CS 313: Intermediate Computer Programming Individual Project (Friday, 16th Sept. 2022)

Instructions

You must work individually on this project. You are allowed to think through this project with other students, however, remember that each student should be fully engaged and fully understand all work done, so that you can contribute effectively and learn from others in the process. You then leverage on those insights to challenge yourself to complete this project. The Honour Code applies here!

Submission

You are supposed to submit the GitHub project link (e.g., dadjepon/ICP2022_Java (github.com) or *https://github.com/dadjepon/ICPJavaIndividualProject2022*) to your project (containing your code and data files, readme.txt file and 1-page reflective report) on CANVAS by **11:55pm** on **Thursday 29th September 2022**.

[50 POINTS]

Objectives

The objectives of this project work are for you to demonstrate that you can:

- Define classes, fields, constructors, methods in Java and error handling

- Use appropriate types, including collections and justifiable data structures

- Implement basic algorithms and appropriate technologies (e.g., through APIs)

- Write a main method including console I/O

- Document your solution with Javadoc

Contents

Problem Description

You have been provided with a database of airports, airlines and routes. Your task is to write a programme which, given a start city (e.g., Accra, Ghana) and a destination city (e.g., Winnipeg, Canada), outputs a series of flights that take a passenger from the start city to the destination city, such as in the example below.

```
      1. BA from ACC to LHR 0 stops
      2. DL from LHR to MSP 0 stops
      3. DL from MSP to YWG 0 stops.
Total flights: 3
Total additional stops: 0
```

(To understand the output, note the following: As specified in the data files provided, "BA" and "DL" are the airline codes for British Airways and Delta Airlines respectively. "ACC" is the airport code for Kotoka International Airport in Accra Ghana, "LHR" is the airport code for London Heathrow Airport, "MSP" is the airport code for Minneapolis–Saint Paul International Airport and "YWG" is the airport code for James Armstrong Richardson International Airport in Winnipeg, Canada.) The input information (start and destination cities) will be provided to your application in a text file, and your programme will also provide the output information in a text file.

At a basic level, your programme should find a **valid** route between the two cities. For an extra challenge, your programme can find an **optimal** route between the two cities. Optimality could be judged either in terms of the number of flights or in terms of the distance covered by the route (you'll need to use the Haversine formula to compute the distance between two points with given latitude & longitude coordinates).

Program Execution Specifications

Your programme should be written in Java, and it should be such that it can be executed by taking the input filename, process it and generate the result in an output file.

Note that your programme will use the data files provided – do not change the names of the data files. Your programme should write its results to an output file whose name is the name of the input file, with "_output" inserted between the file name and the extension. E.g., if the input file is "accra-winnipeg.txt", the output file would be "accra-winnepeg_output.txt". For simplicity, you can assume the extension of the input and output files are always .txt. The format of the data files, input file, and output file are described in the following sections.

Data File Formats

You have been provided with three data files in CSV (comma-separate-value) format, downloaded from https://openflights.org/data.html. Note that CSV files are simply plain text files in which the data values on each line are separated by commas. You can read such files in your Java program like you would read any text file. You can view such files with a text editor (such as Notepad). You can also use Excel to view such files in a nice tabular format.

The data files are:

- **airports.csv:**
  - A database of airports spanning the globe. Each entry contains the following information: Airport ID, Name, City, Country, IATA code, ICAO code, Latitude, Longitude, Altitude, Timezone, DST (Daylight savings time), Tz database time zone, Type, Source of this data.
- **airlines.csv**:
  - A database of airlines. Each entry contains the following information: Airline ID, Name, Alias, IATA code, ICAO code, Callsign, Country, Active
- **routes.csv**:
  - A database of routes/flights between airports on various airlines spanning the globe. Each entry contains the following information: Airline code, Airline ID, Source airport code, Source airport ID, Destination airport code, Destination airport ID, Codeshare, Stops, Equipment

More details about the meaning of each of the data entries can be found on the website above.

Input File Format

The input file should have two lines: the start city and country should be on the first line, and the destination city and country on the second line. There should be a comma between the city name and the country name. For example:

Sample input file:

```
Accra, Ghana
Winnipeg, Canada
```

Output File Format

Your output file should have a numbered list of flights of the form "<airline code> from <airport code> to <airport code> n stops" where n is the number of stops made by the flight (as indicated in the routes.csv file). The first flight must depart from the start city and the last flight must arrive in the destination city. Your file should have two additional lines indicating the total number of flights needed and the total number of additional stops made on those flights. Optionally, you file can include the total distance covered by the flights. If your programme computes the optimal route, you can also indicate the optimality criteria (distance or number of flights)

Sample output file:

```
      1. BA from ACC to LHR 0 stops
      2. DL from LHR to MSP 0 stops
      3. DL from MSP to YWG 0 stops.
 Total flights: 3
 Total additional stops: 0
```

Sample output file with optional entries:

```
      1. BA from ACC to LHR 0 stops
      2. DL from LHR to MSP 0 stops
      3. DL from MSP to YWG 0 stops.
  Total flights: 3
  Total additional stops: 0
  Total distance: 10453km
  Optimality criteria: flights
```

* Please note that the values listed above are not necessarily correct!

What to submit

You should submit:

1. Your code and data files
2. A readme.txt file indicating any additional information needed to run your code.
3. A 1-page reflective report/document briefly describing your high-level approach, lessons learnt and listing any references (see rules below).

Rules of the Challenge

1. Solutions must be coded in Java. Standard built-in classes and libraries can be used (e.g. file reading libraries, the math library classes in the java.util package in Java). If planning to use any non-standard packages, it is best to enquire first about whether the package will be allowed.
2. All submitted code must be your own. Code may not be copied from any source.
3. You are allowed to do research if needed. However, your document **must** cite **all** resources you consult.

Evaluation Criteria [50 POINTS]

Solutions will be evaluated according to the following criteria. The order below will be used to assess the various submissions.

1. **[25 points]** Completion & correctness.
   a. **[20 points]** Your solution should indicate a valid route between the two pairs of cities: [Java Classes, Error Handling, File I/O]
   b. Extra credit **[5 points]** will be given if your solution indicates the distance of the route, and/or if your solution computes an *optimal* route.
2. **[10 points]** Efficiency (the time it takes to execute your solution approach – *Time Complexity*).
3. **[5 points]** Code formatting and structure: code should be modular, well-structured, well-formatted, and well-commented.
4. **[10 points]** Clarity of 1-page documentation of approach: [Reflective Report]

All the best!