Project Report (Group 20)

Outline of Design

Our program is designed to filter data and display it in multiple different ways. The first screen shows all the data in its raw form, straight from the excel file. The user can select which individual piece of the data to display by clicking one of the buttons to the right. Data can be further narrowed by using the dropdowns and entering values to filter by. Each time the user pressed the enter key, the current array of resulting data is filtered further. Pressing the reset button sets the result array back to the full data set. The buttons on the bottom allow the user to select how to display the filtered data - raw form, bar chart, or pie chart. The heat map button brings up a separate screen that then prompts the user to choose which data set they want to view a heat map for.

User Interface and Widgets

We created many widgets and methods in order to create a clean and easy to understand UI. We decided to make the search bars at the right of the screen expandable and retractable, so that each query title is clearly visible. Each of these queries is created with an index that links back to the user's input, which in turn changes the display.

When initialised, the widget is given an index (last parameter) based on the queries index in the array.

```
dropList.add( new DropDown(dListX, dListY, "Flight Date", font, 60, 0));
dropList.add( new DropDown(dListX, dListY+50, "Origin", font, 60, 3));
dropList.add( new DropDown(dListX, dListY+100, "Destination", font, 60, 8));
dropList.add( new DropDown(dListX, dListY+150, "Flight Number", font, 60, 2));
dropList.add( new DropDown(dListX, dListY+200, "Departure Time", font, 60, 13));
dropList.add( new DropDown(dListX, dListY+250, "Arrival Time", font, 60, 15));
```

We also created a dynamic title that appears at the top of the screen that changes based on the user's input. This was added as it allows the user to remember their input, and also lets their selection be readable to others that may use the program.

The start of the title is based on the screen number.

```
case 1:
  onHeatmapScreen = false;
  pageTitle += "Bar Chart of Flight Information";
```

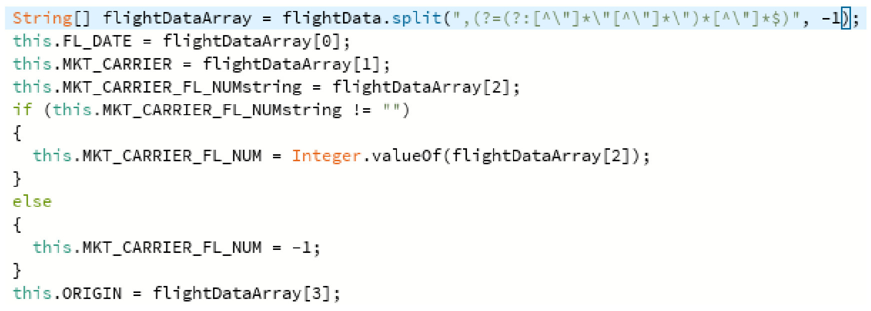
```
case 2:
  onHeatmapScreen = false;
  pageTitle += "Pie Chart of Flight Information";
```

Based on the User Input, the title changes.

```
if (!dropList.get(4).child.output.equals(""))
{
  pageTitle += ", with dep time: " + dropList.get(4).child.output;
}
if (!dropList.get(5).child.output.equals(""))
{
  pageTitle += ", with arrival time: " + dropList.get(5).child.output;
}
```

How we Organised Data

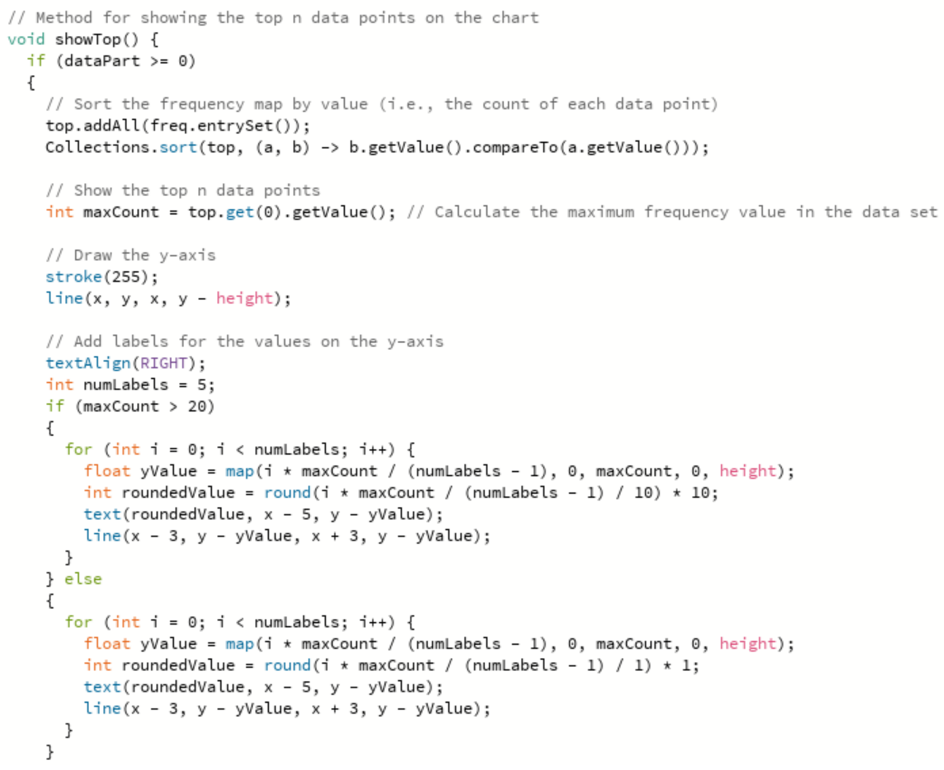
We organised the data by creating a data class that represents one line in the excel file. Each column is set as a separate variable, allowing easy access to specific pieces of data. The data class has methods getStrVal() and getIntVal() that return the associated values based on a String parameter that specifies the column requested. To sort data, we wrote a class called Query that can narrow the full data set into a smaller array list of data that meet certain criteria. There are multiple constructors to handle different types of query input, and the run() method contains a switch statement to differentiate between the query types before “running” the query and narrowing the data. It loops through each data object (each line in the excel file) in the full data list and compares the specific piece of data (the value at the user-specified column, “paramName”) to the user-specified search value (strValToCheck / intValToCheck). If the current value is equal to the search value, the data object is added to the ArrayList “result.



```
String[] flightDataArray = flightData.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);
this.FL_DATE = flightDataArray[0];
this.MKT_CARRIER = flightDataArray[1];
this.MKT_CARRIER_FL_NUMstring = flightDataArray[2];
if (this.MKT_CARRIER_FL_NUMstring != "")
{
  this.MKT_CARRIER_FL_NUM = Integer.valueOf(flightDataArray[2]);
}
else
{
  this.MKT_CARRIER_FL_NUM = -1;
}
this.ORIGIN = flightDataArray[3];
```

Bar Chart

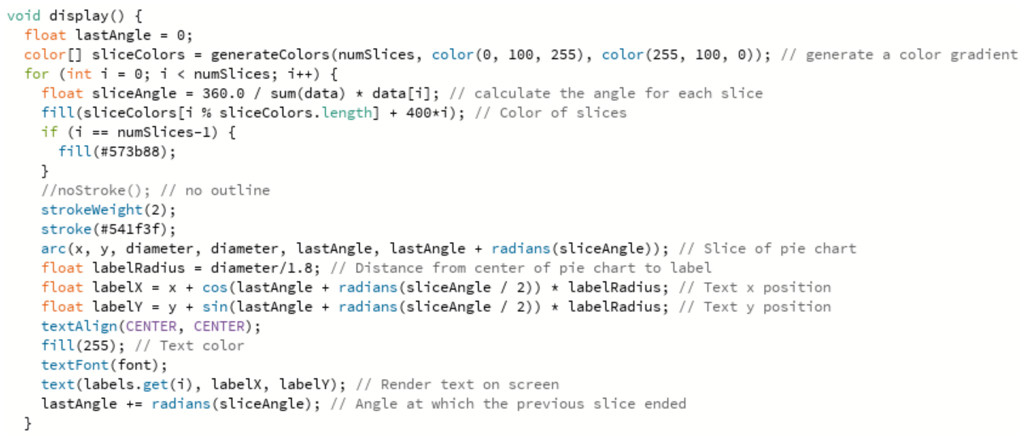
The BarChart class has a constructor that takes in parameters for the position, dimensions of the chart, font, and the number of data points to display on the chart. The addData() method is used to add data to the chart and showTop() is used to display the top (n) data points on the chart, which aids with data visualisation. The method sorts the frequency map by value and calculates the height of each bar based on the count of the data point. The BarChart class allows for sorting and visualisation of flight data.



```
// Method for showing the top n data points on the chart
void showTop() {
  if (dataPart >= 0)
  {
    // Sort the frequency map by value (i.e., the count of each data point)
    top.addAll(freq.entrySet());
    Collections.sort(top, (a, b) -> b.getValue().compareTo(a.getValue()));

    // Show the top n data points
    int maxCount = top.get(0).getValue(); // Calculate the maximum frequency value in the data set

    // Draw the y-axis
    stroke(255);
    line(x, y, x, y - height);

    // Add labels for the values on the y-axis
    textAlign(RIGHT);
    int numLabels = 5;
    if (maxCount > 20)
    {
      for (int i = 0; i < numLabels; i++) {
        float yValue = map(i * maxCount / (numLabels - 1), 0, maxCount, 0, height);
        int roundedValue = round(i * maxCount / (numLabels - 1) / 10) * 10;
        text(roundedValue, x - 5, y - yValue);
        line(x - 3, y - yValue, x + 3, y - yValue);
      }
    } else
    {
      for (int i = 0; i < numLabels; i++) {
        float yValue = map(i * maxCount / (numLabels - 1), 0, maxCount, 0, height);
        int roundedValue = round(i * maxCount / (numLabels - 1) / 1) * 1;
        text(roundedValue, x - 5, y - yValue);
        line(x - 3, y - yValue, x + 3, y - yValue);
      }
    }
```

Pie Chart

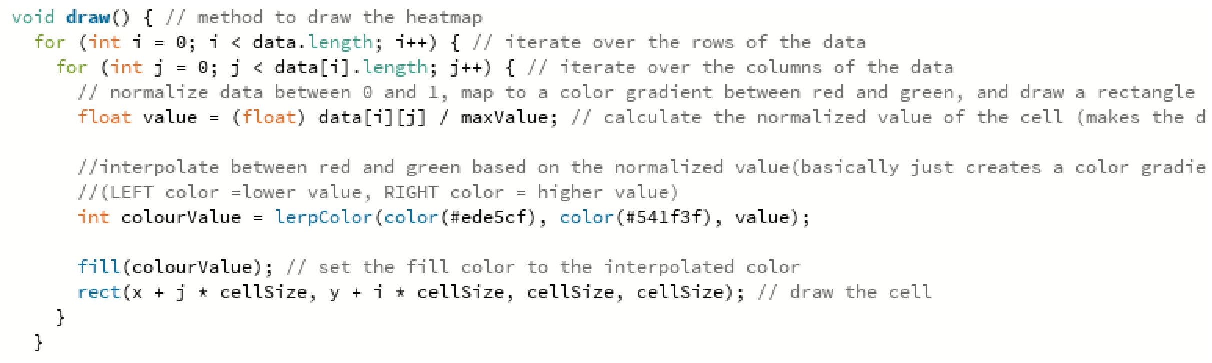
The pie chart class constructs a pie chart object for the selected data (coarse answer), filtered by any parameters typed in by the user. The method getLabels() returns an array list of all the unique values in the selected data column to use as labels for the chart. Next, getData() runs a query for each label, counts the number of data objects that match the label in the selected column, and adds these counts to a list; the order of this list matches the order of the labels. Display() then calculates the angles for each slice based on the number of slices and draws them to the screen along with the corresponding labels.



```
void display() {
  float lastAngle = 0;
  color[] sliceColors = generateColors(numSlices, color(0, 100, 255), color(255, 100, 0)); // generate a color gradient
  for (int i = 0; i < numSlices; i++) {
    float sliceAngle = 360.0 / sum(data) * data[i]; // calculate the angle for each slice
    fill(sliceColors[i % sliceColors.length] + 400*i); // Color of slices
    if (i == numSlices-1) {
      fill(#573b88);
    }
    //noStroke(); // no outline
    strokeWeight(2);
    stroke(#541f3f);
    arc(x, y, diameter, diameter, lastAngle, lastAngle + radians(sliceAngle)); // Slice of pie chart
    float labelRadius = diameter/1.8; // Distance from center of pie chart to label
    float labelX = x + cos(lastAngle + radians(sliceAngle / 2)) * labelRadius; // Text x position
    float labelY = y + sin(lastAngle + radians(sliceAngle / 2)) * labelRadius; // Text y position
    textAlign(CENTER, CENTER);
    fill(255); // Text color
    textFont(font);
    text(labels.get(i), labelX, labelY); // Render text on screen
    lastAngle += radians(sliceAngle); // Angle at which the previous slice ended
  }
```

Heatmap

The Heatmap class is a tool for visualising two-dimensional numerical data. It contains a constructor method that takes in the data to be visualised, the position and size of the cells in the heatmap, the font to be used for the labels, and the labels for the rows. The maxValue method is a helper function that returns the maximum value in the data array, which is used to normalise the data to a value between 0 and 1 so that the colours can be mapped appropriately. The draw method is used to draw the heatmap, iterating over the 2D array and normalising the data to a value between 0 and 1. It then maps each normalised value to a colour gradient between two colours using the lerpColor function, and draws a rectangle for each cell in the heatmap. The Heatmap class also provides methods to draw the column labels, row labels, and a title for the heatmap. In main we create as many Heatmap objects(max size 6 columns*20 rows) based on the size of the 2d int data array.



```
void draw() { // method to draw the heatmap
  for (int i = 0; i < data.length; i++) { // iterate over the rows of the data
    for (int j = 0; j < data[i].length; j++) { // iterate over the columns of the data
      // normalize data between 0 and 1, map to a color gradient between red and green, and draw a rectangle
      float value = (float) data[i][j] / maxValue; // calculate the normalized value of the cell (makes the d

      //interpolate between red and green based on the normalized value(basically just creates a color gradie
      //(LEFT color =lower value, RIGHT color = higher value)
      int colourValue = lerpColor(color(#ede5cf), color(#541f3f), value);

      fill(colourValue); // set the fill color to the interpolated color
      rect(x + j * cellSize, y + i * cellSize, cellSize, cellSize); // draw the cell
    }
  }
```

HeatmapData

This class is used to parse and generate data which is then converted to the appropriate type and sorted for the Heatmap class. It contains methods to store unique values for a given column in a CSV file, count how many times a value in a given list appears in the column from the CSV file, sort arrays in descending order using the bubble sort algorithm, and create a 2D integer array representing the data. In main we use this class to parse the data.

HeatmapButton
This class contains three methods: display(), clicked(), and update(). It is used with the heatmap to return a value associated with the column of data in the CSV file that is to be used to compile the data needed for the heatmap.

Problems Encountered

- At first we tried to just split the data at the comma, but that didn't work for the DEST_CITY_NAME because there is a comma included in the value (ex "Seattle, WA"). We instead had to use a positive lookahead to verify the amount of upcoming double quotes in the string before splitting it.
- The pie chart proved to be a bit harder than anticipated, as we had to filter to get not only the data but the labels for each slice as well.
- A lot of heatmap methods were required to sort the data correctly and map it to an appropriate index in the 2D array whilst making sure it corresponded to a String array.
- Fitting the entire heatmap onscreen for large datasets/more months
- Outliers in the data skewing the ratio of the data relative to each other in the heatmap
- HeatmapData class loops through the CSV file multiple times which can slow down the program.

Splitting the Work

Emma
I wrote the data class to organise the data and convert each value into the appropriate data type. I then wrote the query class to enable the filtering of data, which can be used in the other classes to display only the user requested data. I was also involved in implementing the piechart class; I wrote methods that get the labels and data to fill the chart with.

Travis
I helped implement the initial data class, created the first draft of the Pie Chart class which Emma helped to complete. I then wrote the Heatmap class, the HeatmapData class and the HeatmapButton class and implemented them into the appropriate screen. I wrote methods used for data parsing, data sorting, data visualisation and debugged the graphs to show data correctly.

Luke
I wrote the scrollBar class, and implemented the scrollbar widget to sift through the 'Raw Information' screen proportional to which data file has been loaded in. I also helped write some of the Widget class and cleaned up the UI on each screen to give a more sleek look to the program.

Peter
I created the UI at the right hand side of the screen which allows users to sort the data by certain queries, and by a more specific search. This UI also entailed creating multiple widgets, such as the "DropDown" widget, the checkboxes, and the search bars used for fine searches. I also created the dynamic title that changes based on the query at the top of the screen.

David
I created the BarChart class that displays data as a bar chart using HashMap and ArrayList. I also created the Screen and Widget classes to make a graphical user interface with clickable widgets on the bottom of the screen. This enabled the user to switch between the screens. The Screen has a list of widgets and a background colour, while the Widget defines individual GUI elements like position and label.