Zi Xuan Li

Professor Auda

CSC 22100

April 30th, 2023

Assignment #3

Statement of the problem:

The purpose of this assignment is to build off the previous assignment and create a class that calculates the frequencies of probabilities of alphabet characters in a text using map collections that will be used to create a pie chart to show the frequencies and probabilities of the n most frequent occurrences in the text.

Solution methods:

1. Implement a Java class **HistogramAlphabet** that calculates the frequencies and probabilities of the alphabet characters in *Leo Tolstoy's "Moby Dick*" (file MobyDick.txt). The **HistogramAlphabet** class utilizes **Map** collections, Map<**Character, Integer**> and Map<**Character, Doubl**e> and stream operations, for a statistical calculation and sorting of the frequencies and probabilities. It also includes a **MyPieChart** class as a *non-static* inner class.

2. Class **MyPieChart** displays a pie chart of the probabilities of the **n** most frequent occurrences of an event – the frequency of characters in a document. The probability of the event is given by: Probability of event = frequency of event/summation of frequencies of all events.

i. Each event is represented by a slice of the pie chart. The size of the slice is proportional to the probability of the corresponding event: Probability of event = Central angle of slice/2pi.

ii. Each slice has a different color for your choice of type enum **MyColor**.

iii. Each slice has a legend showing the corresponding events and its probability.
iv. The slices are displayed in order of decreasing probability.
v. The last slice represents "All Other Events" and their cumulative probability. As an example, in the graph below where the event is the occurrence of a letter in a text: **n** = 5 and the probability of All Other Events is one minus the sum of the probabilities of events e, t, a, o, and n.

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

4. Class **Slice** includes appropriate constructors and methods, including methods that perform the following operations:

a. toString – returns a string representation of a Slice object.

b. draw – draws a Slice object.

5. Amend the JavaFx **BorderPane** Layout in Assignment 2, as follows:

BorderPane Layout:

BorderPane Layout is the root of the scene graph in the JavaFx Application. It is comprised of three regions – Top, Left, and Center:

a. Top Region: The region encompasses a **HBox** Layout, encompassing an additional geometric image of a pie chart which, upon selection, provides a dialogue box for entering the number of characters to be displayed and starting angle, and draws the pie chart on the **Canvas** in the Center region.

b. Left region: The region encompasses a **VBox** Layout, encompassing a **Label** object displaying the text "MyColor Palette" and a **MyColorPalatte** object showing a set of constant colors of enum **MyColor** type for color selection.

c. Center region: The region encompasses a **Canvas** object used for drawing **MyShape** objects and their intersection.

d. Right region: The region is utilized to provide a legend for the pie chart showing the character frequencies and associated pie chart colors.

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

a. The code is applicable to canvases of variable height and width.

b. The size of the pie char tis proportional to the smallest dimension of the canvas.

7. <u>Explicitly specify all the classes imported and used in your Java code.</u>

All classes that imported:

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.text.Font;
import javafx.stage.Stage;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Paths;
import java.util.*;
import java.util.List;
import java.util.stream.Collectors;
import javafx.scene.shape.ArcType;
import java.lang.IllegalStateException;
```

Java code: (Classes made previously were not edited with the exception of MyShapeApplication which is included below.)

Slice.java

```
package com.example.assignment1;

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

public class Slice {

    MyPoint center;
    double width, height;
    double startAngle;
    double arcAngle;
    MyColor color;
    String information;

    Slice(MyPoint center, double width, double height, double startAngle,
double arcAngle, MyColor color, String information) {
        this.center = center;
        this.width = width; this.height = height;
        this.startAngle = startAngle; this.arcAngle = arcAngle;
        this.color = color;
        this.information = information;
    }

    Slice (Slice s) {
        this.center = s.getCenter();
        this.width = s.getWidth(); this.height = s.getHeight();
        this.startAngle = s.getStartAngle();
        this.arcAngle = s.getArcAngle();
        this.information = s.getInformation();
    }

    public void setColor(MyColor color) {
        this.color = color;
    }

    public MyPoint getCenter() {
        return center;
    }

    public double getWidth() {
        return width;
    }

    public double getHeight() {
        return height;
    }

    public double getStartAngle() {
        return startAngle;
    }
```

```
    public double getArcAngle() {
        return arcAngle;
    }

    public String getInformation() {
        return information;
    }

    public void draw(GraphicsContext GC){
        GC.setFill(color.getJavaFXColor());
        GC.fillArc(center.getX() - 0.5 * width, center.getY() - 0.5 * height,
width, height, startAngle, arcAngle, ArcType.ROUND);

        GC.setStroke((MyColor.WHITE.getJavaFXColor()));
        GC.strokeArc(center.getX() - 0.5 * width, center.getY() - 0.5 *
height, width, height, startAngle, arcAngle, ArcType.ROUND);

        double x = center.getX(); double y = center.getY();
        double a = 0.55 * width; double b = 0.55 * height; double midAngle =
startAngle + 0.5 * arcAngle;
        midAngle = midAngle < 360 ? midAngle : midAngle - 360;

        double u = a * Math.abs(Math.sin(Math.toRadians(midAngle)));
        double v = b * Math.abs(Math.cos(Math.toRadians(midAngle)));
        double w = 1.0 / Math.sqrt(u * u + v * v);

        u = (midAngle >= 0.0 && midAngle <= 180.0) ? -u : u;
        v = (midAngle >= 90.0 && midAngle <= 270.0) ? -v : v;
        double xText = x + a * v * w;
        double yText = y + b * u * w;

        xText = (midAngle >= 90.0 && midAngle <= 270.0) ? xText - 0.125 *
width : xText;

        GC.setStroke(MyColor.WHITE.getJavaFXColor());
        GC.setFont(Font.font("Calibri",13));
        GC.strokeText(information, xText, yText);
    }

    @Override
    public String toString(){
        return "Slice: Center (" + center.getX() + ", " + center.getY() + ")"
                + " Width " + width + " Height " + height
                + " (Starting Angle, Angle): (" + startAngle + ", " +
arcAngle + "),"
                + " Color " + color.getJavaFXColor();
    }
}
```

HistogramAlphabet.java

```
package com.example.assignment1;

import javafx.scene.canvas.GraphicsContext;
import java.util.*;
import java.util.stream.Collectors;
```

```java
public class HistogramAlphabet {

    Map <Character, Integer> frequency = new HashMap<Character, Integer>();

    Map <Character, Double> probability = new HashMap<Character, Double>();

    HistogramAlphabet(){}

    HistogramAlphabet(Map<Character, Integer> m) {
        frequency.putAll(m);
    }

    HistogramAlphabet(HistogramAlphabet h) {
        frequency.putAll(h.getFrequency());
probability.putAll(h.getProbability());
    }

    HistogramAlphabet(String text){
        String w = text.replaceAll("[^a-zA-Z]", "").toLowerCase();

        for (int i = 0; i < w.length(); i++){
            Character key = w.charAt(i);
            incrementFrequency(frequency, key);
        }
        probability = getProbability();
    }

    /*
    HistogramAlphabet(ResultSet RS, String fieldKey, String fieldValue){
        try{
            while (RS.next()){
                frequency.put(RS.getString(fieldKey).charAt(8),
RS.getInt(fieldValue));
            }
        }
        catch(NoSuchElementException | IllegalStateException | SQLException
e) {
            System.out.println(e);
        }
     */

    public Map<Character, Integer> getFrequency() {
        return frequency;
    }

    public Integer getCumulativeFrequency() {
        return frequency.values().stream().reduce(0, Integer::sum);
    }

    public Map <Character, Integer> sortUpFrequency() {
        return frequency
                .entrySet()
                .stream()
                .sorted(Map.Entry.comparingByValue())
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
```

```
    }

    public Map <Character, Integer> sortDownFrequency() {
        return frequency
                .entrySet()
                .stream()

.sorted(Collections.reverseOrder(Map.Entry.comparingByValue()))
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Map <Character, Double> getProbability() {
        double inverseCumulativeFrequency = 1.0 / getCumulativeFrequency();
        for (Character Key : frequency.keySet()){
            probability.put(Key, (double) frequency.get(Key) *
inverseCumulativeFrequency);
        }
        return probability;
    }

    public Map <Character, Double> sortDownProbability() {
        return getProbability().entrySet()
                .stream()

.sorted(Collections.reverseOrder(Map.Entry.comparingByValue()))
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Map <Character, Double> sortUpProbability() {
        return getProbability().entrySet()
                .stream()
                .sorted(Map.Entry.comparingByValue())
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Double getSumOfProbability() {
        return probability.values().stream().reduce(0.0, Double::sum);
    }

    public boolean checkSumOfProbability() {
        return getSumOfProbability() == 1;
    }

    @Override
    public String toString() {
        String output = "Frequency of Characters:\n";
        for (Character K : frequency.keySet()){
            output += K + ": " + frequency.get(K) + "\n";
        }
        return output;
    }

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

```
        int N;
        int M;
        MyPoint center;
        double width, height;
        double rotateAngle;

        MyPieChart(int N, int M, MyPoint center, double width, double height,
double rotateAngle) {
            this.N = N; this.M = M;
            this.center = center;
            this.width = width; this.height = height;
            this.rotateAngle = rotateAngle < 360 ? rotateAngle : rotateAngle
- 360;

            slices = getMyPieChart();
        }

        public Map <Character, Slice> getMyPieChart(){
            MyColor [] colors = MyColor.getMyColors();
            int colorsSize = colors.length;

            Map <Character, Double> sortedProbability =
sortDownProbability();
            Random rand = new Random();
            double sliceStartAngle = this.rotateAngle;
            for (Character key : sortedProbability.keySet()){
                double sliceValue = sortedProbability.get(key);

                double sliceArcAngle = 360.0 * sliceValue;
                MyColor color = colors[rand.nextInt(colorsSize)];
                String sliceInformation = key + ": " + String.format("%.4f",
sliceValue);
                slices.put(key, new Slice(center, width, height,
sliceStartAngle, sliceArcAngle, color, sliceInformation));

                sliceStartAngle += sliceArcAngle;
                sliceStartAngle = sliceStartAngle < 360.0 ? sliceStartAngle :
sliceStartAngle - 360.0;
            }
            return slices;
        }

        public void draw(GraphicsContext GC) {
            Map <Character, Double> sortedProbability =
sortDownProbability();

            GC.clearRect(0.0, 0.0, GC.getCanvas().getWidth(),
GC.getCanvas().getHeight());
            GC.setFill(MyColor.GRAY.getJavaFXColor());
            GC.fillRect(0.0, 0.0, GC.getCanvas().getWidth(),
GC.getCanvas().getHeight());

            int n = 0;
            double probabilityAllOtherCharacters = 1.0;
            for (Character key : sortedProbability.keySet()){
                double sliceStartAngle = slices.get(key).getStartAngle();
```

```
                double sliceArcAngle = slices.get(key).getArcAngle();
                if (n < N){
                    slices.get(key).draw(GC);
                    probabilityAllOtherCharacters -=
sortedProbability.get(key);
                    n++;
                }
                else {
                    if (N != M) {
                        String information = "All other characters: "+
String.format("%.4f", probabilityAllOtherCharacters);
                        if (sliceStartAngle < rotateAngle){
                            Slice sliceAllOtherCharacters = new Slice(center,
width, height, sliceStartAngle, rotateAngle - sliceStartAngle,
MyColor.getRandomColor(), information);
                            sliceAllOtherCharacters.draw(GC);
                        }
                        else {
                            Slice sliceAllOtherCharacters = new Slice(center,
width, height, sliceStartAngle, 360.0 - sliceStartAngle + rotateAngle,
MyColor.getRandomColor(), information);
                            sliceAllOtherCharacters.draw(GC);
                        }
                        break;
                    }
                }
            }
        }
    }

    private static <K> void incrementFrequency(Map<K, Integer> m, K Key) {
        m.putIfAbsent(Key, 0);
        m.put(Key, m.get(Key) + 1);
    }
}
```

FrequencyOfCharacters.java

```
package com.example.assignment1;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.TilePane;
import javafx.scene.text.Font;
import javafx.stage.Stage;

import java.io.FileNotFoundException;
import java.nio.file.Paths;
```

```
import java.lang.IllegalStateException;
import java.util.*;
import java.io.IOException;
public class FrequencyOfCharacters extends Application {
    Integer N;
    Integer M;
    double startAngle;
    double scale;
    String Title;
    String filename;
    Scanner input;

    Boolean isPiechart;
    List<String> barchartInputs = new ArrayList();
    List<String> piechartInputs = new ArrayList();

    public void toggleGroup(){
        ToggleGroup group = new ToggleGroup();

        RadioButton radioPiechart = new RadioButton("Pie Chart");
        radioPiechart.setToggleGroup(group);

        Dialog<Boolean> dialog = new Dialog<>();
        dialog.setTitle("Chart Picker");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,200,20,10));

        gridDialog.add(radioPiechart, 0, 0);

        Platform.runLater(() -> radioPiechart.setSelected(true));

        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                this.isPiechart = radioPiechart.isSelected();
                if (this.isPiechart) { dialogPiechart(); }
                }
            return null;
        });

        Optional <Boolean> Result = dialog.showAndWait();
    }

    public void dialogPiechart() {
        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("Pie Chart");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);
```

```
        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets( 20, 150, 10, 10));

        TextField numberEvents = new TextField();
        TextField totalNumberEvents = new TextField();
        TextField startingAngle = new TextField();

        ComboBox title = new ComboBox();
        title.getItems().addAll( "Alice in Wonderland",
                "A Tale of Two Cities", "David Copperfield", "Oliver Twist",
                "Emma", "Pride and Prejudice", "Moby Dick", "War and Peace",
                "xWords");

        gridDialog.add(new Label("Display"), 0, 0);
        gridDialog.add(numberEvents, 1, 0);
        gridDialog.add(new Label("Total"), 2, 0);
        gridDialog.add(totalNumberEvents, 3,0);
        gridDialog.add(new Label("Starting Angle"), 0, 1);
        gridDialog.add(startingAngle, 1, 1);
        gridDialog.add(new Label("Title"), 0, 2);
        gridDialog.add(title, 1, 2);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> numberEvents.requestFocus());

        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                piechartInputs.add(numberEvents.getText());
piechartInputs.add(totalNumberEvents.getText());
piechartInputs.add(startingAngle.getText());
piechartInputs.add(title.getValue().toString());
                return piechartInputs;
            }
            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Result.ifPresent(event -> {
            this.N = Integer.parseInt(piechartInputs.get(0));
            this.M = Integer.parseInt(piechartInputs.get(1));
            this.startAngle = Double.parseDouble(piechartInputs.get(2));
            this.Title = piechartInputs.get(3);
            this.filename =
"C:\\Users\\zixua\\IdeaProjects\\Assignment3\\Texts\\" + Title + ".txt";
        });
    }

    public void openFile() {
        try {
            input = new Scanner(Paths.get(filename));
        } catch (IOException ioException){
            System.err.println("File is not found");
        }
```

```
    }

    public String readFile() {
        String w = "";
        try {
            while (input.hasNext()) {
                w += input.nextLine().replaceAll("[^a-zA-
Z]","").toLowerCase();
            }
        }
        catch (NoSuchElementException elementException) {
            System.err.println("Invalid input! Terminating...");
        }
        catch (IllegalStateException stateException) {
            System.out.println("Error processing file! Terminating...");
        }
        return w;
    }

    public void closeFile() {
        if (input != null) input.close();
    }

    public Canvas addCanvasLegend(double widthCanvas, double heightCanvas,
HistogramAlphabet H) {
        String information;

        Canvas CV = new Canvas(widthCanvas, heightCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();

        MyColor colorLeftCanvas = MyColor.LINEN;
        GC.setFill(colorLeftCanvas.getJavaFXColor());
        GC.fillRect(0.0, 0.0, widthCanvas, heightCanvas);

        double xText = 20; double yText = 0.03625 * heightCanvas;
        MyColor colorStroke = MyColor.GRAY;
        GC.setStroke(colorStroke.invertColor());
        GC.setFont(Font.font("Calibri", 13));
        GC.strokeText("Frequency: Cumulative " + H.getCumulativeFrequency(),
xText, yText);

        Map <Character, Integer> sortedFrequency = H.sortDownFrequency();

        Double yStep = 0.03625 * heightCanvas;
        for (Character K: sortedFrequency.keySet()){
            yText += yStep;
            information = K + ":\t" + sortedFrequency.get(K);
            GC.strokeText(information, xText, yText);
        }
        return CV;
    }

    public Canvas addCanvasPieChart(double widthCanvas, double heightCanvas,
HistogramAlphabet H){
        Canvas CV = new Canvas(widthCanvas, heightCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
```

```
        MyPoint center = new MyPoint(0.4 * widthCanvas, 0.5 * heightCanvas,
null);

        double diameterPieChart = 0.60 * Math.min(widthCanvas, heightCanvas);
        HistogramAlphabet.MyPieChart pieChart = H.new MyPieChart(N, M,
center, diameterPieChart, diameterPieChart, startAngle);
        Map <Character, Slice> slices = pieChart.getMyPieChart();

        System.out.println("\nPie Chart");
        slices.forEach(((K,V) -> System.out.println(K + ": " +
slices.get(K))));

        double sumOfAngles = 0.0;
        for (Character key : slices.keySet()){
            sumOfAngles += slices.get(key).getArcAngle();
        }
        System.out.println("\nSum Of Angles: " + sumOfAngles);

        pieChart.draw(GC);

        return CV;
    }

    @Override
    public void start(Stage PS){
        toggleGroup();

        openFile();
        String w = readFile();
        closeFile();

        HistogramAlphabet H = new HistogramAlphabet(w);
        Map <Character, Integer> sortedFrequency = H.sortDownFrequency();

        System.out.println("\nFrequency of Characters");
        sortedFrequency.forEach((K, V) -> System.out.println(K + ": " + V));
        System.out.println("\nCumulative Frequency: " +
H.getCumulativeFrequency());

        System.out.println("\nSorted Probability of Characters");
        System.out.println(H.sortDownProbability());
        System.out.println("\nSum of Probabilities : " +
H.getSumOfProbability());

        BorderPane BP = new BorderPane();
        Pane leftP = new Pane();
        Pane centerP = new Pane();

        double widthCanvas = 800; double heightCanvas = 400;
        double widthLeftCanvas = 0.275 * widthCanvas;
        double widthCenterCanvas = widthCanvas - widthLeftCanvas;

        leftP.getChildren().add(addCanvasLegend(widthCenterCanvas,
heightCanvas, H));
        BP.setLeft(leftP);

        if (isPiechart){
```

```
            centerP.getChildren().add(addCanvasPieChart(widthCenterCanvas,
heightCanvas, H));
        }
        BP.setCenter(centerP);

        Scene SC = new Scene(BP, widthCanvas, heightCanvas,
MyColor.WHITE.getJavaFXColor());
        PS.setTitle("Frequency of Characters: " + Title);
        PS.setScene(SC);
        PS.show();
    }

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

MyShapeApplication.java

```
package com.example.assignment1;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.text.Font;
import javafx.stage.Stage;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Paths;
import java.util.*;
import java.util.List;

public class MyShapeApplication extends Application{

    Integer N;
    Integer M;
    double startAngle;
    double scale;
    String Title;
    String filename;
    Scanner input;

    List<String> piechartInputs = new ArrayList();

    public VBox addLeftVBox(double widthLeftCanvas, double heightLeftCanvas,
```

```
TilePane TP, MyColor color){

        VBox VB = new VBox();
        VB.setPrefWidth(widthLeftCanvas);
        VB.setPadding(new Insets(5));

        Label lblMyColorPalette = new Label("MyColor Palette");
        lblMyColorPalette.setPrefWidth(widthLeftCanvas);
        lblMyColorPalette.setTextFill(MyColor.WHITE.getJavaFXColor());
        lblMyColorPalette.setBackground(new Background(new
BackgroundFill(Optional.ofNullable(color).orElse(MyColor.GREY).getJavaFXColor
(), CornerRadii.EMPTY, Insets.EMPTY)));

        VB.getChildren().addAll(lblMyColorPalette, TP);

        return VB;
    }

    public HBox addTopHBox(double widthTopCanvas, double heightTopCanvas,
double widthCenterCanvas, double heightCenterCanvas, double widthRightCanvas,
BorderPane BP, MyColorPalette CP, TilePane TP) throws FileNotFoundException {

        HBox HB = new HBox();
        HB.setPrefWidth(widthTopCanvas);
        HB.setPadding(new Insets(5, 5, 5, 5));

        String [] nameImages = new String [] {"Line", "Oval", "Rectangle",
"Piechart"};
        String pathFile =
"C:\\Users\\zixua\\IdeaProjects\\Assignment2\\Geometric Shapes\\";

        Deque<MyShape> stackMyShapes = new ArrayDeque<MyShape>();
        for (String nameImage : nameImages){
            String nameFile = pathFile + nameImage + ".PNG";
            ImageView geometricImage = new ImageView(new Image(new
FileInputStream(nameFile), heightTopCanvas, heightTopCanvas, true, false));

            geometricImage.setOnMouseClicked(e -> {
                switch(nameImage) {
                    /*(case "Arc":
                        dialogArc(widthCenterCanvas, heightCenterCanvas, BP,
CP, TP, stackMyShapes);
                        break;
                    */

                    case "Line":
                        dialogLine(widthCenterCanvas, heightCenterCanvas, BP,
CP, TP, stackMyShapes);
                        break;

                    case "Oval":
                        dialogOval(widthCenterCanvas, heightCenterCanvas, BP,
CP, TP, stackMyShapes);
                        break;

                    /*case "Polygon":
                        dialogPolygon(widthCenterCanvas, heightCenterCanvas,
```

```
BP, CP, TP, stackMyShapes);
                        break;
                    */

                    case "Rectangle":
                        dialogRectangle(widthCenterCanvas,
heightCenterCanvas, BP, CP, TP, stackMyShapes);
                        break;

                    /*
                    case "Triangle":
                        dialogTriangle(widthCenterCanvas, heightCenterCanvas,
BP, CP, TP, stackMyShapes);
                        break;
                    */

                    case "Piechart":
                        dialogPiechart(widthCenterCanvas, heightCenterCanvas,
widthRightCanvas, BP, CP, TP);
                        break;
                }
            });

            HB.getChildren().add(geometricImage);
        }

        return HB;
    }

    /*
    public void dialogArc(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShape){

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyOval");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20, 100, 10, 10));

        TextField xCenter = new TextField(); TextField yCenter = new
TextField();
        TextField width = new TextField(); TextField height = new
TextField();
        TextField startingAngle = new TextField(); TextField extentAngle =
new TextField();

        gridDialog.add(new Label("Oval center"), 0, 0);
        gridDialog.add(xCenter, 1, 0);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 0);
```

```
        gridDialog.add(yCenter, 1, 1);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas width"),
2, 1);
        gridDialog.add(new Label("Oval Width"), 0,2);
        gridDialog.add(width, 1,2);
        gridDialog.add(new Label("Width as fraction of canvas height"), 2,
2);
        gridDialog.add(new Label("Oval height"), 0, 3);
        gridDialog.add(height, 1, 3);
        gridDialog.add(new Label("Height as fraction of canvas height"), 2,
3);
        gridDialog.add(new Label("Arc start angle"), 0, 4);
        gridDialog.add(startingAngle, 1, 4);
        gridDialog.add(new Label("In degrees"), 2, 4);
        gridDialog.add(new Label("Arc [extent] angle"), 0, 5);
        gridDialog.add(extentAngle, 1, 5);
        gridDialog.add(new Label("In degrees"), 2, 5);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> xCenter.requestFocus());

        List <String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(xCenter.getText());
geometricImageInputs.add(yCenter.getText());
                geometricImageInputs.add(width.getText());
geometricImageInputs.add(height.getText());
                geometricImageInputs.add(startingAngle.getText());
geometricImageInputs.add(extentAngle.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint pLTC = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            double w = Double.parseDouble(geometricImageInputs.get(2)) *
widthCenterCanvas;
            double h = Double.parseDouble(geometricImageInputs.get(3)) *
heightCenterCanvas;
            double startAngle =
Double.parseDouble(geometricImageInputs.get(4));
            double arcAngle =
Double.parseDouble(geometricImageInputs.get(5));

            TP.setOnMouseClicked(e -> {
```

```
                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyOval O = new MyOval(pTLC, w, h, color);
                        MyArc A = new MyArc(O, startAngle, arcAngle, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        O.stroke(GC);
                        A.draw(GC);
                        A.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(A);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });

    }
    */

    public void dialogLine(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShape) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyTriangle");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20, 100, 10, 10));

        TextField x1 = new TextField();
        TextField y1 = new TextField();
        TextField x2 = new TextField();
        TextField y2 = new TextField();

        gridDialog.add(new Label("End points as fraction of canvas width and
height"), 0, 0);
        gridDialog.add(new Label("Point 1"), 0, 1);
        gridDialog.add(x1, 1, 1);
        gridDialog.add(y1, 1, 2);
        gridDialog.add(new Label("Point 2"), 0, 2);
        gridDialog.add(x2, 1, 2);
        gridDialog.add(y2, 2, 2);
```

```
        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> x1.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK) {
                geometricImageInputs.add(x1.getText());
geometricImageInputs.add(y1.getText());
                geometricImageInputs.add(x2.getText());
geometricImageInputs.add(y2.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint p = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            MyPoint q = new
MyPoint(Double.parseDouble(geometricImageInputs.get(2)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(3)) * heightCenterCanvas, null);

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyLine L = new MyLine(p, q, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        L.draw(GC);
                        L.getMyBoundingRectangle().stroke(GC);

                        stackMyShape.push(L);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }

    public void dialogOval(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
```

```
Deque<MyShape> stackMyShapes){

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyOval");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10, 10));

        TextField xCenter = new TextField(); TextField yCenter = new
TextField();
        TextField width = new TextField();
        TextField height = new TextField();

        gridDialog.add(new Label("Center"), 0,0);
        gridDialog.add(xCenter, 1, 0);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 0);
        gridDialog.add(yCenter, 1, 1);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas width"),
2, 1);
        gridDialog.add(new Label("width"), 0, 2);
        gridDialog.add(width, 1,2);
        gridDialog.add(new Label("Width as fraction of canvas width"), 2, 2);
        gridDialog.add(new Label("Height"), 0,3);
        gridDialog.add(height, 1,3);
        gridDialog.add(new Label("Height as fraction of canvas height"), 2,
3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> xCenter.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(xCenter.getText());
geometricImageInputs.add(yCenter.getText());
geometricImageInputs.add(width.getText());
geometricImageInputs.add(height.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional <List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
```

```
        Result.ifPresent(event -> {
            MyPoint pTLC = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            double w = Double.parseDouble(geometricImageInputs.get(2)) *
widthCenterCanvas;
            double h = Double.parseDouble(geometricImageInputs.get(3)) *
heightCenterCanvas;

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyOval O = new MyOval(pTLC, w, h, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        O.draw(GC);
                        O.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(O);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }

    /*
    public void dialogPolygon(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyPolygon");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        TextField numberSides = new TextField();
        TextField xCenter = new TextField(); TextField yCenter = new
TextField();
        TextField radius = new TextField();

        gridDialog.add(new Label("Number of sides"), 0 , 0);
        gridDialog.add(numberSides, 1, 0);
```

```
        gridDialog.add(new Label("Maximum number of sides: " +
MyShapeInterface.maxNumberPolygonSides), 2, 0);
        gridDialog.add(new Label("Center"), 0 ,1);
        gridDialog.add(xCenter, 1, 1);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 1);
        gridDialog.add(yCenter, 1, 2);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas width"),
2, 2);
        gridDialog.add(new Label("Radius"), 0, 3);
        gridDialog.add(radius, 1, 3);
        gridDialog.add(new Label ("Radius as fraction of the minimum of
canvas width and canvas height"), 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> numberSides.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(numberSides.getText());
geometricImageInputs.add(xCenter.getText());
geometricImageInputs.add(yCenter.getText());
geometricImageInputs.add(radius.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            int N = Integer.parseInt(geometricImageInputs.get(0));
            MyPoint center = new
MyPoint(Double.parseDouble(geometricImageInputs.get(1)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(2)) * heightCenterCanvas, null);
            double r = Double.parseDouble(geometricImageInputs.get(3)) *
Math.min(widthCenterCanvas, heightCenterCanvas);

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyPolygon Y = new MyPolygon(N, center, r, color);

                        GC.clearRect(0,0,widthCenterCanvas,
heightCenterCanvas);
                        Y.draw(GC);
                        Y.getMyBoundingRectangle().stroke(GC);
```

```
                        stackMyShapes.push(Y);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }
    */

    public void dialogRectangle(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyRectangle");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        TextField xPTLC = new TextField(); TextField yPTLC = new TextField();
        TextField width = new TextField();
        TextField height = new TextField();

        gridDialog.add(new Label("Top Left Corner Point"), 0, 0);
        gridDialog.add(xPTLC, 1, 0);
        gridDialog.add(new Label("x-Coordinate as fraction of canvas width"),
2, 0);
        gridDialog.add(yPTLC, 1, 1);
        gridDialog.add(new Label("y-Coordinate as fraction of canvas
height"), 2, 1);
        gridDialog.add(new Label("Width"), 0 , 2);
        gridDialog.add(width, 1, 2);
        gridDialog.add(new Label("As fraction of canvas width"), 2, 2);
        gridDialog.add(new Label("Height"), 0, 3);
        gridDialog.add(height, 1, 3);
        gridDialog.add(new Label("As fraction of canvas height"), 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> xPTLC.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(xPTLC.getText());
geometricImageInputs.add(xPTLC.getText());
geometricImageInputs.add(width.getText());
```

```
geometricImageInputs.add(height.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint pTLC = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            double w = Double.parseDouble(geometricImageInputs.get(2)) *
widthCenterCanvas;
            double h = Double.parseDouble(geometricImageInputs.get(3)) *
heightCenterCanvas;

            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile : TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyRectangle R = new MyRectangle(pTLC, w, h, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        R.draw(GC);
                        R.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(R);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }

    /*
    public void dialogTriangle(double widthCenterCanvas, double
heightCenterCanvas, BorderPane BP, MyColorPalette CP, TilePane TP,
Deque<MyShape> stackMyShapes) {

        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("MyTriangle");
        dialog.setHeaderText(null);

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);
```

```
        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20,100,10,10));

        TextField x1 = new TextField(); TextField y1 = new TextField();
        TextField x2 = new TextField(); TextField y2 = new TextField();
        TextField x3 = new TextField(); TextField y3 = new TextField();

        gridDialog.add(new Label("Triangle vertices as fraction of canvas
width and height"), 0, 0);
        gridDialog.add(new Label("Vertex 1"), 0 , 1);
        gridDialog.add(x1, 1, 1);
        gridDialog.add(y1, 1, 2);
        gridDialog.add(new Label("Vertex 2"), 0 , 2);
        gridDialog.add(x2, 1, 2);
        gridDialog.add(y2, 2, 2);
        gridDialog.add(new Label("Vertex 3"), 0, 3);
        gridDialog.add(x3, 1, 3);
        gridDialog.add(y3, 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        Platform.runLater(() -> x1.requestFocus());

        List<String> geometricImageInputs = new ArrayList();
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                geometricImageInputs.add(x1.getText());
geometricImageInputs.add(y1.getText());
                geometricImageInputs.add(x2.getText());
geometricImageInputs.add(y2.getText());
                geometricImageInputs.add(x3.getText());
geometricImageInputs.add(y3.getText());
                return geometricImageInputs;
            }

            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Pane centerPane = new Pane();

        Canvas CV = new Canvas(widthCenterCanvas, heightCenterCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();
        Result.ifPresent(event -> {
            MyPoint p = new
MyPoint(Double.parseDouble(geometricImageInputs.get(0)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(1)) * heightCenterCanvas, null);
            MyPoint q = new
MyPoint(Double.parseDouble(geometricImageInputs.get(2)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(3)) * heightCenterCanvas, null);
            MyPoint r = new
MyPoint(Double.parseDouble(geometricImageInputs.get(4)) * widthCenterCanvas,
Double.parseDouble(geometricImageInputs.get(5)) * heightCenterCanvas, null);
```

```
            TP.setOnMouseClicked(e -> {

                MyColor color = CP.getColorPicked(); String tileId =
color.toString();
                for (Node tile: TP.getChildren()) {
                    if (tile.getId() == tileId) {
                        MyTriangle T = new MyTriangle(p, q, r, color);

                        GC.clearRect(0, 0, widthCenterCanvas,
heightCenterCanvas);
                        T.draw(GC);
                        T.getMyBoundingRectangle().stroke(GC);

                        stackMyShapes.push(T);
                        break;
                    }
                }
            });

            centerPane.getChildren().add(CV);
            BP.setCenter(centerPane);
        });
    }
    */

    public void dialogPiechart(double widthCenterCanvas, double
heightCenterCanvas, double widthRightCanvas, BorderPane BP, MyColorPalette
CP, TilePane TP){
        Dialog<List<String>> dialog = new Dialog<>();
        dialog.setTitle("Pie Chart");
        dialog.setHeaderText(null);
        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK,
ButtonType.CANCEL);

        GridPane gridDialog = new GridPane();
        gridDialog.setHgap(10);
        gridDialog.setVgap(10);
        gridDialog.setPadding(new Insets(20, 150, 10, 10));

        TextField numberEvents = new TextField();
        TextField totalNumberEvents = new TextField();
        TextField startingAngle = new TextField();

        ComboBox title = new ComboBox();
        title.getItems().addAll("Moby Dick");

        ToggleGroup group = new ToggleGroup();
        //RadioButton radioRandom = new RadioButton("Random");
        //radioRandom.setToggleGroup(group);
        RadioButton radioPalette = new RadioButton("Palette");
        radioPalette.setToggleGroup(group);

        gridDialog.add(new Label("Display"), 0, 0);
        gridDialog.add(numberEvents, 1, 0);
        gridDialog.add(new Label("Total"), 2, 0);
        gridDialog.add(totalNumberEvents, 3, 0);
```

```
        gridDialog.add(new Label("Starting Angle"), 0, 1);
        gridDialog.add(startingAngle, 1, 1);
        gridDialog.add(new Label("Title"), 0, 2);
        gridDialog.add(title, 1, 2);

        gridDialog.add(new Label("Colors"), 0, 3);
        gridDialog.add(radioPalette, 1, 3);
        //gridDialog.add(radioRandom, 2, 3);

        dialog.getDialogPane().setContent(gridDialog);

        //Platform.runLater(() -> radioRandom.setSelected(true));

        Platform.runLater(() -> numberEvents.requestFocus());

        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == ButtonType.OK){
                //piechartInputs.add(radioRandom.getText());
                piechartInputs.add(numberEvents.getText());
piechartInputs.add(totalNumberEvents.getText());
                piechartInputs.add(startingAngle.getText());
piechartInputs.add(title.getValue().toString());
                return piechartInputs;
            }
            return null;
        });

        Optional<List<String>> Result = dialog.showAndWait();

        Result.ifPresent(event -> {
            this.N = Integer.parseInt(piechartInputs.get(0));
            this.M = Integer.parseInt(piechartInputs.get(1));
            this.startAngle = Double.parseDouble(piechartInputs.get(2));
            this.Title = piechartInputs.get(3);
            this.filename =
"C:\\Users\\zixua\\IdeaProjects\\Assignment3\\Texts\\" + Title + ".txt";

            openFile();
            String w = readFile();
            closeFile();

            HistogramAlphabet H = new HistogramAlphabet(w);
            Map<Character, Integer> sortedFrequency = H.sortDownFrequency();

            MyPoint center = new MyPoint(0.4 * widthCenterCanvas, 0.5 *
heightCenterCanvas, null);

            double diameterPieChart = 0.60 * Math.min(widthCenterCanvas,
heightCenterCanvas);
            HistogramAlphabet.MyPieChart pieChart = H.new MyPieChart(N, M,
center, diameterPieChart, diameterPieChart, startAngle);

            Pane rightPane = new Pane();
            rightPane.getChildren().add(addCanvasLegend(widthRightCanvas,
heightCenterCanvas, H));
            BP.setRight(rightPane);
```

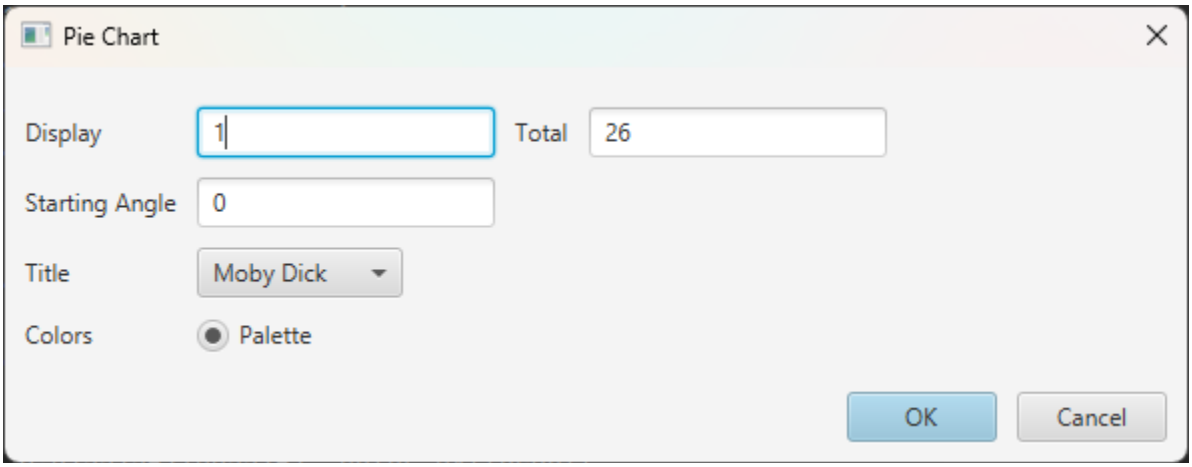
```
            Pane centerPane = new Pane();
            centerPane.getChildren().add(addCanvasPieChart(widthCenterCanvas,
heightCenterCanvas, H));
            BP.setCenter(centerPane);
        });
    }

    public void openFile() {
        try {
            input = new Scanner(Paths.get(filename));
        } catch (IOException ioException){
            System.err.println("File is not found");
        }
    }

    public String readFile() {
        String w = "";
        try {
            while (input.hasNext()) {
                w += input.nextLine().replaceAll("[^a-zA-
Z]","").toLowerCase();
            }
        }
        catch (NoSuchElementException elementException) {
            System.err.println("Invalid input! Terminating...");
        }
        catch (IllegalStateException stateException) {
            System.out.println("Error processing file! Terminating...");
        }
        return w;
    }

    public void closeFile() {
        if (input != null) input.close();
    }

    public Canvas addCanvasLegend(double widthCanvas, double heightCanvas,
HistogramAlphabet H) {
        String information;

        Canvas CV = new Canvas(widthCanvas, heightCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();

        MyColor colorLeftCanvas = MyColor.LINEN;
        GC.setFill(colorLeftCanvas.getJavaFXColor());
        GC.fillRect(0.0, 0.0, widthCanvas, heightCanvas);

        double xText = 20; double yText = 0.03625 * heightCanvas;
        MyColor colorStroke = MyColor.GRAY;
        GC.setStroke(colorStroke.invertColor());
        GC.setFont(Font.font("Calibri", 13));
        GC.strokeText("Frequency: Cumulative " + H.getCumulativeFrequency(),
xText, yText);

        Map <Character, Integer> sortedFrequency = H.sortDownFrequency();

        Double yStep = 0.03625 * heightCanvas;
```

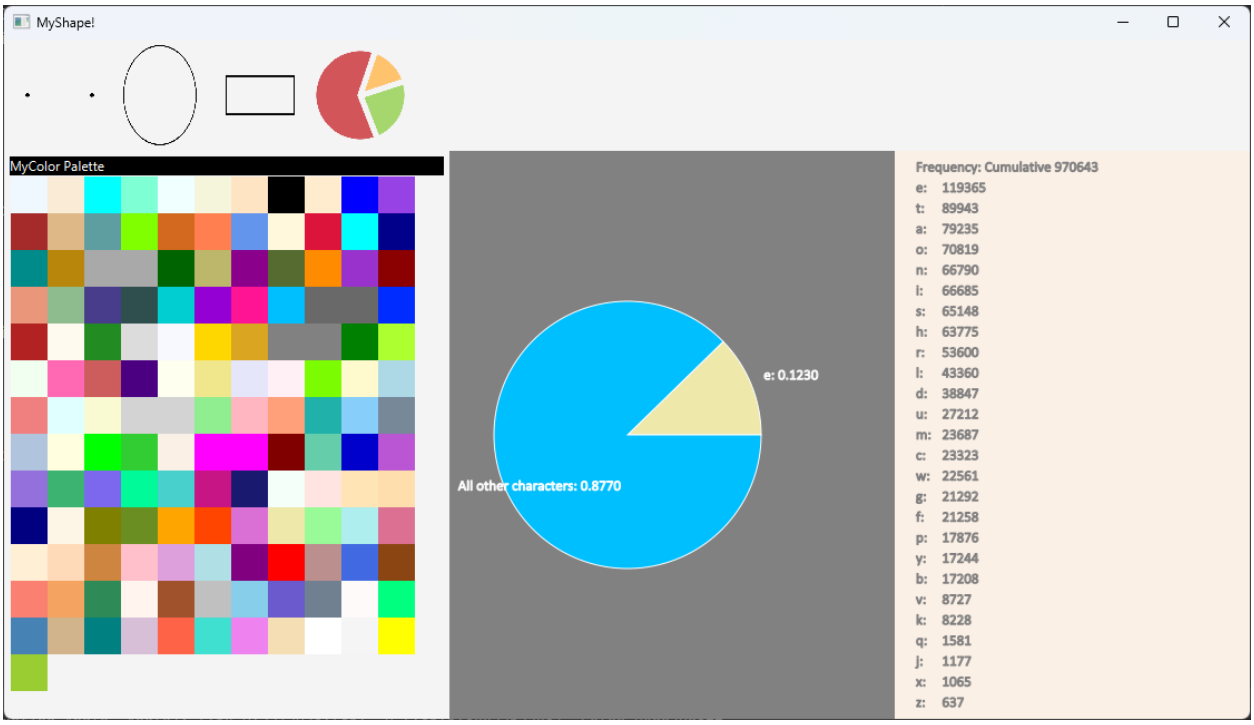
```
        for (Character K: sortedFrequency.keySet()){
            yText += yStep;
            information = K + ":\t" + sortedFrequency.get(K);
            GC.strokeText(information, xText, yText);
        }
        return CV;
    }

    public Canvas addCanvasPieChart(double widthCanvas, double heightCanvas,
HistogramAlphabet H){
        Canvas CV = new Canvas(widthCanvas, heightCanvas);
        GraphicsContext GC = CV.getGraphicsContext2D();

        MyPoint center = new MyPoint(0.4 * widthCanvas, 0.5 * heightCanvas,
null);

        double diameterPieChart = 0.60 * Math.min(widthCanvas, heightCanvas);
        HistogramAlphabet.MyPieChart pieChart = H.new MyPieChart(N, M,
center, diameterPieChart, diameterPieChart, startAngle);
        Map <Character, Slice> slices = pieChart.getMyPieChart();

        System.out.println("\nPie Chart");
        slices.forEach(((K,V) -> System.out.println(K + ": " +
slices.get(K))));

        double sumOfAngles = 0.0;
        for (Character key : slices.keySet()){
            sumOfAngles += slices.get(key).getArcAngle();
        }
        System.out.println("\nSum Of Angles: " + sumOfAngles);

        pieChart.draw(GC);

        return CV;
    }

    public Canvas addCenterCanvas(double widthCenterCanvas, double
heightCenterCanvas, MyShape S1, MyShape S2, MyColor color){

        return S1.drawIntersectMyShapes(widthCenterCanvas,
heightCenterCanvas, S1, S2, color);
    }

    @Override
    public void start(Stage PS) throws FileNotFoundException {

        double widthCanvas = 800.0; double heightCanvas = 600.0;

        BorderPane BP = new BorderPane();
        Pane topPane = new Pane(); Pane leftPane = new Pane();

        double widthLeftCanvas = 0.5 * widthCanvas; double heightTopCanvas =
0.15 * heightCanvas; double widthRightCanvas = 0.4 * widthCanvas;
        double widthCenterCanvas = widthCanvas - widthLeftCanvas;
        double heightCenterCanvas = heightCanvas - heightTopCanvas;

        MyColorPalette CP = new MyColorPalette(widthLeftCanvas,
```

```
heightCenterCanvas);
        TilePane TP = CP.getPalette();

        Scene SC = new Scene(BP, widthCanvas, heightCanvas,
MyColor.WHITE.getJavaFXColor());
        PS.setTitle("MyShape!");
        PS.setScene(SC);

        MyPoint r = new MyPoint(widthCenterCanvas, 0.5 * heightCenterCanvas,
null);
        MyPoint n = new MyPoint(widthCenterCanvas, heightCenterCanvas, null);
        MyPoint v = new MyPoint(0.5 * widthCenterCanvas, heightCenterCanvas,
null);
        MyPoint h = new MyPoint(0, widthCenterCanvas, null);
        MyPoint u = new MyPoint(0, 0.5 * heightCenterCanvas, null);
        MyPoint t = new MyPoint();
        MyPoint p = new MyPoint(0.5 * widthCenterCanvas, 0, null);
        MyPoint k = new MyPoint(widthCenterCanvas, 0, null);

        MyPoint q = new MyPoint(0.5 * widthCenterCanvas, 0.5 *
heightCenterCanvas, null);

        System.out.println("\nAngle of the line extending from" + r + "to
[origin]" + q + ": " + r.angleX(q));
        System.out.println("Angle of the line extending from" + n + "to
[origin]" + q + ": " + n.angleX(q));
        System.out.println("Angle of the line extending from" + v + "to
[origin]" + q + ": " + v.angleX(q));
        System.out.println("Angle of the line extending from" + h + "to
[origin]" + q + ": " + h.angleX(q));
        System.out.println("Angle of the line extending from" + u + "to
[origin]" + q + ": " + u.angleX(q));
        System.out.println("Angle of the line extending from" + t + "to
[origin]" + q + ": " + t.angleX(q));
        System.out.println("Angle of the line extending from" + p + "to
[origin]" + q + ": " + p.angleX(q));
        System.out.println("Angle of the line extending from" + k + "to
[origin]" + q + ": " + k.angleX(q));

        MyLine L1 = new MyLine(p, q, null);
        System.out.println("\n" + L1);

        MyLine L2 = new MyLine(r, q, null);
        System.out.println("\n" + L2);

        MyRectangle R = new MyRectangle(p, 0.5 * widthCenterCanvas, 0.5 *
widthCenterCanvas, MyColor.LIME);
        System.out.println("\n" + R);

        MyOval O = new MyOval(q, 0.5 * widthCenterCanvas, 0.5 *
heightCenterCanvas, MyColor.GOLD);
        System.out.println("\n" + O);

        double radius = 0.25 * Math.min(widthCenterCanvas,
heightCenterCanvas);
        MyCircle C = new MyCircle(q, radius, MyColor.GREEN);
        System.out.println("\n" + C);
```

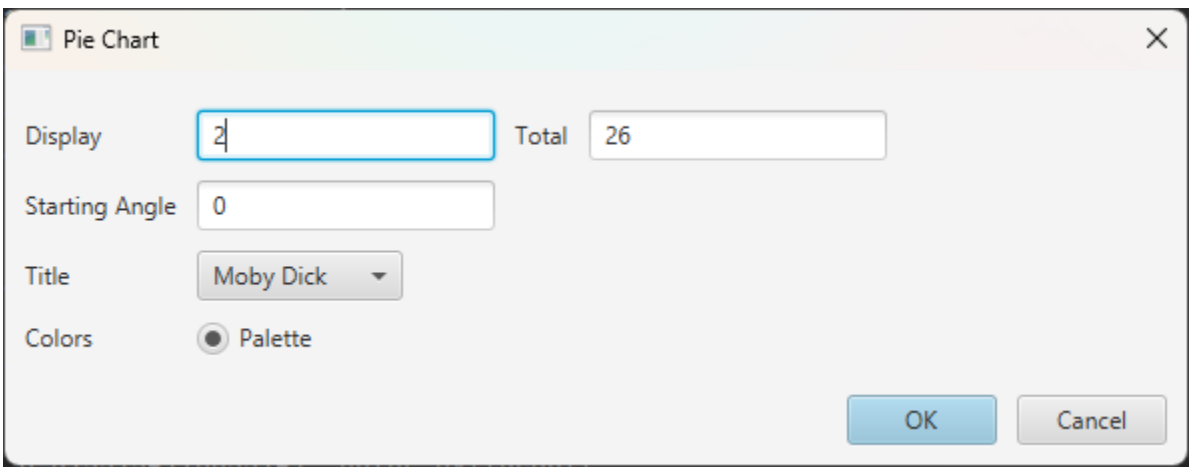
```
        /*
        MyTriangle T = new MyTriangle(t, r, v, MyColor.CYAN);
        System.out.println("\n" + T);

        double startAngle = 40;
        double arcAngle = 210;
        MyArc A = new MyArc(O, startAngle, arcAngle, MyColor.GREY);
        System.out.println("\n" + A);
        */

        MyShapeInterface [] shapes = new MyShape [5];
        shapes[0] = L1;
        shapes[1] = L2;
        shapes[2] = R;
        shapes[3] = O;
        shapes[4] = C;
        //shapes[5] = T;
        //shapes[6] = A;
        //shape[7] = Y;

        for (MyShapeInterface shape: shapes){
            System.out.println("\n" + shape);
            System.out.println(shape.getMyBoundingRectangle());
        }

        MyRectangle RR = (MyRectangle) shapes[2];
        MyOval OO = (MyOval) shapes[3];
        MyCircle CC = (MyCircle) shapes[4];
        //MyTriangle TT = (MyTriangle) shapes[5];
        //MyArc AA = (MyArc) shapes[6];
        //MyPolygon PP = (MyPolygon) shapes[7];

        //System.out.println("\n" + MyShapeInterface.intersectMyShapes(TT,
AA));

        topPane.getChildren().add(addTopHBox(widthCanvas, heightTopCanvas,
widthCenterCanvas, heightCenterCanvas, widthRightCanvas, BP, CP, TP));
        BP.setTop(topPane);

        leftPane.getChildren().add(addLeftVBox(widthLeftCanvas,
heightCenterCanvas, TP, MyColor.BLACK));
        BP.setLeft(leftPane);

        PS.show();
    }

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

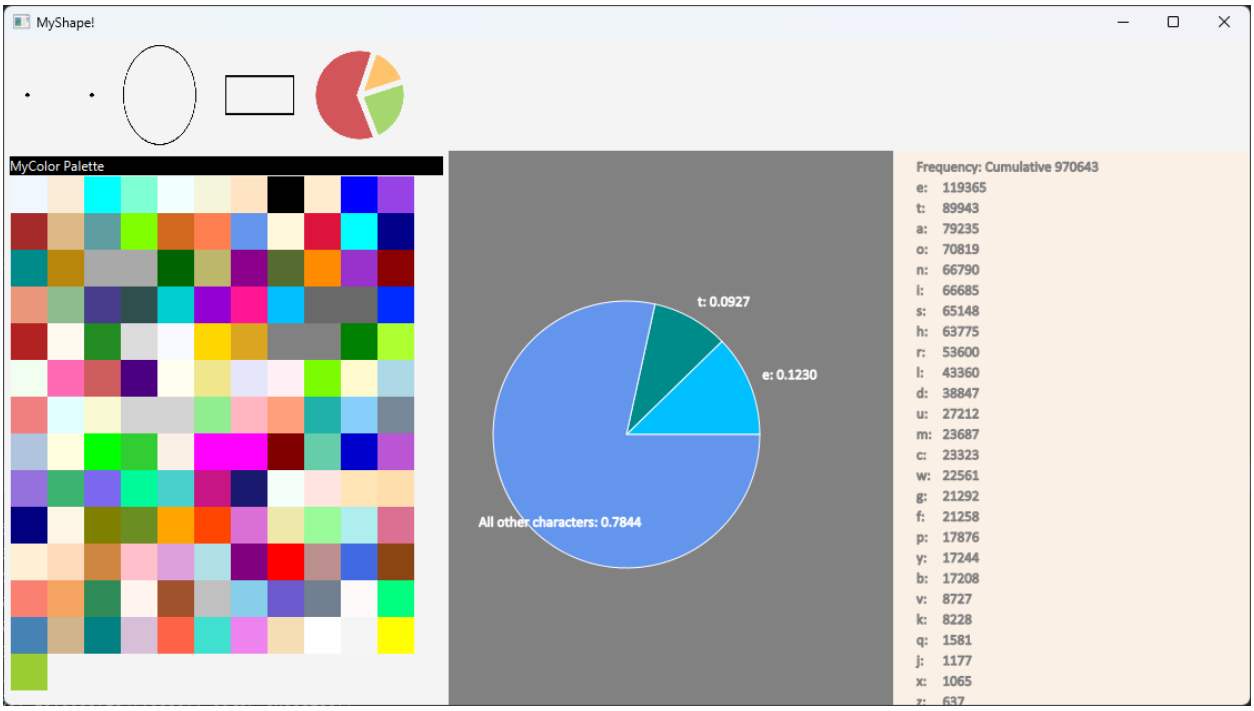
Output of the code:


Pie Chart
Display
1
Total
26
Starting Angle
0
Title
Moby Dick
Colors
Palette
OK
Cancel

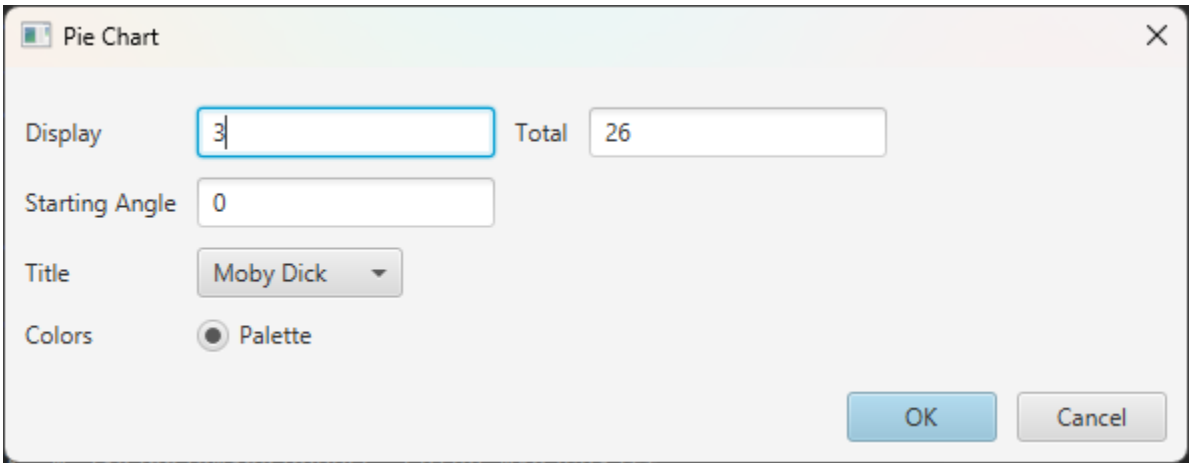

MyShape!
MyColor Palette
e: 0.1230
All other characters: 0.8770
Frequency: Cumulative 970643
e: 119365
t: 89943
a: 79235
o: 70819
n: 66790
i: 66685
s: 65148
h: 63775
r: 53600
l: 43360
d: 38847
u: 27212
m: 23687
c: 23323
w: 22561
g: 21292
f: 21258
p: 17876
y: 17244
b: 17208
v: 8727
k: 8228
q: 1581
j: 1177
x: 1065
z: 637

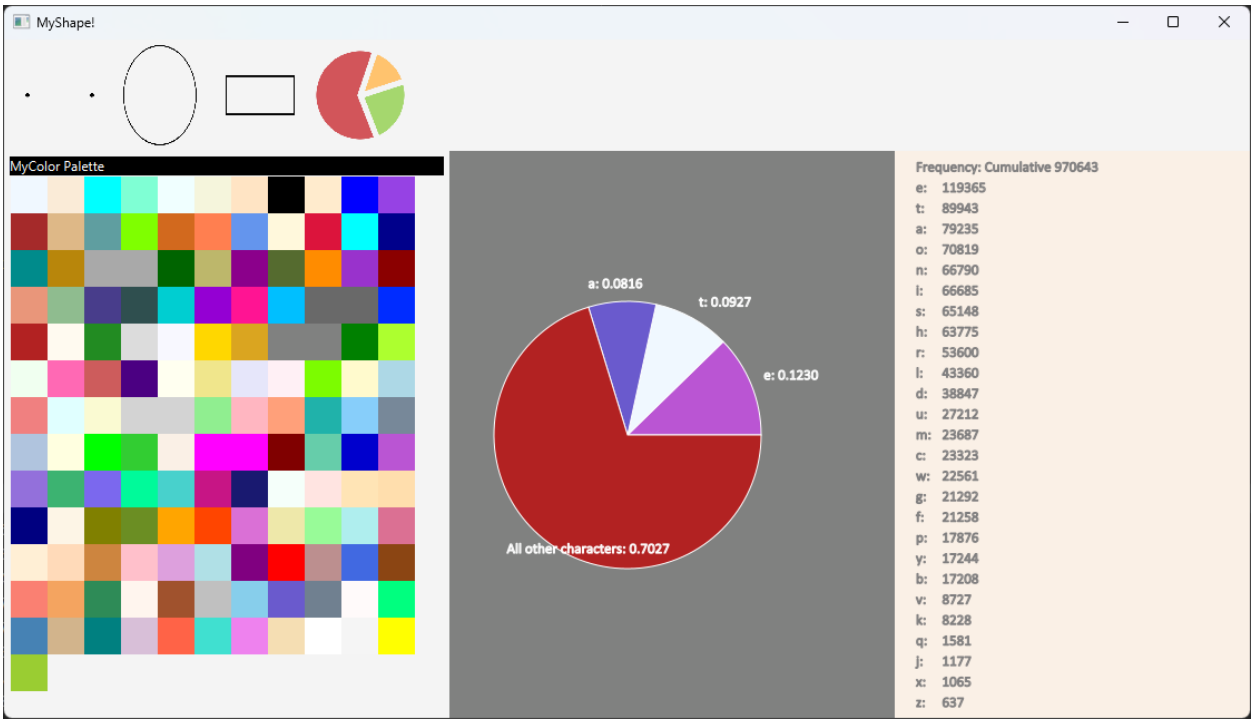
```
Run: MyShapeApplication
Pie Chart
a: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (77.62985979397163, 29.387323660707388), Color 0xf0f8ffff
b: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (345.6751864485706, 6.382243523107879), Color 0xf0fff0ff
c: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (299.85047025528434, 8.650224644900034), Color 0xda70d6ff
d: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (266.56474110460795, 14.407892500126207), Color 0xdeb887ff
e: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (0.0, 44.27106567502161), Color 0xeee8aaff
f: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (324.7652535484209, 7.884340586600842), Color 0xff00ffff
g: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (316.86830276425, 7.896950784170906), Color 0x6495edff
h: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (206.9500320921286, 23.653392647966635), Color 0xfaf0e6ff
i: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (158.05473279053163, 24.732677204698334), Color 0xffdeadff
j: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.93221297634653, 0.43653536882252286), Color 0xda70d6ff
k: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (355.294170977383, 3.0516678119555802), Color 0xa52a2aff
l: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (250.48303032113762, 16.081710783470342), Color 0xff00ffff
m: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (291.06524231875153, 8.785227936532793), Color 0x228b22ff
n: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (133.28311232863163, 24.771620461900003), Color 0xffdab9ff
o: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (107.01718345467901, 26.265928873952628), Color 0x8a2be2e1
p: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (332.64959413502174, 6.629996816543261), Color 0x3cb371ff
q: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.34583878933853, 0.5863741870079937), Color 0xffe4c4ff
r: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (230.60342474009494, 19.87960558104267), Color 0x8b4513ff
s: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (182.78740999522995, 24.162622096898655), Color 0x7cfc00ff
t: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (44.27106567502161, 33.358794118950016), Color 0xfffaf0ff
u: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (280.97263360473414, 10.092608714017741), Color 0x8fbc8fff
v: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (352.0574299716785, 3.2367410057044665), Color 0x808000ff
w: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (308.50069490018467, 8.367607864065368), Color 0xe6e6faff
x: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.3687483451691, 0.3949958944740755), Color 0xeee8aaff
y: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (339.279590951565, 6.395595497005594), Color 0xdda0ddff
z: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.76374423964313, 0.23625576035679444), Color 0xdc143cff

Sum Of Angles: 360.0
```



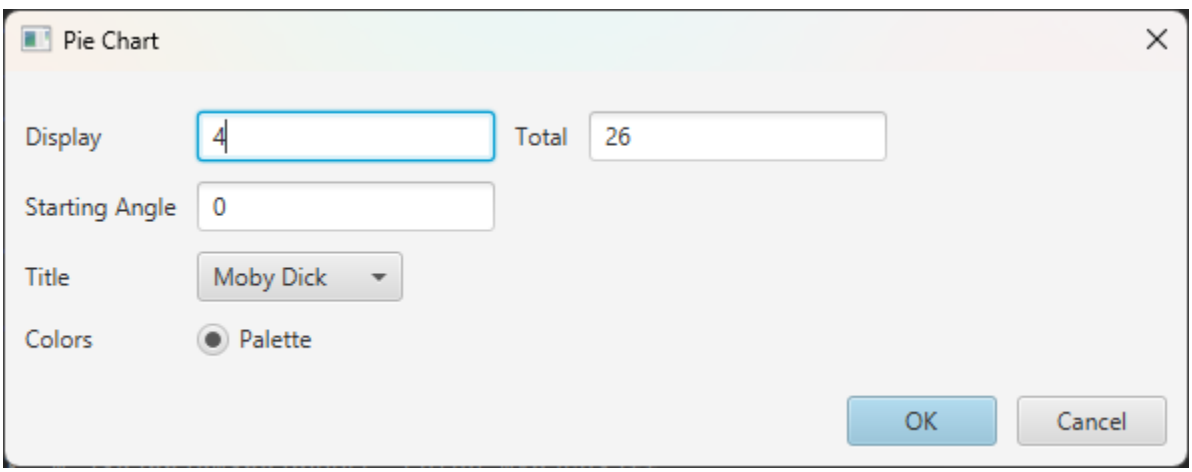


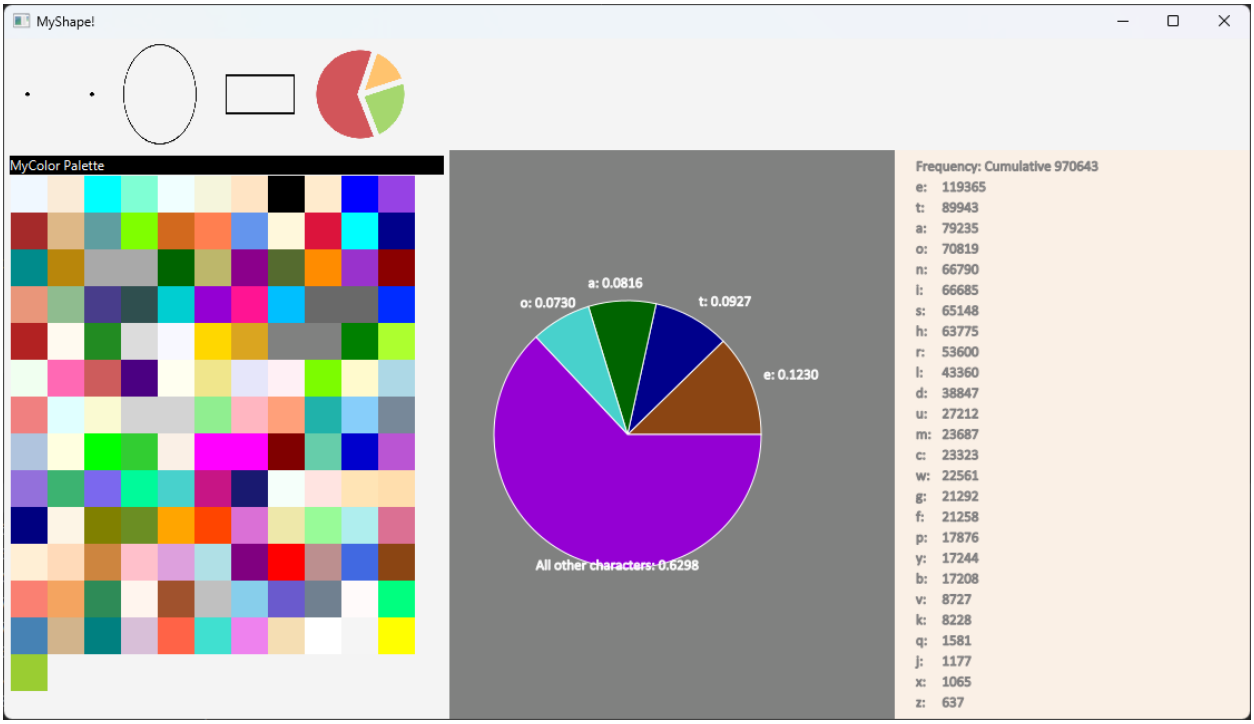

```
Run: MyShapeApplication
Pie Chart
a: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (77.62985979397163, 29.387323660707388), Color 0xffc0cbff
b: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (345.6751864485706, 6.382243523107879), Color 0x7cfc00ff
c: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (299.85047025528434, 8.650224664490034), Color 0x008b8bff
d: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (266.56474110460795, 14.407892500126207), Color 0xe0ffffff
e: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (0.0, 44.27106567502161), Color 0x00bfffff
f: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (324.7652535484209, 7.884340586600842), Color 0x8fbc8fff
g: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (316.86830276425, 7.896950784170906), Color 0xffffe0ff
h: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (206.9500320921286, 23.653392647966635), Color 0x808180ff
i: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (158.05473279053163, 24.732677204698334), Color 0x9400d3ff
j: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.93221297634653, 0.43653536882252286), Color 0xe6e6faff
k: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (355.294170977383, 3.0516678119555802), Color 0xbdb76bff
l: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (250.48303032113762, 16.081710783470342), Color 0xff69b4ff
m: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (291.06524231875153, 8.785227936532793), Color 0x7fffd4ff
n: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (133.28311232863163, 24.771620461900003), Color 0x228b22ff
o: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (107.01718345467901, 26.265928873952628), Color 0x8b0000ff
p: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (332.64959413502174, 6.629996816543261), Color 0xfffe0ff
q: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.34583878933853, 0.5863741870079937), Color 0xf0ffffff
r: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (230.60342474009494, 19.87960558104267), Color 0x708090ff
s: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (182.78740999522995, 24.162622096898655), Color 0xff0000ff
t: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (44.27106567502161, 33.358794118950016), Color 0x008b8bff
u: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (280.97263360473414, 10.09260871401741), Color 0x9370dbff
v: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (352.0574299716785, 3.2367410057044665), Color 0xc0c0c0ff
w: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (308.50069490018467, 8.367607864065368), Color 0x0000cdff
x: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.3687483451691, 0.3949958944740755), Color 0xff69b4ff
y: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (339.279590951565, 6.395595497005594), Color 0x696969ff
z: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.76374423964313, 0.23625576035679444), Color 0xf5fffaff

Sum Of Angles: 360.0
```


Pie Chart
Display
3
Total
26
Starting Angle
0
Title
Moby Dick
Colors
Palette
OK
Cancel


MyShape!
MyColor Palette
a: 0.0816
t: 0.0927
e: 0.1230
All other characters: 0.7027
Frequency: Cumulative 970643
e: 119365
t: 89943
a: 79235
o: 70819
n: 66790
i: 66685
s: 65148
h: 63775
r: 53600
l: 43360
d: 38847
u: 27212
m: 23687
c: 23323
w: 22561
g: 21292
f: 21258
p: 17876
y: 17244
b: 17208
v: 8727
k: 8228
q: 1581
j: 1177
x: 1065
z: 637

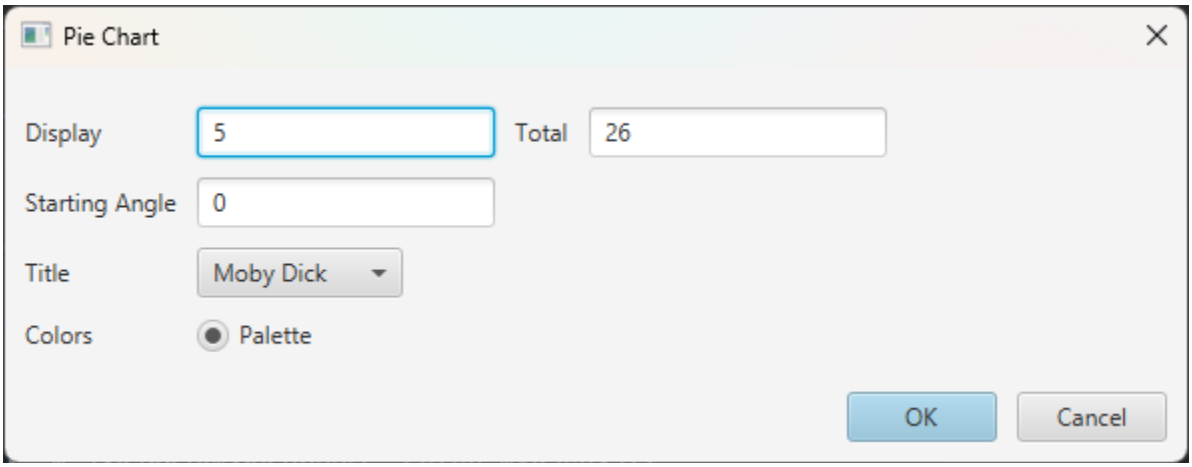
```
Run: MyShapeApplication
Pie Chart
a: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (77.62985979397163, 29.387323660707388), Color 0x6a5acdff
b: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (345.6751864485706, 6.382243523107879), Color 0xcd5c5cff
c: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (299.85047025528434, 8.65022464490034), Color 0x8b008bff
d: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (266.56474110460795, 14.407892500126207), Color 0xfffacdff
e: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (0.0, 44.27106567502161), Color 0xba55d3ff
f: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (324.7652535484209, 7.884340586600842), Color 0xf5deb3ff
g: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (316.86830276425, 7.896950784170906), Color 0x696969ff
h: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (206.9500320921286, 23.65339264796635), Color 0xdda0ddff
i: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (158.05473279053163, 24.732677204698334), Color 0x008080ff
j: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.93221297634653, 0.43653536882252286), Color 0xff00ffff
k: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (355.294170977383, 3.0516678119555802), Color 0x00ff00ff
l: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (250.48303032113762, 16.081710783470342), Color 0xcd5c5cff
m: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (291.06524231875153, 8.785227936532793), Color 0x00008bff
n: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (133.28311232863163, 24.771620461900003), Color 0x00bfffff
o: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (107.01718345467901, 26.265928873952628), Color 0x00ffffff
p: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (332.64959413502174, 6.629996816543261), Color 0xcd853fff
q: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.34583878933853, 0.5863741870079937), Color 0x0000ffff
r: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (230.60342474009494, 19.87960558104267), Color 0xffefd5ff
s: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (182.78740999522995, 24.162622096898655), Color 0x6a5acdff
t: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (44.27106567502161, 33.358794118950016), Color 0xf0f8ffff
u: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (280.97263360473414, 10.09260871401741), Color 0xcd5c5cff
v: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (352.0574299716785, 3.2367410057044665), Color 0xdcdcdcff
w: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (308.50069490018467, 8.367607864065368), Color 0xd8bfd8ff
x: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.3687483451691, 0.3949958944740755), Color 0xf5f5dcff
y: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (339.279590951565, 6.395595497005594), Color 0xcd5c5cff
z: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.76374423964313, 0.23625576035679444), Color 0xe9967aff

Sum Of Angles: 360.0
```

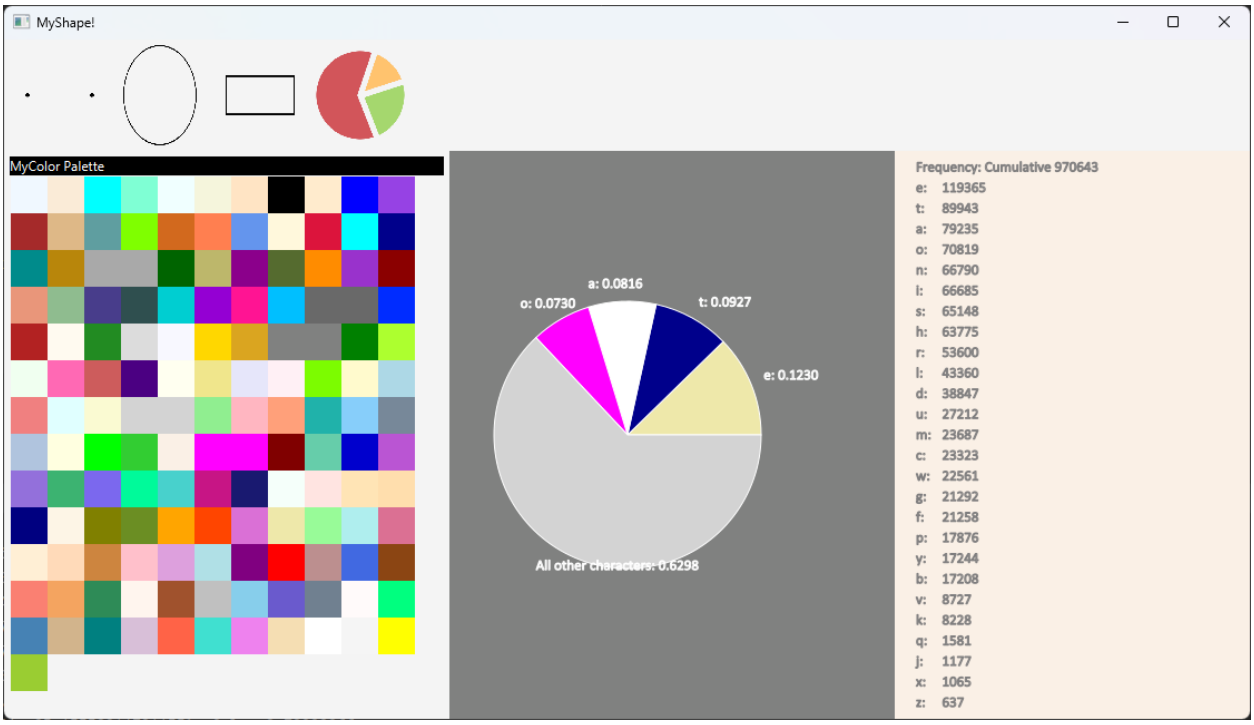




Run: MyShapeApplication

```
Pie Chart
a: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (77.62985979397163, 29.387323660707388), Color 0x006400ff
b: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (345.6751864485706, 6.382243523107879), Color 0x32cd32ff
c: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (299.85047025528434, 8.65022464490034), Color 0x87cefaff
d: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (266.56474110460795, 14.407892500126207), Color 0xff6347ff
e: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (0.0, 44.27106567502161), Color 0x8b4513ff
f: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (324.7652535484209, 7.884340586600842), Color 0x9370dbff
g: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (316.86830276425, 7.896950784170906), Color 0xee82eeff
h: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (206.9500320921286, 23.65339264796635), Color 0x708090ff
i: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (158.05473279053163, 24.732677204698334), Color 0xffa07aff
j: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.93221297634653, 0.43653536882252286), Color 0x00bfffff
k: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (355.294170977383, 3.0516678119555802), Color 0xffa07aff
l: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (250.48303032113762, 16.081710783470342), Color 0xda70d6ff
m: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (291.06524231875153, 8.785227936532793), Color 0xff8c00ff
n: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (133.28311232863163, 24.771620461900003), Color 0x4b0082ff
o: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (107.01718345467901, 26.265928873952628), Color 0x48d1ccff
p: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (332.64959413502174, 6.629996816543261), Color 0xfa8072ff
q: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.34583878933853, 0.5863741870079937), Color 0xffb6c1ff
r: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (230.60342474009494, 19.87960558104267), Color 0x00ff7fff
s: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (182.78740999522995, 24.162622096898655), Color 0xadd8e6ff
t: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (44.27106567502161, 33.358794118950016), Color 0x00008bff
u: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (280.97263360473414, 10.09260871401741), Color 0xba55d3ff
v: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (352.0574299716785, 3.2367410057044665), Color 0x00fa9aff
w: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (308.50069490018467, 8.367607864065368), Color 0xda70d6ff
x: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.3687483451691, 0.3949958944740755), Color 0x9400d3ff
y: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (339.279590951565, 6.395595497005594), Color 0xa9a9a9ff
z: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.76374423964313, 0.23625576035679444), Color 0x696969ff

Sum Of Angles: 360.0
```


Pie Chart
Display
5
Total
26
Starting Angle
0
Title
Moby Dick
Colors
Palette
OK
Cancel


MyShape!
MyColor Palette
a: 0.0816
o: 0.0730
t: 0.0927
e: 0.1230
All other characters: 0.6298
Frequency: Cumulative 970643
e: 119365
t: 89943
a: 79235
o: 70819
n: 66790
i: 66685
s: 65148
h: 63775
r: 53600
l: 43360
d: 38847
u: 27212
m: 23687
c: 23323
w: 22561
g: 21292
f: 21258
p: 17876
y: 17244
b: 17208
v: 8727
k: 8228
q: 1581
j: 1177
x: 1065
z: 637

Run: MyShapeApplication

```
Pie Chart
a: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (77.62985979397163, 29.387323660707388), Color 0xffffffff
b: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (345.6751864485706, 6.382243523107879), Color 0xbdb76bff
c: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (299.85047025528434, 8.650224664490034), Color 0x708090ff
d: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (266.56474110460795, 14.407892500126207), Color 0xf0e68cff
e: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (0.0, 44.27106567502161), Color 0xeee8aaff
f: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (324.7652535484209, 7.884340586600842), Color 0xff7f50ff
g: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (316.86830276425, 7.896950784170906), Color 0xa52a2aff
h: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (206.9500320921286, 23.653392647966635), Color 0xffd700ff
i: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (158.05473279053163, 24.732677204698334), Color 0x6a5acdff
j: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.93221297634653, 0.43653536882252286), Color 0x8b0000ff
k: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (355.294170977383, 3.0516678119555802), Color 0x8a2be2e1
l: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (250.48303032113762, 16.081710783470342), Color 0xfff0f5ff
m: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (291.06524231875153, 8.785227936532793), Color 0xffb6c1ff
n: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (133.28311232863163, 24.771620461900003), Color 0xff69b4ff
o: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (107.01718345467901, 26.265928873952628), Color 0xff00ffff
p: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (332.64959413502174, 6.629996816543261), Color 0x9370dbff
q: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (358.34583878933853, 0.5863741870079937), Color 0xd2b48cff
r: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (230.60342474009494, 19.87960558104267), Color 0xffe4b5ff
s: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (182.78740999522995, 24.162622096898655), Color 0x00008bff
t: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (44.27106567502161, 33.358794118950016), Color 0x00008bff
u: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (280.97263360473414, 10.09260871401741), Color 0xeee8aaff
v: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (352.0574299716785, 3.2367410057044665), Color 0xffff00ff
w: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (308.50069490018467, 8.367607864065368), Color 0xa0522dff
x: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.3687483451691, 0.3949958944740755), Color 0xfffaf0ff
y: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (339.279590951565, 6.395595497005594), Color 0xffdeadff
z: Slice: Center (160.0, 255.0) Width 240.0 Height 240.0 (Starting Angle, Angle): (359.76374423964313, 0.23625576035679444), Color 0x9acd32ff

Sum Of Angles: 360.0
```