CMSC B113 - Computer Science 1
Fall 2020
Homework Assignment #7

Overview

In this assignment, you will take on the role of a consultant who has been hired by a major airline to analyze data regarding flights into and out of Philadelphia International Airport. You will develop **a few programs** that allow the airline to understand the extent of their flight delays so that they can address them and better serve their customers.

You will be writing a total of five **short** Java programs:

1. A class that defines a **`Flight`** object containing data about a single flight.
2. A reader library that reads flight data from either **`StdIn`** or a data file.
3. A client program that computes the number of flights that operated on a given day.
4. A client program that computes average delay in a year for a specific flight.
5. A client program that computes the % of delayed flights that recovered from the delay.

All client programs will use the **`Flight`** data type (Part 1), and the reader library (Part 2).

You should plan to implement the program in stages, taking a break after completing each part. We recommend the following timeline:

- Finish Part 1 as soon as possible.
- Complete Parts 2 and 3 by the end of the week when the assignment is posted.
- Complete Part 4 well before the deadline.
- Part 5 is optional, for earning Extra Credit.

For each part, first read the explanation, then read it again! Then devise a solution to that part by clearly identifying all the inputs and outputs and sketching out the steps and calculations that will need to be performed. Code the steps into a Java program, then compile and run it. Double-check the program's answers with the ones we have provided to confirm that the results are correct.

Grading Rubric:

Part 1 is worth 1.7/4.0 of the grade for this assignment.
Part 2 is worth 1.0/4.0.
Part 3 is worth 0.6/4.0.
Part 4 is worth 0.7/4.0.
Part 5 is worth 0.3/4.0 as Extra Credit.

All submissions are due on the due date and time posted for this assignment.

Late submissions will not receive any credit.

Learning Outcomes

In completing this assignment, you will learn to:

- Define a custom class in Java
- Write a library class that can be used in multiple programs
- Write a Java program consisting of multiple source code files

Code Understandability and Readability

In addition to producing the correct output, in this assignment you are also asked to write code that is easy to read and understand. As with previous assignments, part of your score on this assignment will be determined by:

- **Variable naming:** Variables should have meaningful names that indicate what they represent, using full English words or common abbreviations, e.g. “wins” or “votes” instead of “w” or “vot”. Variable names should start with a lowercase letter and use “camelCaps”, e.g. “creditCardNumber” instead of “creditcardnumber” or “credit_card_number”.

- **Method naming:** Methods should have meaningful names that indicate what they do. Like variable names, they should use full English words or common abbreviations, start with a lowercase letter, and use camelCaps.

- **Appearance:** The code should be formatted so that indentation and spacing make it easy to understand which parts of the code are within the bodies of if-statements and loops. Additionally, there should be spacing between variables and operators to make it easy to read each individual line of code.

For example, a method that determines the maximum value in an array should look something like this:

```java
public static int findMax(int[] values) {
    int max = values[0];
    for (int i = 1; i < values.length; i++) {
        if (values[i] > max) {
            max = values[i];
        }
    }
    return max;
}
```

Not like this:

```java
public static int method(int[] values) {
int a=values[0];
for(int i=1;i<values.length;i++) {
if(values[i]>a) {
a=values[i];
}}
return a;
}
```

Please speak with your Instructor if you have any questions about this requirement!

Part 1: Define a Flight class

First you will create a class that represents a single flight from one airport to another. This class will be used throughout the rest of this assignment. Each flight is modeled to store the following information:

```
2020 11 23 113 3 8 PHL SFO
```

The above represents a date (year/month/day: 2020/11/23), followed by the flight number (113), its departure delay (in minutes: 3), arrival delay (in minutes: 8), its origin airport (PHL), and its destination airport (SFO). The above data represents the flight instance:

Flight 113 from PHL departed 3 minutes late on 11/23/2020 and arrived 8 minutes late at SFO

In this part, you will create a **`Flight`** class that has the following:

```
 public  class Flight
─────────────────────────────────────────────────────────────────
private  int year                                       instance
private  int month                                      variables
private  int day
private  int flightNum
private  int departureDelay
private  int arrivalDelay
private  String origin
private  String destination

 public  Flight(int y, int m, int d, int fn,            constructor
                int dd, int ad, String org, String des)

 public  int getYear()                                  accessors
 public  int getMonth()
 public  int getDay()
 public  int getFlightNumber()
 public  int getDepartureDelay()
 public  int getArrivalDelay()
 public  String getOrigin()
 public  String getDestination()

 public  String toString()                              print method
```

Notice that each piece of information about the flight is represented as a *private instance variable* of the **`Flight`** class. The *print method* **`toString()`** should return a **`String`** as follows, with the placeholders replaced by the values of the instance variables:

"Flight [**`flightNum`**] on [**`month`**]/[**`day`**]/[**`year`**] had a delay of [**`departureDelay`**] leaving [**`origin`**] and a delay of [**`arrivalDelay`**] arriving at [**`destination`**]."

Once you have implemented the **`Flight`** class as described above, write a **`main()`** method in it that tests the **`Flight`** class by creating a new instance and then printing the object, for example:

```
    public static void main(String[] args) {
        Flight f = new Flight(2020, 11, 23, 113, 3, 8, "PHL", "SFO");
        StdOut.println(f);
    }
```

Compile and run your program; if you run your program using the code above like this:

```
$ java-introcs Flight
```

then the output should be as follows:

```
Flight 113 on 11/23/2020 had a delay of 3 leaving PHL and a delay of 8
arriving at SFO.
```

Be sure you complete this part before moving on to Part 2!

Part 2: Create a flight data reader

Next, you will create another class, **`FlightReader`**, that enables its clients to read flight data about several flights arriving at or departing from an airport. For example, here are the contents of a small data file (**`testFlights.txt`**):

```
3
2020 11 23 113 3 8 PHL SFO
2020 11 24 206 2 -3 ORD PHL
2020 11 25 231 0 4 PHL SEA
```

The file contains information for three flights. The first line indicates how many flights are recorded in the file. The **`FlightReader`** class has the following static methods defined in it:

`public class FlightReader`	
`public static Flight[] readFlights()`	*reads data from `StdIn`*
`public static Flight[] readFlights(String file)`	*reads data from `file`*

It contains two static methods, both called **`readFlights()`**, that read data from **`StdIn`** or a data file as described above. Both perform the same task: they read the data, create flight objects for each line of data, and store them in a **`Flight[]`** array. Once done, the **`readFlights()`** method returns the array of **`Flight`** objects. The **`readFlights()`** method will be used in the remaining parts of this assignment.

First, write the class with the first **`readFlights()`** method and test it by creating the **`FlightReader`** class as defined above, starting with the **`readFlights()`** method for reading from **`StdIn`**. Because you will need to know the size of the array when you create it, the first thing the **`readFlights()`** method should read from **`StdIn`** is the number of flights. Following that, it should read the data from **`StdIn`** in the following order, where each value is separated by a single space:

```
year month day flightNum departureDelay arrivalDelay origin destination
```

To test your implementation, write a **`main()`** method in **`FlightReader`** that invokes **`readFlights()`**, and then prints out each **`Flight`** in the array that is returned, for example:

```java
public static void main(String[] args) {
      Flight[] flights = readFlights();
      for (Flight f : flights) {
          StdOut.println(f);
      }
  }
```

Compile and run your program. If you were to save the above input (with the information about the three flights) to a file named **testFlights.txt**, and ran your program like this:

```
$ java-introcs FlightReader < testFlights.txt
```

then the output should be as follows:

```
Flight 113 on 11/23/2020 had a delay of 3 leaving PHL and a delay of 8
arriving at SFO.
Flight 206 on 11/24/2020 had a delay of 2 leaving ORD and a delay of -3
arriving at PHL.
Flight 231 on 11/25/2020 had a delay of 0 leaving PHL and a delay of 4
arriving at SEA.
```

Great!

Next, implement the second **`readFlights()`** method, which reads data from a given file:

```
        public  static Flight[] readFlights(String file)    reads data from file
```

In order to read data from a given file, you will need to use the **`In`** library provided by the authors of your text. It is described in Section 3.1 (pages 354-355). Make sure you read these before proceeding.

This version of **`readFlights()`** works exactly the same way as the one you wrote above. However, instead of reading from **`StdIn`**, you will now read from the **`In`** object that you can create as shown below:

```
public static Flights[] readFlights(String file) {
    In inFile = new In(file);
    // Now read from the inFile object instead of StdIn…
    . . .
} // readFlights()
```

Complete the above implementation in the **`FlightReader`** class. Then modify the **`main()`** method as shown below:

```
    public static void main(String[] args) {
        Flight[] flights = readFlights(“testFlights.txt”);
        for (Flight f : flights) {
            StdOut.println(f);
        }
    }
```

Compile your program and then run it; note that you no longer need to I/O redirection since you are not reading from stdin:

```
$ java-introcs FlightReader
```

The output should be as follows:

```
Flight 113 on 11/23/2020 had a delay of 3 leaving PHL and a delay of 8
arriving at SFO.
Flight 206 on 11/24/2020 had a delay of 2 leaving ORD and a delay of -3
arriving at PHL.
Flight 231 on 11/25/2020 had a delay of 0 leaving PHL and a delay of 4
arriving at SEA.
```

You can now provide a file name to `readFlights()` to read data from *any* file. In fact, the file name could also be a complete URL of a file anywhere on the web! We will exploit this new functionality in the next three parts.

Be sure you complete this part before moving on to Part 3!

Part 3: Client #1 - Number of flights on a specified day

Now that you have a method that can convert an input file to an array of **`Flight`** objects, you are ready to start writing some programs that can query that data. We will do this by writing three separate client programs. The data we will use is a file containing all the flights flying into and out of Philadelphia International Airport (PHL) in the year 2008. The data is provided to you in the file at this URL: https://www.cs.brynmawr.edu/~cdmurphy/phl-flights.txt

The file contains data for almost 200,000 flights. It is large enough that we will not make copies of the file and you do not need to download it. Instead we will use the second **`readFlights()`** method written in Part 2 that reads from the **`In`** object, and use the URL as the name of the input file. All three client programs will use this data file.

The first client, which you will write in this part, reads a date as the runtime argument. It then scans the array of **`Flight`** objects for all the flights that flew in and out on that date. This client, **`FlightCounter`**, counts them and prints the count as its answer. How many flights do you think flew in and out of PHL on your birthday?

For instance, using the small **`testFlights.txt`** data file from Part 2, if you run your program like this:

```
$ java-introcs FlightCounter 2020 11 25 < testFlights.txt
```

then the output should look something like this:

```
Number of flights on 11/25/2020: 1
```

Your program can assume that the three runtime arguments all exist and are all integers. You do not need to determine whether the numbers represent valid years, months, or days.

In a file named **FlightCounter.java**, define a class named **`FlightCounter`** that contains a **`main()`** method that reads a year, month, and day as command line arguments in that order and displays the number of flights on that date, based on the flight information produced by the **`FlightReader.readFlights()`** method from Part 2.

Your **FlightCounter** program must use the **`FlightReader.readFlights()`** method from Part 2 to get all the **`Flight`** objects and should not read the input itself. Note that your code will need to invoke **`FlightReader.readFlights()`** instead of just **`readFlights()`**, since the **`readFlights()`** method is defined in a different class. In the example above, we are using the first version of **`readFlights()`** that reads data from **`StdIn`**.

Once your program is giving correct output on the **`testFlights.txt`** data, change the call to **`readFlights()`** method to read the big data file from
"`https://www.cs.brynmawr.edu/~cdmurphy/phl-flights.txt`"

To demonstrate that your **FlightCounter** program works correctly, run your program for the following input dates. Your program output should be as follows:

```
$ java-introcs FlightCounter 2008 6 10
Number of flights on 6/10/2008: 565
```

```
$ java-introcs FlightCounter 2008 7 14
Number of flights on 7/14/2008: 593

$ java-introcs FlightCounter 2008 12 25
Number of flights on 12/25/2008: 462
```

Be sure to include the program output for these inputs in the write-up that you submit for this assignment. You can now also try your birthday to see how good your guess was.

Be sure you complete this part before moving on to Part 4!

Part 4: Client #2: Average delay for a specified flight number

Next, we will compute the average flight arrival delay, **AverageDelay**, for a given flight. For example, for the **testFlights.txt** data, if you run the program like this:

```
$ java-introcs AverageDelay 231 < testFlights.txt
```

then the output should look something like this:

```
Flight 231 has an average arrival delay of 4
```

However, if no flight with the specified flight number is found, then the program should indicate that the flight number is not valid. For instance, if you run your program like this:

```
$ java-introcs AverageDelay 123 < testFlights.txt
```

then the output should look something like this, since there is no flight number 123:

```
123 is not a valid flight number
```

In a file named **AverageDelay.java**, define a class named **AverageDelay** that contains a **main()** method that reads a flight number as a runtime argument and displays the average arrival delay for that flight number, based on the flight information produced by the **FlightReader.readFlights()** method from Part 2. As with Part 3, your **AverageDelay** program must use the **FlightReader.readFlights()** method from Part 2 to get all the **Flight** objects and should not read the input itself.

The average delay should be displayed as an integer; you should truncate the value when displaying it, e.g. if the average delay is calculated as 5.75, it should be displayed as 5.

Test your program with the **testFlights.txt** data file (as shown above). Once your program is giving correct output on the **testFlights.txt** data, change the call to **readFlights()** method to read the big data file from https://www.cs.brynmawr.edu/~cdmurphy/phl-flights.txt

To demonstrate that your **AverageDelay** program works correctly, run your program using the large input file for flight numbers **22**, **259**, **1233**, and **1300**. Your program output should be as follows:

```
$ java-introcs AverageDelay 22
Flight 22 has an average arrival delay of 92

$ java-introcs AverageDelay 259
Flight 259 has an average arrival delay of 8

$ java-introcs AverageDelay 1233
Flight 1233 has an average arrival delay of -3

$ java-introcs AverageDelay 1300
1300 is not a valid flight number
```

Be sure to include your program output for these inputs in the write-up that you submit for this assignment.

Part 5: Client #3 - Percentage of flights that recover from delays from a specified airport [Note: This part is optional. For Extra Credit only.]

In a file named **RecoveredFlights.java**, define a class named **`RecoveredFlights`** that contains a **`main`** method that reads an airport code as a runtime argument and displays the percentage of delayed flights originating at that airport that recovered from the delay (as defined below) based on the flight information produced by the **`FlightReader.readFlights()`** method from Part 2.

For our purposes, we will say that:

- a **delayed flight** is one that has a departure delay greater than 0
- a **recovered flight** is a delayed flight that has an arrival delay less than or equal to 0.

That is, a recovered flight is one that was delayed when it departed, but arrived early or on time, and your program should display the result of (# recovered flights) divided by (# delayed flights) as a percentage.

The percentage should be displayed as a floating point number between 0 and 100 to two digits of precision; your program should round the value when displaying it, e.g. if the percentage is calculated as 85.3275, it should be displayed as 85.33%.

However, if there were no flights with the specified airport code as their origin and a delay greater than 0, then the program should show a message indicating that there were no delayed flights from that airport.

As with previous parts of this assignment, your **RecoveredFlights** program must use the **`FlightReader.readFlights()`** method from Part 2 to get all the **`Flight`** objects and should not read the input itself.

HINT! Be careful about how you compare the **`String`**s representing the airport codes!

To demonstrate that your **RecoveredFlights** program works correctly, run your program using the large input file from https://www.cs.brynmawr.edu/~cdmurphy/phl-flights.txt for airport codes **JFK**, **DFW**, and **YYZ**. Your program output should be as follows:

```
$ java-introcs RecoveredFlights JFK
34.94% of delayed flights from JFK recovered

$ java-introcs RecoveredFlights DFW
22.70% of delayed flights from DFW recovered

$ java-introcs RecoveredFlights YYZ
There were no delayed flights from YYZ
```

Be sure to include the program output for these inputs in the write-up that you submit for this assignment.

<u>Submitting Your Solution</u>

To submit your work, copy/paste all of the **formatted** Java source code created for this assignment, as well as the outputs produced in Parts 3-5, into a *single* PDF, and submit the PDF in the “Assignment #7” subfolder of the Dropbox folder that your instructor created for you.

Be sure you include the source code for **Flight.java, FlightReader.java, FlightCounter.java, AverageDelay.java**, and **RecoveredFlights.java** (if you did Part 5) in the PDF that you submit.

Please be sure to put all of your code and program outputs into a *single* document and please only submit a PDF, not a Microsoft Word .docx file or the .java source files. Your work will not be graded if you submit anything other than a PDF file.