this.x = x;
691    }
692
693    /**
694     * @param y the y coordinate of the center of the pie chart
695     */
696    public void setY() {
697        this.y = y;
698    }
699
700    /**
701     * @param arcType the arc type of the pie chart
702     */
703    public void setArcType() {
704        this.arcType = arcType;
705    }
706
707    /**
708     * @param centralAngleOfSegments the central angle of segments
709     */
710    public void setCentralAngleOfSegments() {
711        this.centralAngleOfSegments = centralAngleOfSegments;
712    }
713
714    /**
715     * @param n the number of events
716     */
717    public void setN() {
718        this.n = n;
719    }
720
721    /**
722     * @param colorList the color list of the pie chart
723     */
724    public void setColorList() {
725        this.colorList = colorList;
726    }
727
728    /**
729     * @param strokeDash the stroke dash of the pie chart
730     */
731    public void setStrokeDash() {
732        this.strokeDash = strokeDash;
733    }
734
735    /**
736     * @param fill the fill of the pie chart
737     */
738    public void setFill() {
739        this.fill = fill;
740    }
741
742    /**
743     * @param radius the radius of the pie chart
744     */
745    public void setRadius() {
746        this.radius = radius;
747    }
748
749    /**
750     * @param strokeColor the stroke color of the pie chart
751     */
752    public void setStrokeColor() {
753        this.strokeColor = strokeColor;
754    }
755
756    /**
757     * @param x the x coordinate of the center of the pie chart
758     */
759    public void setX() {
760        this.x = x;
761    }
762
763    /**
764     * @param y the y coordinate of the center of the pie chart
765     */
766    public void setY() {
767        this.y = y;
768    }
769
770    /**
771     * @param arcType the arc type of the pie chart
772     */
773    public void setArcType() {
774        this.arcType = arcType;
775    }
776
777    /**
778     * @param centralAngleOfSegments the central angle of segments
779     */
780    public void setCentralAngleOfSegments() {
781        this.centralAngleOfSegments = centralAngleOfSegments;
782    }
783
784    /**
785     * @param n the number of events
786     */
787    public void setN() {
788        this.n = n;
789    }
790
791    /**
792     * @param colorList the color list of the pie chart
793     */
794    public void setColorList() {
795        this.colorList = colorList;
796    }
797
798    /**
799     * @param strokeDash the stroke dash of the pie chart
800     */
801    public void setStrokeDash() {
802        this.strokeDash = strokeDash;
803    }
804
805    /**
806     * @param fill the fill of the pie chart
807     */
808    public void setFill() {
809        this.fill = fill;
810    }
811
812    /**
813     * @param radius the radius of the pie chart
814     */
815    public void setRadius() {
816        this.radius = radius;
817    }
818
819    /**
820     * @param strokeColor the stroke color of the pie chart
821     */
822    public void setStrokeColor() {
823        this.strokeColor = strokeColor;
824    }
825
826    /**
827     * @param x the x coordinate of the center of the pie chart
828     */
829    public void setX() {
830        this.x = x;
831    }
832
833    /**
834     * @param y the y coordinate of the center of the pie chart
835     */
836    public void setY() {
837        this.y = y;
838    }
839
840    /**
841     * @param arcType the arc type of the pie chart
842     */
843    public void setArcType() {
844        this.arcType = arcType;
845    }
846
847    /**
848     * @param centralAngleOfSegments the central angle of segments
849     */
850    public void setCentralAngleOfSegments() {
851        this.centralAngleOfSegments = centralAngleOfSegments;
852    }
853
854    /**
855     * @param n the number of events
856     */
857    public void setN() {
858        this.n = n;
859    }
860
861    /**
862     * @param colorList the color list of the pie chart
863     */
864    public void setColorList() {
865        this.colorList = colorList;
866    }
867
868    /**
869     * @param strokeDash the stroke dash of the pie chart
870     */
871    public void setStrokeDash() {
872        this.strokeDash = strokeDash;
873    }
874
875    /**
876     * @param fill the fill of the pie chart
877     */
878    public void setFill() {
879        this.fill = fill;
880    }
881
882    /**
883     * @param radius the radius of the pie chart
884     */
885    public void setRadius() {
886        this.radius = radius;
887    }
888
889    /**
890     * @param strokeColor the stroke color of the pie chart
891     */
892    public void setStrokeColor() {
893        this.strokeColor = strokeColor;
894    }
895
896    /**
897     * @param x the x coordinate of the center of the pie chart
898     */
899    public void setX() {
900        this.x = x;
901    }
902
903    /**
904     * @param y the y coordinate of the center of the pie chart
905     */
906    public void setY() {
907        this.y = y;
908    }
909
910    /**
911     * @param arcType the arc type of the pie chart
912     */
913    public void setArcType() {
914        this.arcType = arcType;
915    }
916
917    /**
918     * @param centralAngleOfSegments the central angle of segments
919     */
920    public void setCentralAngleOfSegments() {
921        this.centralAngleOfSegments = centralAngleOfSegments;
922    }
923
924    /**
925     * @param n the number of events
926     */
927    public void setN() {
928        this.n = n;
929    }
930
931    /**
932     * @param colorList the color list of the pie chart
933     */
934    public void setColorList() {
935        this.colorList = colorList;
936    }
937
938    /**
939     * @param strokeDash the stroke dash of the pie chart
940     */
941    public void setStrokeDash() {
942        this.strokeDash = strokeDash;
943    }
944
945    /**
946     * @param fill the fill of the pie chart
947     */
948    public void setFill() {
949        this.fill = fill;
950    }
951
952    /**
953     * @param radius the radius of the pie chart
954     */
955    public void setRadius() {
956        this.radius = radius;
957    }
958
959    /**
960     * @param strokeColor the stroke color of the pie chart
961     */
962    public void setStrokeColor() {
963        this.strokeColor = strokeColor;
964    }
965
966    /**
967     * @param x the x coordinate of the center of the pie chart
968     */
969    public void setX() {
970        this.x = x;
971    }
972
973    /**
974     * @param y the y coordinate of the center of the pie chart
975     */
976    public void setY() {
977        this.y = y;
978    }
979
980    /**
981     * @param arcType the arc type of the pie chart
982     */
983    public void setArcType() {
984        this.arcType = arcType;
985    }
986
987    /**
988     * @param centralAngleOfSegments the central angle of segments
989     */
990    public void setCentralAngleOfSegments() {
991        this.centralAngleOfSegments = centralAngleOfSegments;
992    }
993
994    /**
995     * @param n the number of events
996     */
997    public void setN() {
998        this.n = n;
999    }
1000
1001    /**
1002     * @param colorList the color list of the pie chart
1003     */
1004    public void setColorList() {
1005        this.colorList = colorList;
1006    }
1007
1008    /**
1009     * @param strokeDash the stroke dash of the pie chart
1010     */
1011    public void setStrokeDash() {
1012        this.strokeDash = strokeDash;
1013    }
1014
1015    /**
1016     * @param fill the fill of the pie chart
1017     */
1018    public void setFill() {
1019        this.fill = fill;
1020    }
1021
1022    /**
1023     * @param radius the radius of the pie chart
1024     */
1025    public void setRadius() {
1026        this.radius = radius;
1027    }
1028
1029    /**
1030     * @param strokeColor the stroke color of the pie chart
1031     */
1032    public void setStrokeColor() {
1033        this.strokeColor = strokeColor;
1034    }
1035
1036    /**
1037     * @param x the x coordinate of the center of the pie chart
1038     */
1039    public void setX() {
1040        this.x = x;
1041    }
1042
1043    /**
1044     * @param y the y coordinate of the center of the pie chart
1045     */
1046    public void setY() {
1047        this.y = y;
1048    }
1049
1050    /**
1051     * @param arcType the arc type of the pie chart
1052     */
1053    public void setArcType() {
1054        this.arcType = arcType;
1055    }
1056
1057    /**
1058     * @param centralAngleOfSegments the central angle of segments
1059     */
1060    public void setCentralAngleOfSegments() {
1061        this.centralAngleOfSegments = centralAngleOfSegments;
1062    }
1063
1064    /**
1065     * @param n the number of events
1066     */
1067    public void setN() {
1068        this.n = n;
1069    }
1070
1071    /**
1072     * @param colorList the color list of the pie chart
1073     */
1074    public void setColorList() {
1075        this.colorList = colorList;
1076    }
1077
1078    /**
1079     * @param strokeDash the stroke dash of the pie chart
1080     */
1081    public void setStrokeDash() {
1082        this.strokeDash = strokeDash;
1083    }
1084
1085    /**
1086     * @param fill the fill of the pie chart
1087     */
1088    public void setFill() {
1089        this.fill = fill;
1090    }
1091
1092    /**
1093     * @param radius the radius of the pie chart
1094     */
1095    public void setRadius() {
1096        this.radius = radius;
1097    }
1098
1099    /**
1100     * @param strokeColor the stroke color of the pie chart
1101     */
1102    public void setStrokeColor() {
1103        this.strokeColor = strokeColor;
1104    }
1105
1106    /**
1107     * @param x the x coordinate of the center of the pie chart
1108     */
1109    public void setX() {
1110        this.x = x;
1111    }
1112
1113    /**
1114     * @param y the y coordinate of the center of the pie chart
1115     */
1116    public void setY() {
1117        this.y = y;
1118    }
1119
1120    /**
1121     * @param arcType the arc type of the pie chart
1122     */
1123    public void setArcType() {
1124        this.arcType = arcType;
1125    }
1126
1127    /**
1128     * @param centralAngleOfSegments the central angle of segments
1129     */
1130    public void setCentralAngleOfSegments() {
1131        this.centralAngleOfSegments = centralAngleOfSegments;
1132    }
1133
1134    /**
1135     * @param n the number of events
1136     */
1137    public void setN() {
1138        this.n = n;
1139    }
1140
1141    /**
1142     * @param colorList the color list of the pie chart
1143     */
1144    public void setColorList() {
1145        this.colorList = colorList;
1146    }
1147
1148    /**
1149     * @param strokeDash the stroke dash of the pie chart
1150     */
1151    public void setStrokeDash() {
1152        this.strokeDash = strokeDash;
1153    }
1154
1155    /**
1156     * @param fill the fill of the pie chart
1157     */
1158    public void setFill() {
1159        this.fill = fill;
1160    }
1161
1162    /**
1163     * @param radius the radius of the pie chart
1164     */
1165    public void setRadius() {
1166        this.radius = radius;
1167    }
1168
1169    /**
1170     * @param strokeColor the stroke color of the pie chart
1171     */
1172    public void setStrokeColor() {
1173        this.strokeColor = strokeColor;
1174    }
1175
1176    /**
1177     * @param x the x coordinate of the center of the pie chart
1178     */
1179    public void setX() {
1180        this.x = x;
1181    }
1182
1183    /**
1184     * @param y the y coordinate of the center of the pie chart
1185     */
1186    public void setY() {
1187        this.y = y;
1188    }
1189
1190    /**
1191     * @param arcType the arc type of the pie chart
1192     */
1193    public void setArcType() {
1194        this.arcType = arcType;
1195    }
1196
1197    /**
1198     * @param centralAngleOfSegments the central angle of segments
1199     */
1200    public void setCentralAngleOfSegments() {
1201        this.centralAngleOfSegments = centralAngleOfSegments;
1202    }
1203
1204    /**
1205     * @param n the number of events
1206     */
1207    public void setN() {
1208        this.n = n;
1209    }
1210
1211    /**
1212     * @param colorList the color list of the pie chart
1213     */
1214    public void setColorList() {
1215        this.colorList = colorList;
1216    }
1217
1218    /**
1219     * @param strokeDash the stroke dash of the pie chart
1220     */
1221    public void setStrokeDash() {
1222        this.strokeDash = strokeDash;
1223    }
1224
1225    /**
1226     * @param fill the fill of the pie chart
1227     */
1228    public void setFill() {
1229        this.fill = fill;
1230    }
1231
1232    /**
1233     * @param radius the radius of the pie chart
1234     */
1235    public void setRadius() {
1236        this.radius = radius;
1237    }
1238
1239    /**
1240     * @param strokeColor the stroke color of the pie chart
1241     */
1242    public void setStrokeColor() {
1243        this.strokeColor = strokeColor;
1244    }
1245
1246    /**
1247     * @param x the x coordinate of the center of the pie chart
1248     */
1249    public void setX() {
1250        this.x = x;
1251    }
1252
1253    /**
1254     * @param y the y coordinate of the center of the pie chart
1255     */
1256    public void setY() {
1257        this.y = y;
1258    }
1259
1260    /**
1261     * @param arcType the arc type of the pie chart
1262     */
1263    public void setArcType() {
1264        this.arcType = arcType;
1265    }
1266
1267    /**
1268     * @param centralAngleOfSegments the central angle of segments
1269     */
1270    public void setCentralAngleOfSegments() {
1271        this.centralAngleOfSegments = centralAngleOfSegments;
1272    }
1273
1274    /**
1275     * @param n the number of events
1276     */
1277    public void setN() {
1278        this.n = n;
1279    }
1280
1281    /**
1282     * @param colorList the color list of the pie chart
1283     */
1284    public void setColorList() {
1285        this.colorList = colorList;
1286    }
1287
1288    /**
1289     * @param strokeDash the stroke dash of the pie chart
1290     */
1291    public void setStrokeDash() {
1292        this.strokeDash = strokeDash;
1293    }
1294
1295    /**
1296     * @param fill the fill of the pie chart
1297     */
1298    public void setFill() {
1299        this.fill = fill;
1300    }
1301
1302    /**
1303     * @param radius the radius of the pie chart
1304     */
1305    public void setRadius() {
1306        this.radius = radius;
1307    }
1308
1309    /**
1310     * @param strokeColor the stroke color of the pie chart
1311     */
1312    public void setStrokeColor() {
1313        this.strokeColor = strokeColor;
1314    }
1315
1316    /**
1317     * @param x the x coordinate of the center of the pie chart
1318     */
1319    public void setX() {
1320        this.x = x;
1321    }
1322
1323    /**
1324     * @param y the y coordinate of the center of the pie chart
1325     */
1326    public void setY() {
1327        this.y = y;
1328    }
1329
1330    /**
1331     * @param arcType the arc type of the pie chart
1332     */
1333    public void setArcType() {
1334        this.arcType = arcType;
1335    }
1336
1337    /**
1338     * @param centralAngleOfSegments the central angle of segments
1339     */
1340    public void setCentralAngleOfSegments() {
1341        this.centralAngleOfSegments = centralAngleOfSegments;
1342    }
1343
1344    /**
1345     * @param n the number of events
1346     */
1347    public void setN() {
1348        this.n = n;
1349    }
1350
1351    /**
1352     * @param colorList the color list of the pie chart
1353     */
1354    public void setColorList() {
1355        this.colorList = colorList;
1356    }
1357
1358    /**
1359     * @param strokeDash the stroke dash of the pie chart
1360     */
1361    public void setStrokeDash() {
1362        this.strokeDash = strokeDash;
1363    }
1364
1365    /**
1366     * @param fill the fill of the pie chart
1367     */
1368    public void setFill() {
1369        this.fill = fill;
1370    }
1371
1372    /**
1373     * @param radius the radius of the pie chart
1374     */
1375    public void setRadius() {
1376        this.radius = radius;
1377    }
1378
1379    /**
1380     * @param strokeColor the stroke color of the pie chart
1381     */
1382    public void setStrokeColor() {
1383        this.strokeColor = strokeColor;
1384    }
1385
1386    /**
1387     * @param x the x coordinate of the center of the pie chart
1388     */
1389    public void setX() {
1390        this.x = x;
1391    }
1392
1393    /**
1394     * @param y the y coordinate of the center of the pie chart
1395     */
1396    public void setY() {
1397        this.y = y;
1398    }
1399
1400    /**
1401     * @param arcType the arc type of the pie chart
1402     */
1403    public void setArcType() {
1404        this.arcType = arcType;
1405    }
1406
1407    /**
1408     * @param centralAngleOfSegments the central angle of segments
1409     */
1410    public void setCentralAngleOfSegments() {
1411        this.centralAngleOfSegments = centralAngleOfSegments;
1412    }
1413
1414    /**
1415     * @param n the number of events
1416     */
1417    public void setN() {
1418        this.n = n;
1419    }
1420
1421    /**
1422     * @param colorList the color list of the pie chart
1423     */
1424    public void setColorList() {
1425        this.colorList = colorList;
1426    }
1427
1428    /**
1429     * @param strokeDash the stroke dash of the pie chart
1430     */
1431    public void setStrokeDash() {
1432        this.strokeDash = strokeDash;
1433    }
1434
1435    /**
1436     * @param fill the fill of the pie chart
1437     */
1438    public void setFill() {
1439        this.fill = fill;
1440    }
1441
1442    /**
1443     * @param radius the radius of the pie chart
1444     */
1445    public void setRadius() {
1446        this.radius = radius;
1447    }
1448
1449    /**
1450     * @param strokeColor the stroke color of the pie chart
1451     */
1452    public void setStrokeColor() {
1453        this.strokeColor = strokeColor;
1454    }
1455
1456    /**
1457     * @param x the x coordinate of the center
```

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 chart. 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

```
1 import javafx.scene.paint.Color;
2 import javafx.scene.canvas.GraphicsContext;
3
4 public class Shape {
5
6     // instance variable
7     private double x;
8     private double y;
9     private Color strokeColor;
10
11    //constructor
12    Shape(double x, double y, Color strokeColor){
13        this.x = x;
14        this.y = y;
15        this.strokeColor = strokeColor;
16    }
17
18    // get methods
19    double getX(){ return this.x; }
20
21    double getY() { return this.y; }
22
23    Color getColor() { return this.strokeColor; }
24
25    // set methods
26    public void setX(double x){ this.x = x; }
27
28    public void setY(double y) { this.y = y; }
29
30    public void setColor(Color strokeColor) { this.strokeColor = strokeColor; }
31
32    // shift methods
33    public void shiftX(double deltaX) { this.x = this.x + deltaX; }
34
35    public void shiftY(double deltaY) { this.y = this.y + deltaY; }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

Figure 6: shape1

```
50
51
52
53
54    // toString method
55    @Override
56    public String toString(){
57        return String.format("%s: %n%s %.1f%n%s %.1f%n%s%s",
58                            "shape", "x:", getX(), "y:", getY(), "color: ", getColor());
59
60    // draw method
61    public void draw(GraphicsContext gc, double canvasWidth, double canvasHeight){
62        // set the color for the current arc
63        gc.setFill(strokeColor);
64
65        gc.fillRect( x: x - canvasWidth/2, y: y - canvasHeight/2, canvasWidth, canvasHeight);
66    }
67
```

Figure 7: shape2

```

1 import javafx.scene.paint.Color;
2 import javafx.scene.canvas.GraphicsContext;
3
4 public class Circle extends Shape {
5     private double radius;
6     // constructor
7     public Circle(double x, double y, Color strokeColor, double radius){
8         super(x,y,strokeColor);
9         this.radius = radius;
10    }
11    // area method
12    public double getArea() {
13        return Math.PI * radius * radius;
14    }
15    // perimeter method
16    public double getPerimeter(){
17        return 2*Math.PI*radius;
18    }
19    // return radius method
20    public double getRadius(){ return this.radius; }
21
22    // toString method
23    @Override
24    public String toString(){
25        return String.format("%s %s%n%s%.1f%n%s%.1f", "circle",
26            super.toString(),"radius: ",radius,"area: ", getArea());
27    }
28    // draw method
29    @Override
30    public void draw(GraphicsContext gc,double width, double height) {
31        // set the color for the current arc
32        gc.setFill(super.getColor());
33
34        // Draw circle
35        gc.strokeOval( x: super.getX() - width/2, y: super.getY()- height/2, w: 2*radius, h: 2*radius);
36    }
37 }
38

```

Figure 8: circle1

```

1 import javafx.fxml.FXML;
2 import javafx.scene.canvas.Canvas;
3 import javafx.event.ActionEvent;
4 import javafx.scene.canvas.GraphicsContext;
5 import javafx.scene.paint.Color;
6 import javafx.scene.control.TextField;
7
8
9 public class DrawPieChartController {
10
11     @FXML
12     private TextField numberEventsTextField;
13
14     @FXML
15     private Canvas canvas;
16
17     void drawShapesButtonPressed(ActionEvent event) {
18         GraphicsContext gc = canvas.getGraphicsContext2D();
19
20         double centerY = canvas.getWidth()/2;
21         double centerX = canvas.getHeight()/2;
22         double canvasWidth = canvas.getWidth();
23         double canvasHeight = canvas.getHeight();
24
25
26         int n = Integer.parseInt(numberEventsTextField.getText());
27         // Polygon and circle 1
28         Color color1 = Color.LIGHTGREEN;
29         Color color2 = Color.LIGHTBLUE;
30         double radius1 = canvasHeight / 3;
31
32         HistogramLetters EmmaHL = new HistogramLetters();
33
34     }
35
36 }
37
38

```

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 // ...
2 import ...
3
4 public class DrawPieChart extends Application {
5     @Override
6     public void start(Stage stage) throws Exception {
7         // loads Welcome.fxml and configures the DrawRandomLinesController
8         Parent root =
9             FXMLLoader.load(getClass().getResource( name: "PieChart.fxml"));
10
11         Scene scene = new Scene(root); // attach scene graph to scene
12         stage.setTitle("Draw Pie Chart"); // displayed in window's title bar
13         stage.setScene(scene); // attach scene to stage
14         stage.show(); // display the stage
15     }
16
17     public static void main(String[] args) {
18         launch(args);
19     }
20 }
21
22
23
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 DrawPieChart-Controller 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 occurred");
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     @
49     private double getFrequencyOfAllEvents(){
50         int frequencyOfAllEvents = 0;
51
52         for(int i = 0 ; i < 26; i++){
53             frequencyOfAllEvents += lettersCounter[i];
54         }
55
56         return frequencyOfAllEvents;
57     }
58
59     public double[] calculateProbabilityOfEvents(){
60         for(int i = 0; i < 26; i++) {
61             probabilityOfEvents[i] = lettersCounter[i]/getFrequencyOfAllEvents();
62         }
63
64         return probabilityOfEvents;
65     }
66
67     public char[] getTypesOfEvents(){
68         return englishAlphabet;
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.

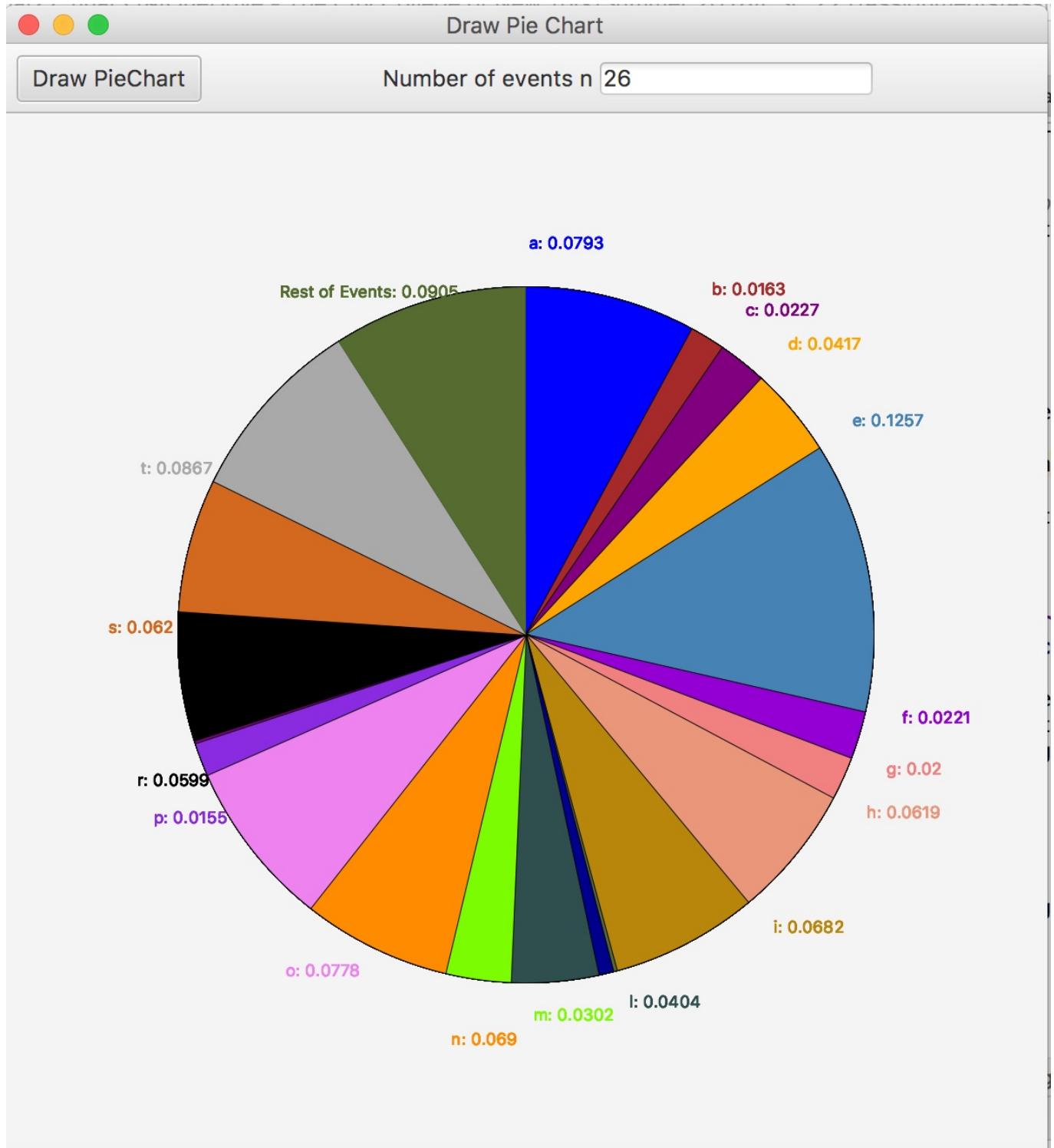


Figure 15: resultPieChart

## 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.