Airy Fairy Project Report

How Work was Split Up and Organised:

Before each lab session, we got together to discuss the upcoming week's requirements. Then, we picked tasks to work on and set goals for the tasks we needed to do before and during the lab session. We would all collaborate to find a solution if any team member was having problems completing their tasks. As a result, we made sure that each work was finished before its due date.

Problems Encountered:

In the beginning, many of our team members struggled to adjust to using github as figuring out how to push and pull code was challenging. The issue was finally resolved with assistance from our professors and peers. There was also trouble getting our screens class to function. It was initially somewhat lengthy, but we were able to make it shorter and more effective. We struggled creating bar charts and histograms. The histogram just took a while to refine, while the idea to use bar charts to represent flights per airport was ultimately abandoned.

Design Outline:

The design opens onto a starting screen, displaying the website logo and name "AiryFairy". The aesthetically pleasing bubble letters and colours encourage users. The user is prompted to click anywhere, which leads them to the home screen, and this allows the user to access data and flight statistics by selecting the corresponding buttons. The homepage centres around the idea of a calendar, giving the user access to information regarding each date when they click that number. On the left, a taskbar makes it easy for the user to navigate the website, for example pushing the 'home page' button to go back home at any time. The pie and bar charts make the flight statistics easy to comprehend for the user. The simple and clean structure demonstrates the information a user may need while allowing for a pleasant experience.

Image 1: Start Screen

Extra Benefits of this Solution:

The solution uses a Table class to load the rows and columns of flight data from the given files. Loading in data simply requires entering the name of the data file, provided the file exists in the project folder, so the code can work with many data sets as the different files loaded into the code are easily interchangeable. It's worth noting that this worked especially well for us as we were able to change the size of the dataset we were working with, because they all concerned flights and had the same variables. If a dataset with differing information was passed into the code, having to change variable names throughout would be more effort, however the general framework would remain the same.

Image 2: Table Class Code

```
//Table table=loadTable("flights2k.csv", "header");
Table table=loadTable("flights_full (1).csv", "header");

void initialiseData()
{
  flightDates=new ArrayList<Integer>();
  airO=new ArrayList<String>();
  airD=new ArrayList<String>();
  schDept=new ArrayList<Integer>();
  accDept=new ArrayList<Integer>();
  cancelled= new ArrayList<Integer>();
  distance=new ArrayList<Integer>();
  stateO=new ArrayList<String>();
  stateD=new ArrayList<String>();

  for (int i=0; i<table.getRowCount(); i++)
  {
    TableRow row=table.getRow(i);

    String dateTime = row.getString("FL_DATE");
    int[] dateParts=int(dateTime.split("/"));
    int date=dateParts[1];
    flightDates.add(date);

    String airOInfo=row.getString("ORIGIN");
    airO.add(airOInfo);

    String dest=row.getString("DEST");
    airD.add(dest);

    int scheduledDept=row.getInt("CRS_DEP_TIME");
    schDept.add(scheduledDept);

    int actualDept=row.getInt("DEP_TIME");
    accDept.add(actualDept);

    int can=row.getInt("CANCELLED");
    cancelled.add(can);

    int dist=row.getInt("DISTANCE");
    distance.add(dist);
```

Once data is loaded in, it gets put into tree maps rather than placed in classes. The tree maps order the data alphabetically. They were also created globally, allowing the data to be used across different classes such as the barChart and pieChart classes. Our data is displayed using bar graphs, pie charts. It was more efficient to create a barChart and pieChart class that the array lists can then be plugged into; instead of creating a new class to manipulate the individual subsets of data, there's an existing outline that can be used across the board.

Image 3: Barchart

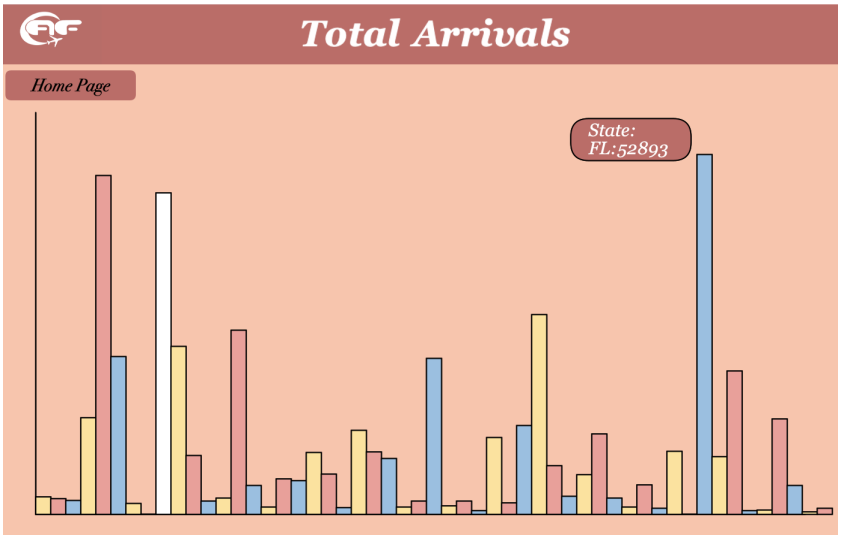


Image 4: Bar Graph code

```
class BarChart
{
  Flights currFlight;

  int[] statesCount = new int[50];
  //int[] airCount = new int[];
  ArrayList<String> data;
  //ArrayList<ArrayList<String>> data2;

  BarChart(ArrayList<String> data)
  {
    this.data = data;
  }



  void countTState()
  {

    for (int i = 0; i < data.size(); i++)
    {
      String destSt = data.get(i);

      switch (destSt)
      {
      case "AL" :
        statesCount[0]++;
        break;
      case "AK" :
        statesCount[1]++;
        break;

      case "AZ" :
        statesCount[2]++;
        break;

      case "AR" :
        statesCount[3]++;
        break;
```

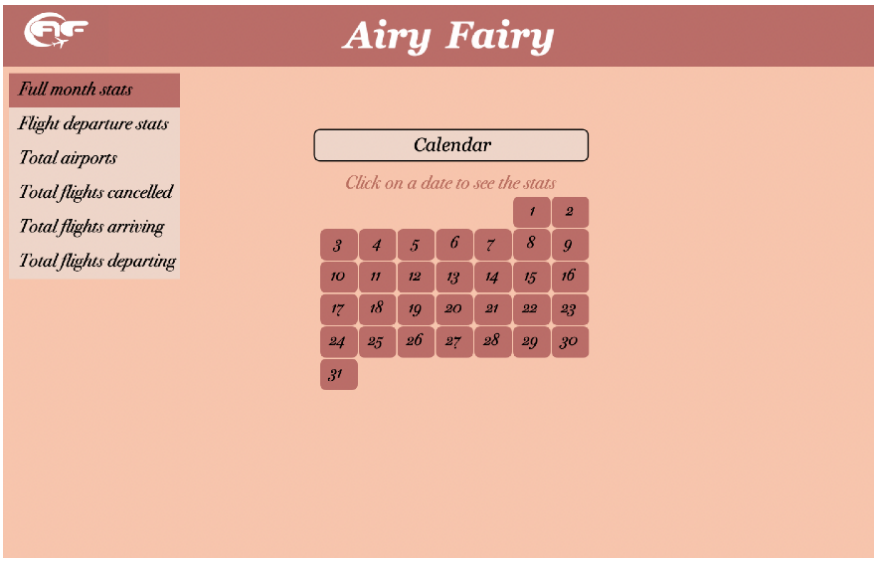
Image 5: Pie Chart code

```
class PieChart
{
  float diam;
  float[] data;
  color[] colors;
  float onTimePerc, latePerc, earlyPerc;
  float canPerc, notCanPerc;

  Flights currflight;

  float countOnTime, countErly, countLate;
  ArrayList<Integer> schDep;
  ArrayList<Integer> accDep;
  ArrayList<Integer> cancelled;


  PieChart (float diam, ArrayList<Integer> schDep, ArrayList<Integer> accDep, color[] colors)
  {
    this.diam = diam;
    this.schDep = schDep;
    this.accDep = accDep;
    this.colors = colors;
  }

  PieChart (float diam, ArrayList<Integer> cancelled, color[] colors)
  {
    this.diam = diam;
    this.cancelled = cancelled;
    this.colors = colors;
  }
```

```
    flights=new TreeMap<Integer, Integer>();
    airportData=new TreeMap<String,Integer>();

    for (int date : flightDates)
    {
      if (flights.containsKey(date))
      {
        int count = flights.get(date);
        flights.put(date, count + 1);
      }
      else
      {
        flights.put(date, 1);
      }
    }

    for (String data : stateO)
    {
      if (airportData.containsKey(data))
      {
        int count = airportData.get(data);
        airportData.put(data, count + 1);
      }
      else
      {
        airportData.put(data, 1);
      }
    }
    airportCount = new ArrayList<Integer>();
    airportName = new ArrayList<String>();

    for (String key : airportData.keySet()) {
    int value = airportData.get(key);
    airportCount.add(value);
    airportName.add(key);
    println(key + ": " + value);
  }
```

User interface is a big part of our solution. There are statistics for the full month of January on the home page including the total airports, flights cancelled, arriving and departing and lateness. However, beyond that the website filters all the data by date, so that users can choose a day to see statistics for, making the information load more manageable for them.

Image 6: Homepage

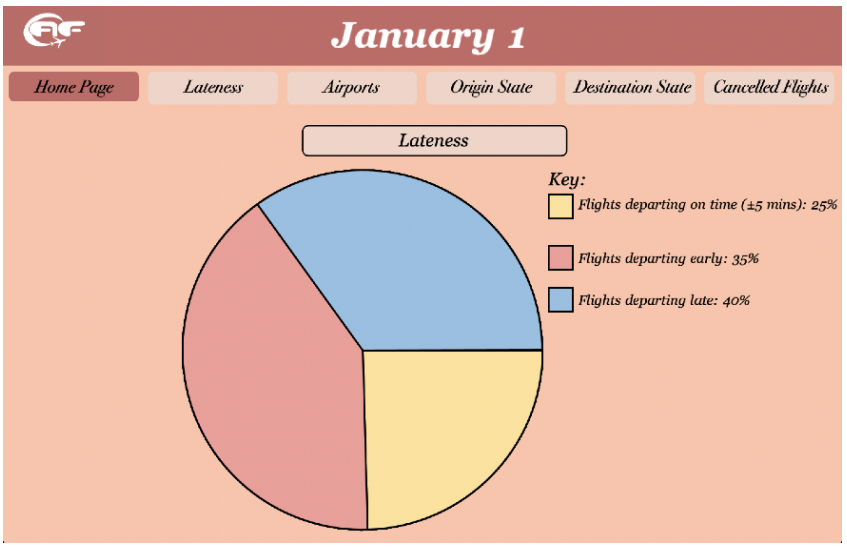


After users select the day they want to see statistics for by clicking the button, they can choose from a list of multiple different queries at the top of the screen, which inform them of helpful things to know regarding flights, such as how likely they are to get cancelled or late and which areas have more flight activity, using the visual data representations of pie charts and bar graphs. There's also a feature to show users the state, and number of flights arriving or departing in that state a specific bar on a graph represents when they hover their mouse over a bar.

Image 7: Query Buttons

```
// labels
lCalendar = new Label (lX, lY, lW, lH, lCol, "Calendar", stdFont, wRad, 660, 235);
lLateness = new Label (lX, lY, lW, lH, lCol, "Lateness", stdFont, wRad, 660, 235);
lAirport = new Label (lX, lY, lW, lH, lCol, "Airport", stdFont, wRad, 665, 235);
lOState = new Label (lX, lY, lW, lH, lCol, "Origin State", stdFont, wRad, 650, 235);
lDState = new Label (lX, lY, lW, lH, lCol, "Destination State", stdFont, wRad, 610, 235);
lCancelled = new Label (lX, lY, lW, lH, lCol, "Flights cancelled", stdFont, wRad, 620, 235);
```

Image 8: Pie Chart



Once everything is sorted onto correct screens in ‘Main’, the program is refined to make it more aesthetically pleasing for the user. From elegant but readable fonts, to colour coordination and organised screens, the website allows for an easy and enjoyable user experience.