Programming Project 7 – Flight Analysis

Note: When you turn in an assignment to be graded in this class you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor).

Program Name: FlightAnalysis.java

You will write a program that will accept at the command line two filenames: one is a text file containing flight data and the other is an output filename. If these files do not come at the command line, then the program will ask the user for them.

The data in the first text file consists of the flight information for some flights throughout the country over the past 3 years. The data contains a header line, which you can skip and then the following information: the year, the first city, the second city, the price of the trip, and the distance between the two cities. A sample text file is below:

Year	city1	city2	mkt_fare	Distance
2018	Asheville, NC	Miami, FL (Metropolitan Area)	119.6646127	677

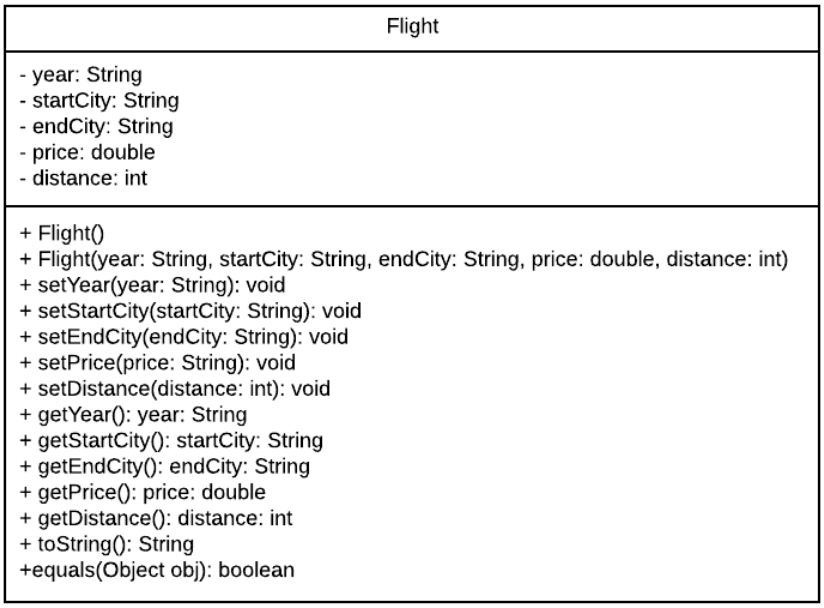
The program will create an ArrayLists of Flight objects that will each contain one row of input data. The array list will be called allFlights.

The main method will call the method **parseData** that will read in the data from the file, create a Flight object and place the object into an ArrayList of Flight objects. The method will return the ArrayList to the main method. The main method will store this ArrayList in a variable called allFlights.

You will wrap the call to this method in a try catch block, so that if there is a problem with the opening of the text file, you will catch it in main. If there is a problem, you will print to the console "Incorrect input filename". If there is not a problem, then you will print to the console "Input file corrrect".

```
public static ArrayList<Flight> parseData(File inputFile){
     //Your code here
}
```

The Flight class, **Flight.java** will have five instance variables: year, startCity, endCity, price, distance. You will have a parameterized constructor that will create an object by setting all these values to the values passed in. The default constructor will set the year to "1970", the startCity to "Boston", the endCity to "Richmond", the price to 40.0, and the distance to 900. You will have getters and setters for each instance variable. You will have a toString() method that will return the year, the startCity, the endCity, the price, and the distance separated by a space. You will also have an equals method that overrides the equals method in the Object class. It will take in an object and then return a Boolean true or false depending on if each instance variable is equal to the Flight object's instance variables passed in. You can use the test instanceOf method to make sure the object passed in is an instanceOf the object Flight before you cast the object as a Flight object. See the Point class for an example.



You will read in the data from the text file one row at a time using a while loop. The loop can use the `in.hasNextLine()` to check to see if there is a line to read in. The data in the text file is separated by semi-colons, so once the line is read in, you can use the String.split(";") method to split the code into an array of Strings.

You will then use this array of Strings to create your Flight object. You will need to use the Double.parseDouble() and Integer.parseInt() methods to convert your String input into the numeric values you need for the Flight price and distance variables. You will want to make sure that your numeric data actually is numeric. You will handle this possible conversion error by catching the exception that is thrown, the NumberFormatException. If the file has incorrect data or negative values for any of the numeric inputs, you will simply just place a 0 or a 0.0 into the numeric instance variables instead of the incorrect value.

Once the data has been read into the ArrayList,,the main method will create the file object:

```
File removeDataFile = new File("dataToRemove.txt");
```

You will then call the parseData method once again passing it the removeDataFile. Store the ArrayList that is returned into a variable called removeItems. You will then call the method called **removeData()**. This method will remove the items from allFlights that are found in removeItems.

```
public static void removeData(ArrayList<Flight> allFlights,
ArrayList<Flight> removeItems){
      //Your code here
  }
```

The main method will then call two methods. The first method, called **calcBestFare()**, that will return the Flight object that is associated with the best priced fare in the file.

```
public static Flight calcBestFare(ArrayList<Flight> allFlights){
     //Your code here
}
```

The second method that the main method will call is **calcBestFarePerMile()** This method will calculate which fare is the best deal given how many miles you are travelling. It will return the Flight object that is associated with the best priced fare in the file.

```
public static Flight calcBestFarePerMile(ArrayList<Flight>
allFlights){
     //Your code here
}
```

The main will write the values in the ArrayList to the given outputFile parameter by calling a **writeOutData** method. The Flight objects from the ArrayList will be outputted to the outputFile by a call to the **writeOutData** method. Each output will be on a new line. You will wrap the call to this method in a try catch block, so that if there is a problem with the writing to the text file, you will catch it in **writeOutData**. If there is a problem, you will print to the console "Incorrect output filename". If there is not a problem, then you will print to the console "Output file corrrect".

```
public static void writeOutData (ArrayList<Flight> allFlights,
File outputFile){
     //Your code here
}
```

Sample output for the sample file is:

At the console line:

```
Incorrect input filename or Input file correct
Incorrect output filename or Output file correct
```

To the output file:

```
2017 Boise, ID Spokane, WA 106.6399809 287
2016 Cincinnati, OH Minneapolis/St. Paul, MN 375.9145278 334
```

Before beginning this project, you will document your algorithm as a list of steps to take you from your inputs to your outputs. This algorithm will be due on paper at the beginning of class on the day it is due. This will be graded and returned to you. It will be your responsibility to understand and correct any errors you might have with your algorithm.

Each step will be added as a comment block within your code. You will have the comment block right above the code that performs the actions specified. For example, before your lines of code that ask the user for inputs, you would have a comment block that states what inputs you are requesting from the user.

This and all program files in this course must include a comment block at the beginning (top) of the source code file that contains:

- the Java program name
- project description
- your name
- the version date
- the course number and section

The comment block for the header as well as all the comment blocks should look like this:

```
/*******************************************************************************
* Java program name
 *******************************************************************************
* Project description
*
 _______________________________________________________________________________
* Your name
* The version date
* The course number and section
 *******************************************************************************/
```

You will document your tests using the form shown below. Your tests should cover each possible path through your program.

You will submit your Java source code file (**FlightAnalysis.java** and **Flight.java**) to Gradescope. You will upload your test cases (TestPlan.docx) by uploading the file to the Assignment link in Blackboard. Please do not submit your files in a zipped folder.

Ask questions about any part of the programming project that is not clear!

Test Plan:

You will fill out the variables and methods to test.

Test Inputs	1	2

Rubric for Programming Project 7

Criteria	Points
Algorithm submitted in class on time	10
Comments used appropriately (including comment header information and fully documented algorithm)	15
Opens and reads input file correctly	10
Correctly creates the file objects	10
Correctly removes items from the array list	5
Correct calculating methods	10
Output is correct	10
Errors are correctly handled	10
Output is written correctly to the output file	10
Test plan completed and submitted with source code	10
TOTAL	100