The City College of New York

CSC 221 – P || Professor Hesham Auda

Exercise 3

Student: Binyamin Radensky

Contents

[Introduction 2](#_Toc24629082)

[Main Class 3](#_Toc24629083)

[PieChart Class 6](#_Toc24629084)

[ReadTextFile class 11](#_Toc24629085)

[HistogramAlphaBet class 12](#_Toc24629086)

[Enum MyColor 13](#_Toc24629087)

[Conclusion 16](#_Toc24629088)

[MAIN 16](#_Toc24629089)

[PieChart 17](#_Toc24629090)

[ReadTextFile 21](#_Toc24629091)

[HistogramAlphaBet 22](#_Toc24629092)

[MyColor 23](#_Toc24629093)

# Introduction

In this assignment our goal was to make a pie chart of a histogram with the probabilities of each letter appearing in the text file ‘Emma’. A pie chart is a rather simple class to program since it is just comprised of various arcs that you can create by iterating through an array with the values you need using JavaFX. We could choose to make this part of the MyShape hierarchy since the only methods it is missing the getMyBoundingBox and the toString method. Each of these methods would be easy to implement, however, our pie chart did not need to use any classes from the previous assignments besides for the main class. The pie chart that we had to implement had more facets than a normal pie chart. First, we need a method to add events and their probabilities (or sizes in a more general view). The twist is that we also need a method that can also take a map data structure, that has frequencies of the events instead of their probabilities and put that int our pie chart. Next, we must make the histogram class which will take the string from the read file class and put it into the map as usable keys and frequency values which will be our histogram. Our histogram class will pass this map into the special method we made in our pie chart class which will give us the result, the pie chart.

# Main Class

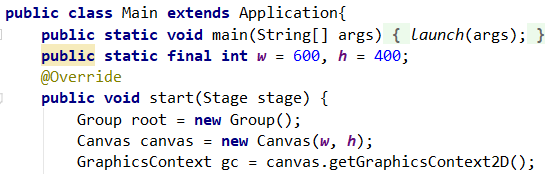
The first thing we did was modify our main method from the last project. It begins by extending the application class, launching our start method, and creating variables for the width and height of our window.

Figure : main header and start method

Our start method is completely the same as the previous project. We create a group called root which will hold all of the canvas elements. Then we create a new canvas with the width and height we specified. Then we instantiate our two dimensional graphics context object which we will use to draw all of our shapes.

Below are all the methods we imported for our main class. As well as the command declaring the main class part of the package project one, even though we are in project 2.

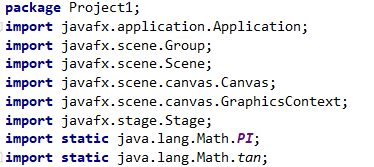


Figure : imported methods for main class

The snippet below adds the canvas to the root group, and then sets the scene with the root and then shows the stage.



Figure : final snippet of code from main class start method

After declaring the graphics context objects and the other object, we declared the PieChart object. For this project we chose the width and height to be half of the size of the canvas. These values which are passed in are used to draw the width and heights of the arcs that make up the pi chart.

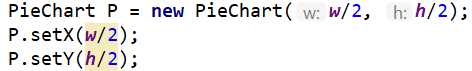


Figure : pie chart declaration and setting of middle point on canvas

We then have some test code that we used to test and make sure that our PieChart class was working properly. We named the different entries by the order in which they were added to the pie chart, that way we could easily observe if there was an error due to the order in which the entries were added.

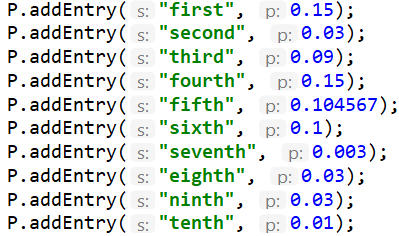


Figure : test code in main for pie chart

Lastly, for the new part of the main class, we declared the HistogramAlphaBet object and used it to create our set of probabilities and draw them. After instantiating the object, H, we called the count function which was designed to clean the input and count the occurrences of each letter in whatever file is put into it. Count will also call the PieChart class and load all the values it generated into the PieChart object that was passed into the count function. Next, we set the number of probabilities we wanted to be displayed. The pie chart is designed so that the highest probabilities will be displayed first. Lastly, we drew the PieChart object onto the canvas, which has all the probabilities as area of the circle as well as text which says what they represent and how large of a probability they have.

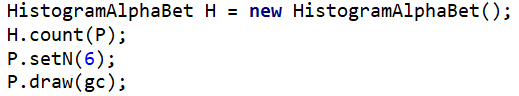


Figure : instantiation of Histogram and method calls to histogram and pie chart

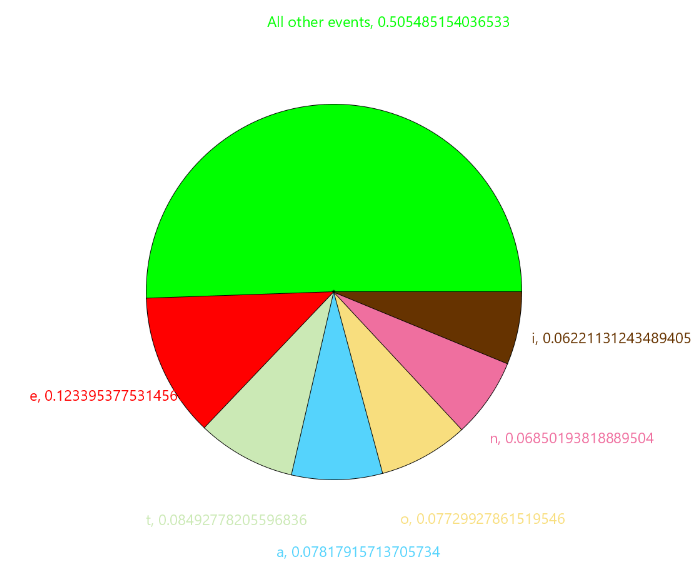


Figure : n = 6 our histogram object with the six largest probability events

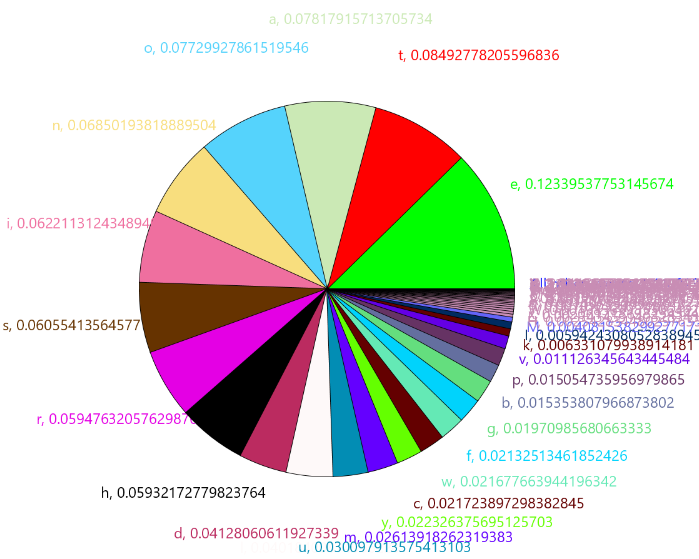


Figure : Our pie chart with all of the events from the emma file in one pie chart

# PieChart Class

The PieChart class will take either inputs of a string and its probability one at a time or a map of characters and values that represent the frequency of those values. We can then set how many probabilities, “n”, we should display. Lastly there is a draw method which allows us to draw the pie chart onto the canvas. For the constructor we just set the local variables and initialized n to zero. Before the constructor we also created dynamic arrays, called array lists to store our string and probability values.

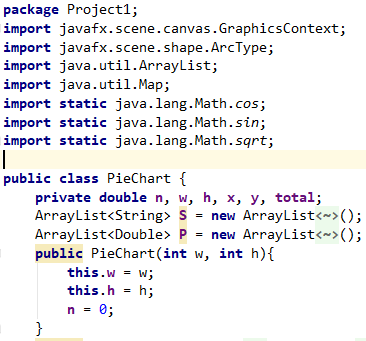


Figure : header, imports, variable declarations, and constructor for the pie chart class

Our next set of methods allow us to set and get the values of x, y and n which are the center of the pie chart and the number of events shown on the chart when drawn. We also included a way of preventing the user from trying to set an n that is larger than the number of entries available.

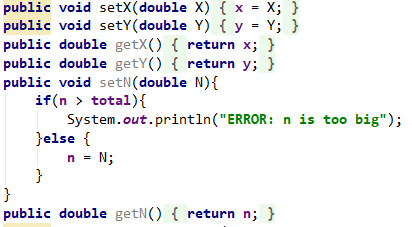


Figure : set and get methods for the pie chart class

Our next method takes in a map as a parameter that is fill with keys and frequencies and inputs them into our addEntry method using a loop. The first thing the method does is add all the frequencies together to get the total frequencies of the events in the map. Then it uses a loop to add the entries and it divides the frequency of each event by the total frequency to get the probability of that event. We also had to convert the character to a string since our PieChart class only uses strings.

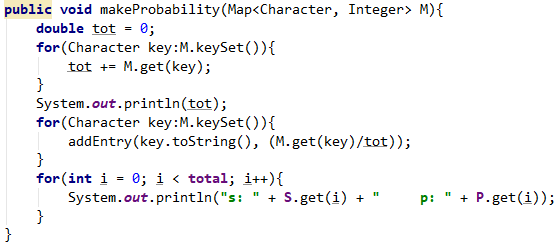


Figure : make probability method for pie chart class, it takes a map and inserts it into the pie chart

Our addEntry method goes from the end of our arrays to the beginning and arranges them from smallest, in the front of the arrays, to largest probabilities. It sorts by searching through the array until it finds an entry in the array hat is smaller than the entry we want to put in, and then we put our entry after that entry. If the arrays are empty, then they will be added to the back of the array. If the entry is smaller than the first entry of the array, we have made an if statement that will check for that and put it in the first spot if it is true. We also increment N and so that we can add events without having to keep track of how many we added. This could pose a potential problem if someone set n then added events to the PieChart since every event they added since setting n will also increment n.

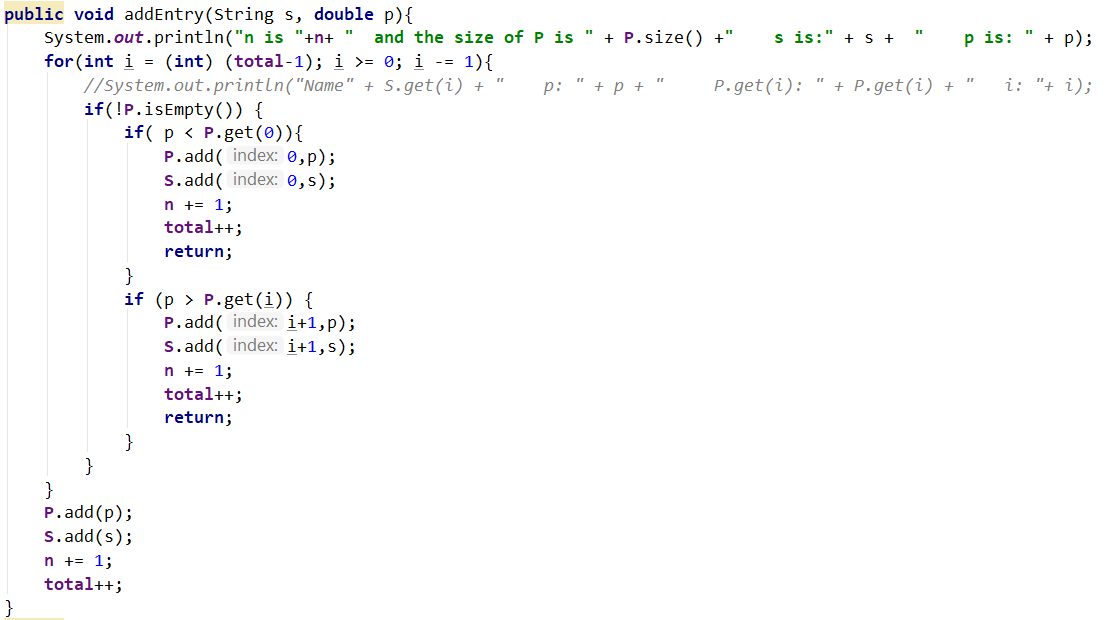


Figure : method to add entries to the pie chart

Our set color method iterates through the different colors of the enum by using a counter that is passed through the method and counts to the next color that hasn’t been used yet. We also have the get color method which returns the color we are using.

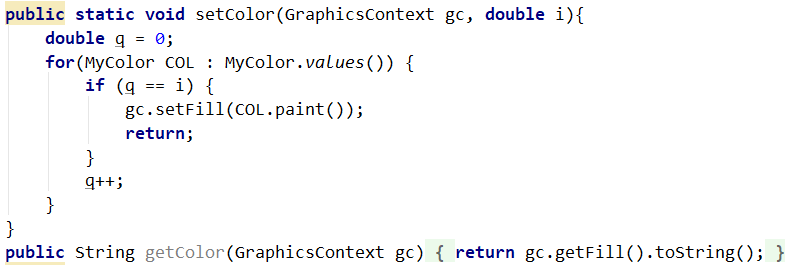


Figure : color iterator for draw method and get color method

Lastly, we have our draw method which draws our pie chart onto the canvas. In order to draw the arcs using graphics context we need seven parameters, the (x, y) coordinates of the arcs, the width and height of the arcs, the starting angle, the angle that of the arc, and the closing type of the arc. The angles gc uses need to be in degrees for this object, hence probability multiplied by 360. We also have the first for loop which adds together the probabilities of the other angles being used and then takes the 1 – value of that to find the value of the “other segment” which is then added to the back of the array so that it can be included in the draw. We also increment total and n so that it is displayed and we de-increment and delete afterwards so that if we add more elements after drawing there are no issues. In the main for loop we start by calculating the angle of each arc, then we get the color set for that arc, then we get down to drawing the arc. We finish up the for loop by also putting down the text onto the canvas and adding the angle of the arc to the position angle so that the next arc can start where the last one finished. We were able to calculate where the text goes by the offset to the middle (adding x and y) but adding the value of the radius multiplied by the cos of the angle to get the x coordinate of the point. To get the y coordinate of the point we di the same thing except with sin and we subtracted instead of added due to the orientation of the angle in relation to the x and y axis, where in our canvas it is essentially flipped since the origin (x =0, y =0) is at the top left corner and going down is positive (flip of y axis).

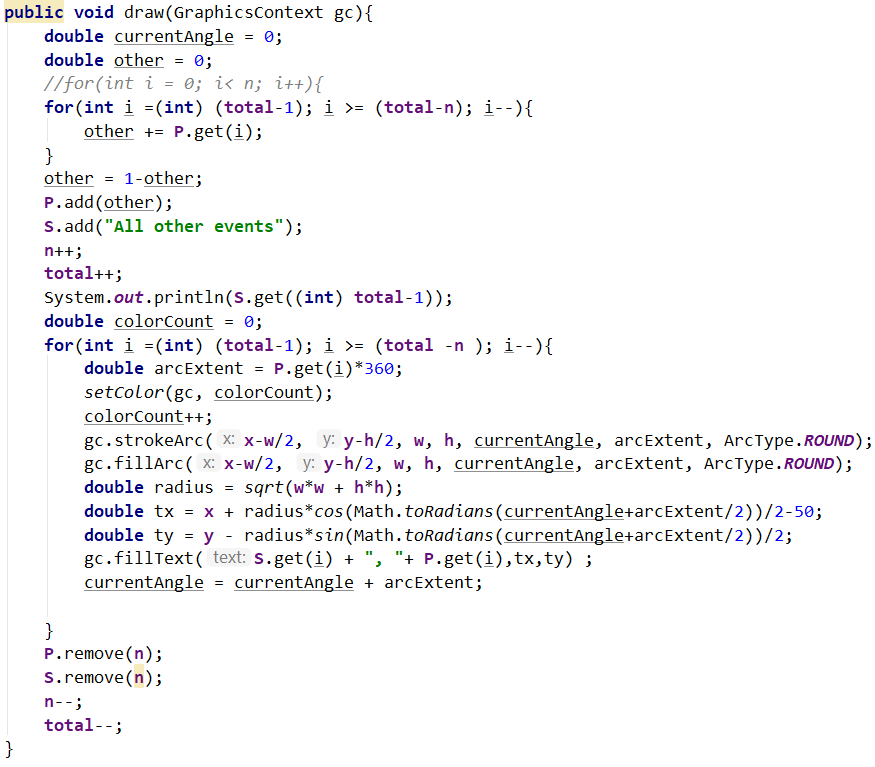


Figure : draw method for pie chart

# ReadTextFile Class

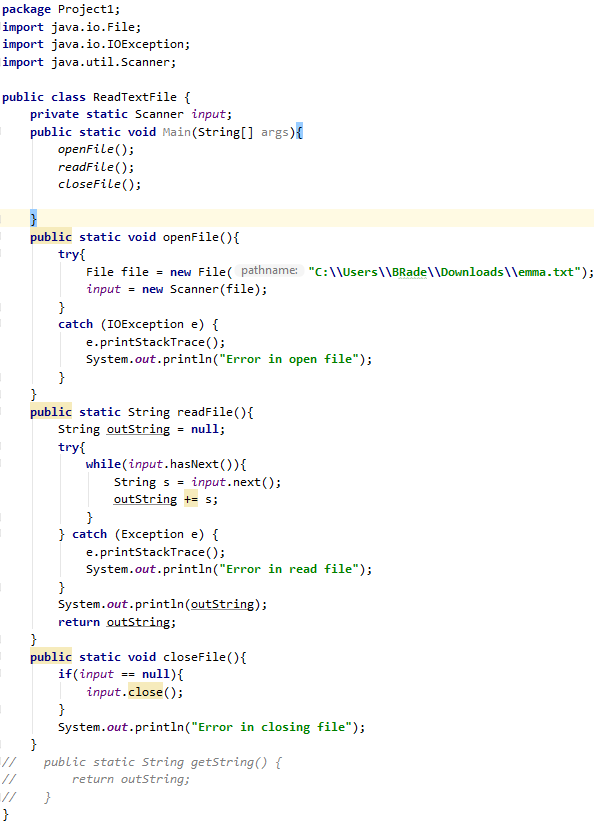
 Our file stars with our imports and other things. Then we declare our scanner which allows use to take a stream or input and do something with it.

Figure : the entire read text file class

In our main method we call all 3 methods consecutively to get the file.

Next, we have our actual implementations. Our openFile method will try to instantiate a connection to the file and then sends it to the scanner. If something goes wrong it will be handled by exception catcher.

Our readFile method adds all the data in the file into a string which is then returned by this method.

Lastly, we have the closeFile method which will attempt to close the file if the input is null.

# HistogramAlphaBet Class

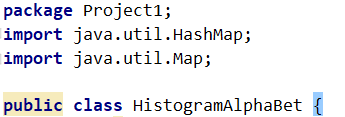
  
 On our right we have the header and the import files as well as the package declaration for our HistogramAlphaBet class.

Figure : first part of our histogram class

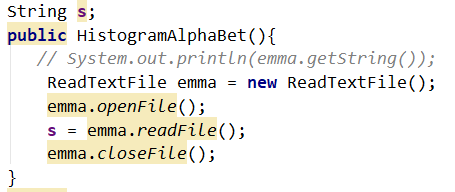
 The first thing we do is declare our string which will store the string passed into it by the ReadTextFile class. Next, we have our first method which is the constructor for the class. We call the ReadTextFile class and set the string s, which is a local variable of class, equal to the output of the readFile method from the ReadTextFile class.

Figure : constructor for histogram class. Instantiates text file object to read the text file and make a histogram out of it.

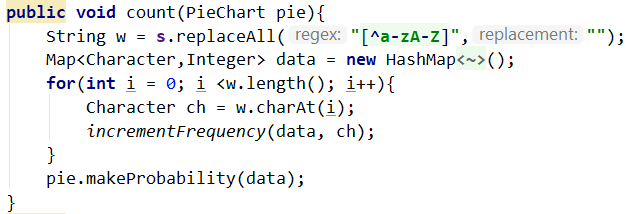
 The count method starts by taking the string and cleaning it using regex. What our regex says is basically if you are not either a lowercase or uppercase alphabet character, then you will be replaced with nothing. After cleaning our string, we instantiate the map that we are going to use to store our values. We use a map because it is designed to store pairs of data with one of them being the key, which perfect when we have the letter which would be the key and the value associated with it, which int his case is the frequency of that letter in the text file. After, we have a for loop that goes through every letter in the string and sends it to our increment function which uses the characters and our map to store the values and increment their entries in our map. Lastly, we call the PieChart object, that was passed into our class method by the main class, and we use it to call the makeProbability method that will make a pie chart out of the values that we have gotten in our map. A nice feature of the makeProbability method is that we can pass the map into the method.

Figure : count method cleans the string containing the text from the file and then goes through the text and calls the incrementor for each character that appears in the string from the text file

Lastly, we have our increment frequency method. Here we use a generic method which has its own type parameters and allows use to choose what type we are going to use with its parameters. In our method we use the k generic type so specify the type for our key. This means that now if we wanted to reuse this method somewhere else, we could use it with a different type (other than Character). The first thing our method does is that if the probability is absent put a zero there, then we increment the value associated with the key that was passed into our method. We then return (it is void so there is no return value) and then it is called again the next time there is match for one of the keys in the map.

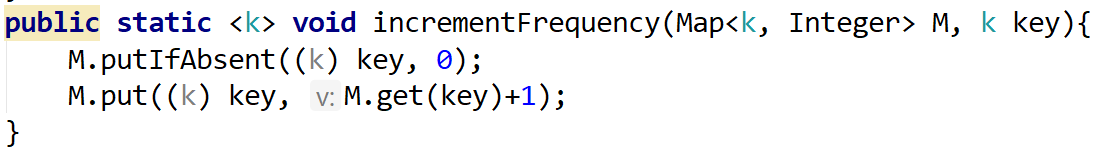


Figure : Increments the value in the map for the key specified

# Enum MyColor

MyColor is an enum type that is used to define, mix, and return colors. It is part of the package Project1 and uses javafx.scene.paint.Color to return usable colors for use with graphics context objects. The header of an enum type has every different option of that type and the values associated with it. That way if we call mycolor.BLUE then the values for blue will be returned. There were several different ways of implementing this. We could have had the associated values for each enum type to be the paint.color value of that color. However, in order to make it easier to return colors with different functions, we made the values integer values of red, green, and blue.



Figure : First part of MyColor enum

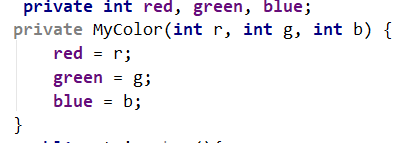
 The constructor for the enum type can take red, green, and blue integer values and use them as a custom color. We can then use different methods that will make the custom color that was passed in usable for graphics context objects.

Figure : Constructor for MyColor enum

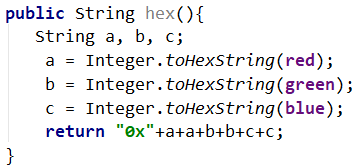
 Our first method returns a string with the hex number of the enum color chosen. It is designed to mimic the way that javafx.scene.paint.Color returns chosen colors. The output of brown, red = 102; green = 51; blue = 0, can be seen below. 

Figure : hex method

Figure : output of hex method

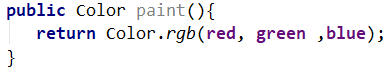
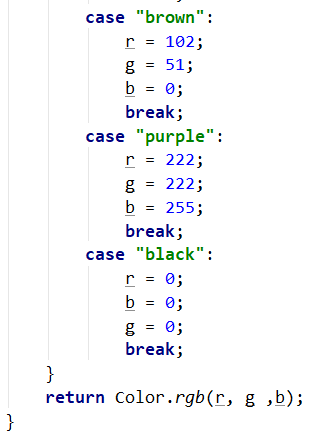
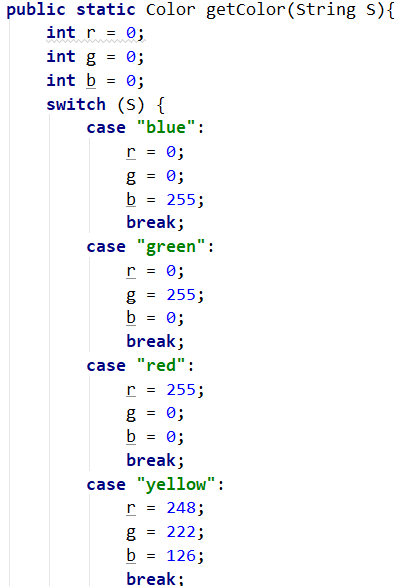
 The method paint uses the enum value and will return the JavaFX color type associated with that value. That way we could call mycolor.BLUE.paint() and It would be like calling color.BLUE. This is the most important method of the enum type as it is what we use in order to paint the canvas and return color types that graphics context can use.

Figure : paint method, most important for our use

 The method getColor acts as if it is a method in a regular class and returns color type values that are usable by graphics context. This method was created as a redundancy however, it does simplify passing colors from the Main class through MyShape to MyColor, since now you can write the string name of the desired color, and that string can be sent directly to the MyColor method.

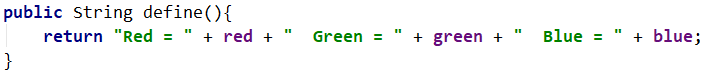
The define method returns a string with the red, green, and blue values of the enum selected color. The code can be seen below and output for the color BROWN is below the code. 

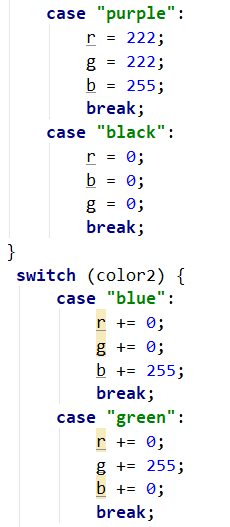
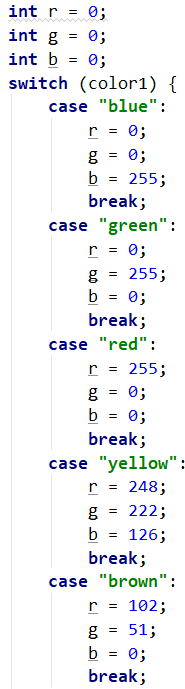
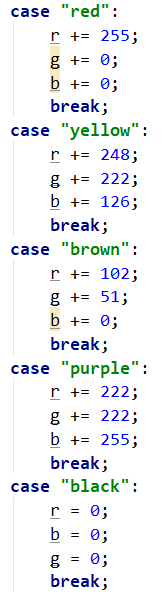
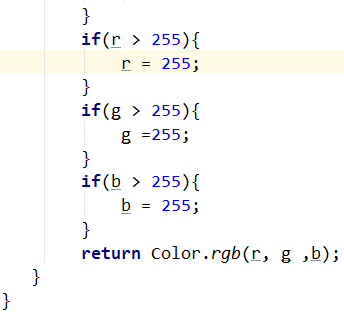
Figure : define method



Figure : Output of define method

Lastly, the mix method will take to colors, specified by the name in a string value, and add their red, green, and blue values together. If any value goes above 255 it will be set back to 255.



# 

# Conclusion

This project was extremely interesting for many reasons. This was the first time we used regex in a project, and here we used it to clean our text file of any unwanted characters. Creating the histogram was mostly straightforward, implementing the code taught to us and understanding how it works and why it is made the way it is. For example, our incrementFrequency method in our read text file class was made so that you could use any type you want for the keys in the map. It happens to be for the map we made we used the Character type, but we could have used a different type and that method would still work. Another interesting part of this project was opening files which we had never done before. Although this was not challenging, due to the considerable amount of time we spent in class learning about how to do this, it was still interesting to see it work and understand why you need each part to make it all work.

We started this project by making the pie chart class. We chose not to make it part of our MyShape hierarchy since it was not necessary, and we had all the functionality we needed to complete this assignment without doing so. One thing that took some time with the pie chart class was getting the coordinates for the text in the right place. In order to fix this, we had to recall that our y axis was flipped so that as you went down the numbers got more positive. However, the x axis was consistent with a usual x axis.

Overall this project was extremely interesting and really taught us how to use files, regex, maps, and many other useful tools from Java. The regex was light, easy, and intuitive, allowing us to clean our file. The map allowed us to easily store the events and their frequencies that we got from the file. We learned how to then how to use these tools in relation to other objects and data structures that we already know, such as adding the contents of a map to an array or a set. This project taught us many useful tools for Java that are critical to creating useful graphics-based objects.

## MAIN

**package** Project1;  
**import** javafx.application.Application;  
**import** javafx.scene.Group;  
**import** javafx.scene.Scene;  
**import** javafx.scene.canvas.Canvas;  
**import** javafx.scene.canvas.GraphicsContext;  
**import** javafx.stage.Stage;  
**import static** java.lang.Math.***PI***;  
**import static** java.lang.Math.*tan*;  
  
  
**public class** Main **extends** Application{  
 **public static void** main(String[] args){  
 *launch*(args);  
 }  
 **public static final int *w*** = 600, ***h*** = 600;  
 @Override  
 **public void** start(Stage stage) {  
 Group root = **new** Group();  
 Canvas canvas = **new** Canvas(***w***, ***h***);  
 GraphicsContext gc = canvas.getGraphicsContext2D();  
 PieChart P = **new** PieChart(***w***/2, ***h***/2);  
 P.setX(***w***/2);  
 P.setY(***h***/2);  
*// P.addEntry("first", 0.15);  
// P.addEntry("second", 0.03);  
// P.addEntry("third", 0.09);  
// P.addEntry("fourth", 0.15);  
// P.addEntry("fifth", 0.104567);  
// P.addEntry("sixth", 0.1);  
// P.addEntry("seventh", 0.003);  
// P.addEntry("eighth", 0.03);  
// P.addEntry("ninth", 0.03);  
// P.addEntry("tenth", 0.01);* HistogramAlphaBet H = **new** HistogramAlphaBet();  
 H.count(P);  
*// P.setN(6);* P.draw(gc);  
  
 *//display canvas / scene / stage* root.getChildren().add(canvas); *//adds the canvas to the root group* stage.setScene(**new** Scene(root));*//sets the scene on the stage that is using the root with the canvas elements* stage.show();*// tells java to show the stage in the application with all of the things that we put in it* }  
 }

## PieChart

**package** Project1;  
**import** javafx.scene.canvas.GraphicsContext;  
**import** javafx.scene.shape.ArcType;  
**import** java.util.ArrayList;  
**import** java.util.Map;  
**import static** java.lang.Math.*cos*;  
**import static** java.lang.Math.*sin*;  
**import static** java.lang.Math.*sqrt*;  
  
**public class** PieChart {  
 **private double n**, **w**, **h**, **x**, **y**, **total**;  
 ArrayList<String> **S** = **new** ArrayList<String>();  
 ArrayList<Double> **P** = **new** ArrayList<Double>();  
 **public** PieChart(**int** w, **int** h){  
 **this**.**w** = w;  
 **this**.**h** = h;  
 **n** = 0;  
 }  
 **public void** setX(**double** X){  
 **x** = X;  
 }  
 **public void** setY(**double** Y){  
 **y** = Y;  
 }  
 **public double** getX(){  
 **return x**;  
 }  
 **public double** getY(){  
 **return y**;  
 }  
 **public void** setN(**double** N){  
 **if**(**n** > **total**){  
 System.***out***.println(**"ERROR: n is too big"**);  
 }**else** {  
 **n** = N;  
 }  
 }  
 **public double** getN(){  
 **return n**;  
 }  
 **public void** makeProbability(Map<Character, Integer> M){  
 **double** tot = 0;  
 **for**(Character key:M.keySet()){  
 tot += M.get(key);  
 }  
 System.***out***.println(tot);  
 **for**(Character key:M.keySet()){  
 addEntry(key.toString(), (M.get(key)/tot));  
 }  
 **for**(**int** i = 0; i < **total**; i++){  
 System.***out***.println(**"s: "** + **S**.get(i) + **" p: "** + **P**.get(i));  
 }  
 }  
 **public void** addEntry(String s, **double** p){  
 System.***out***.println(**"n is "**+**n**+ **" and the size of P is "** + **P**.size() +**" s is:"** + s + **" p is: "** + p);  
 **for**(**int** i = (**int**) (**total**-1); i >= 0; i -= 1){  
 *//System.out.println("Name" + S.get(i) + " p: " + p + " P.get(i): " + P.get(i) + " i: "+ i);* **if**(!**P**.isEmpty()) {  
 **if**( p < **P**.get(0)){  
 **P**.add(0,p);  
 **S**.add(0,s);  
 **n** += 1;  
 **total**++;  
 **return**;  
 }  
 **if** (p > **P**.get(i)) {  
 **P**.add(i+1,p);  
 **S**.add(i+1,s);  
 **n** += 1;  
 **total**++;  
 **return**;  
 }  
 }  
 }  
 **P**.add(p);  
 **S**.add(s);  
 **n** += 1;  
 **total**++;  
 }  
 **public static void** setColor(GraphicsContext gc, **double** i){  
 **double** q = 0;  
 **for**(MyColor COL : MyColor.*values*()) {  
 **if** (q == i) {  
 gc.setFill(COL.paint());  
 **return**;  
 }  
 q++;  
 }  
 }  
 **public** String getColor(GraphicsContext gc){  
 **return** gc.getFill().toString();  
 }  
 **public void** draw(GraphicsContext gc){  
 **double** currentAngle = 0;  
 **double** other = 0;  
 *//for(int i = 0; i< n; i++){* **for**(**int** i =(**int**) (**total**-1); i >= (**total**-**n**); i--){  
 other += **P**.get(i);  
 }  
 other = 1-other;  
 **P**.add(other);  
 **S**.add(**"All other events"**);  
 **n**++;  
 **total**++;  
 System.***out***.println(**S**.get((**int**) **total**-1));  
 **double** colorCount = 0;  
 **for**(**int** i =(**int**) (**total**-1); i >= (**total** -**n** ); i--){  
 **double** arcExtent = **P**.get(i)\*360;  
 *setColor*(gc, colorCount);  
 colorCount++;  
 gc.strokeArc(**x**-**w**/2, **y**-**h**/2, **w**, **h**, currentAngle, arcExtent, ArcType.***ROUND***);  
 gc.fillArc(**x**-**w**/2, **y**-**h**/2, **w**, **h**, currentAngle, arcExtent, ArcType.***ROUND***);  
 **double** radius = *sqrt*(**w**\***w** + **h**\***h**);  
 **double** tx = **x** + radius\**cos*(Math.*toRadians*(currentAngle+arcExtent/2))/2-50;  
 **double** ty = **y** - radius\**sin*(Math.*toRadians*(currentAngle+arcExtent/2))/2;  
 gc.fillText(**S**.get(i) + **", "**+ **P**.get(i),tx,ty) ;  
 currentAngle = currentAngle + arcExtent;  
  
 }  
 **P**.remove(**n**);  
 **S**.remove(**n**);  
 **n**--;  
 **total**--;  
 }  
}

## ReadTextFile



## HistogramAlphaBet

**package** Project1;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** HistogramAlphaBet {  
 String **s**;  
 **public** HistogramAlphaBet(){  
 *// System.out.println(emma.getString());* ReadTextFile emma = **new** ReadTextFile();  
 emma.*openFile*();  
 **s** = emma.*readFile*();  
 emma.*closeFile*();  
 }  
 **public void** count(PieChart pie){  
 String w = **s**.replaceAll(**"[^a-zA-Z]"**,**""**);  
 Map<Character,Integer> data = **new** HashMap<Character, Integer>();  
 **for**(**int** i = 0; i <w.length(); i++){  
 Character ch = w.charAt(i);  
 *incrementFrequency*(data, ch);  
 }  
 pie.makeProbability(data);  
 }  
 **public static** <k> **void** incrementFrequency(Map<k, Integer> M, k key){  
 M.putIfAbsent((k) key, 0);  
 M.put((k) key, M.get(key)+1);  
 }  
}

## MyColor

**package** Project1;  
**import** javafx.scene.paint.Color;  
  
**public enum** MyColor{  
 ***BLUE***(0, 0 , 255), ***GREEN***(0, 255, 0), ***RED***(255, 0, 0),  
 ***HUDSON***(203,233, 181),***SKY***(85, 211, 252), ***YELLOW***(248, 222, 126), ***PINK***(239, 111, 159),  
 ***BROWN***(102, 51, 0), ***PUPLRE***(228, 0, 228), ***BLACK***(0,0,0), ***BERGUNDY***(187,43,96),  
 ***NYCSNOW***(254 ,249 ,249), ***AQUA***(2 ,141 ,180),***BLU***(100, 0 , 255), ***GREE***(100, 255, 0), ***RE***(100, 0, 0),  
 ***HUDSO***(100,233, 181), ***SK***(0, 211, 252), ***YELLO***(100, 222, 126), ***PIN***(100, 111, 159),  
 ***BROW***(102, 51, 100), ***PUPLR***(100, 0, 228), ***BLAC***(100,0,0), ***BERGUND***(0,43,96),  
 ***NYCSNO***(100 ,100 ,249), ***AQU***(200 ,141 ,180);  
 **private int red**, **green**, **blue**;  
 **private** MyColor(**int** r, **int** g, **int** b) {  
 **red** = r;  
 **green** = g;  
 **blue** = b;  
 }  
 **public** String hex(){  
 String a, b, c;  
 a = Integer.*toHexString*(**red**);  
 b = Integer.*toHexString*(**green**);  
 c = Integer.*toHexString*(**blue**);  
 **return "0x"**+a+a+b+b+c+c;  
 }  
 **public** Color paint() {  
 **return** Color.*rgb*(**red**, **green**, **blue**);  
 }  
 **public static** Color getColor(String S){  
 **int** r = 0;  
 **int** g = 0;  
 **int** b = 0;  
 **switch** (S) {  
 **case "blue"**:  
 r = 0;  
 g = 0;  
 b = 255;  
 **break**;  
 **case "green"**:  
 r = 0;  
 g = 255;  
 b = 0;  
 **break**;  
 **case "red"**:  
 r = 255;  
 g = 0;  
 b = 0;  
 **break**;  
 **case "yellow"**:  
 r = 248;  
 g = 222;  
 b = 126;  
 **break**;  
 **case "brown"**:  
 r = 102;  
 g = 51;  
 b = 0;  
 **break**;  
 **case "purple"**:  
 r = 222;  
 g = 222;  
 b = 255;  
 **break**;  
 **case "black"**:  
 r = 0;  
 b = 0;  
 g = 0;  
 **break**;  
 }  
 **return** Color.*rgb*(r, g ,b);  
 }  
 **public** String define(){  
 **return "Red = "** + **red** + **" Green = "** + **green** + **" Blue = "** + **blue**;  
 }  
 **public static** Color mix(String color1, String color2){  
 **int** r = 0;  
 **int** g = 0;  
 **int** b = 0;  
 **switch** (color1) {  
 **case "blue"**:  
 r = 0;  
 g = 0;  
 b = 255;  
 **break**;  
 **case "green"**:  
 r = 0;  
 g = 255;  
 b = 0;  
 **break**;  
 **case "red"**:  
 r = 255;  
 g = 0;  
 b = 0;  
 **break**;  
 **case "yellow"**:  
 r = 248;  
 g = 222;  
 b = 126;  
 **break**;  
 **case "brown"**:  
 r = 102;  
 g = 51;  
 b = 0;  
 **break**;  
 **case "purple"**:  
 r = 222;  
 g = 222;  
 b = 255;  
 **break**;  
 **case "black"**:  
 r = 0;  
 b = 0;  
 g = 0;  
 **break**;  
 }  
 **switch** (color2) {  
 **case "blue"**:  
 r += 0;  
 g += 0;  
 b += 255;  
 **break**;  
 **case "green"**:  
 r += 0;  
 g += 255;  
 b += 0;  
 **break**;  
 **case "red"**:  
 r += 255;  
 g += 0;  
 b += 0;  
 **break**;  
 **case "yellow"**:  
 r += 248;  
 g += 222;  
 b += 126;  
 **break**;  
 **case "brown"**:  
 r += 102;  
 g += 51;  
 b += 0;  
 **break**;  
 **case "purple"**:  
 r += 222;  
 g += 222;  
 b += 255;  
 **break**;  
 **case "black"**:  
 r = 0;  
 b = 0;  
 g = 0;  
 **break**;  
 }  
 **if**(r > 255){  
 r = 255;  
 }  
 **if**(g > 255){  
 g =255;  
 }  
 **if**(b > 255){  
 b = 255;  
 }  
 **return** Color.*rgb*(r, g ,b);  
 }  
}