CSCI 2540 Assignment 7

Due date: Thursday, March 30 (before class)

Getting Started

To begin this assignment, you must set up your project by performing the following steps.

Download the necessary files and have them set up for this assignment:

1. Download the FlightSearch ZIP file posted on Blackboard.
2. Open the downloaded ZIP file and find the **flightSearch** package, and three **text** files.
3. Copy the *flightSearch* package and the three *text* files into your 2540 project folder.
   The text files should go into the root of your 2540 project folder.
4. Open Eclipse, right-click the mouse on the *2540* project folder, and choose *Refresh*. This should cause the new *flightSearch* package and the three text files to be displayed under your 2540 project folder.
5. Rename the package with the require package name (assg7_YourLastname) and change the package name in all the source code files as well.

The Assignment

Create the flight-search program that has been discussed in class. The problem is also discussed in the textbook, at the end of the *Stack* chapter.

As a recap, the program should perform the following steps:

1. Create a *FlightMap* object. Use the object's *loadFlightMap* method to read the data from a file called *cityFile.txt* and a file called *flightFile.txt,* and store this data into the FlightMap object.

2. Open the file called *requestFile.txt*. Read the first line from the file. The first line from this file contains two city names, separated by a *tab* character. This represents a request to fly from the first city on the line to the second city on the line.

3. Use the FlightMap's *servesCity* method to determine if the airline serves both cities in the request.

4. If both cities are served by the airline, then use the FlightMap's *getPath* method to determine the path of flights to take from the origin city to the destination city.

5. Print to the screen one of the following messages:
   a. **The airline does not serve city *X***

   b. **No sequence of flights exist between cities *X* and *Y***

   c. **The following sequence of flights exist between cities *X* and *Y*:**
      (and then display the flight path)

6. Read the next line (request) from the request file and repeat steps 3-6. Continue doing this as long as there are requests left in the request file.

File Summary:

The following describes the files which you will find in your downloaded ZIP file.

Files used to test your main program:

- **cityFile.txt -** A file containing a list of all cities served by the airline; one city name per line.
- **flightFile.txt -** A file containing a list of all adjacencies between cities. Each line contains two city names, separated by a tab character. The first city is the origin city and the second city is the destination. Remember that an adjacency represents a direct flight from the origin to the destination.
- **requestFile.txt -** A file containing a list of request to fly from one city to another. Each line contains two city names, separated by a tab character. The first city is the origin city and the second city is the destination.

  --- Note: the above three files are just samples you can use for testing purposes. These files currently hold the data representing the graph which was shown in class, but you are free to change it as you like so you can better test your program. These may NOT necessarily be the files with which I will use to test your program when I grade them!

Code Files

- **StackInterface -** An interface file describing all methods that are to be included in a stack.

- **Stack -** The beginning skeleton for a stack class which implements StackInterface. No public methods are allowed in this class, other than the ones which are already present. You may, however, include whatever private methods you wish. You are also free to choose how you want to implement the Stack (i.e. array based, reference based, etc). The code for all three is listed in your book, so you don't really have to write any new code in this class. Simply copy from the book for whichever implementation you wish to use.

- **Node -** The node class that has been created and discussed in our lectures. This is needed to create a reference-based Stack, if you wish to create your stack in that manner. If you choose to make it array-based, then you probably will not need to use this class.

- **StackException -** An exception object which is thrown by several of the Stack methods.

- **FlightMapInterface** – An interface file describing all methods that are to be included in a Flight Map.

- **FlightMap** – The beginning skeleton for a flight map class which implements FlightMapInterface. No public methods are allowed in this class, other than the ones which are already present. You may, however, include whatever private methods you wish. Note: the code for most of the *getPath* method is listed in your book, and was also written in class, so you can simply copy the code into this class.

- **City** – The beginning skeleton for the city class referred to in severl of the FlightMap methods. No additional public methods should be added to this class. This class is primarily designed to store a city name, along with a variable which indicates whether or not the city has been visited. Note: the *FlightMap.getPath* method shown in your book uses a *compareTo* method which is expected to be in the City class. I have not included this method because it really isn't needed. Instead, in this City class, the *equals* method has been overridden because it can be used in place of the *compareTo* method in the book's *getPath* code.

- **DeterminePaths** – The main program class. This class is responsible for creating the FlightMap object, and reading and processing requests from the request file.

Summary of what you will need to complete:

- Add code to the **Stack** class. You should simply be able to pick one of the implementations from the book and copy it into this class.
- Add code to the **FlightMap** class. Most of the *getPath* method is in the book, so you can simply copy that into this class. The other methods, however, you will need to write yourself.
- Add code to the **City** class.
- Add code to the **DeterminePaths** class.
- **DO NOT CHANGE** any of the code in any other file.

Technical Notes

- You are **NOT** allowed to add any additional public methods to any of the classes except the main program class (DeterminePaths). You can, however, include any private methods that you feel you need to help with writing the circularly-linked class.

- Do not change the directory structure of the packages, except for renaming the package.

- Suggestions for writing this program:
    - Begin by writing the code for the Stack class. Test the methods in the stack class to ensure they work correctly before proceeding. Remember that you can test the methods by writing a JUnit test file, or by writing a short main program designed specifically for the purpose of testing the Stack class.
    - Next, write the code for the City class. Test it thoroughly to ensure all of the methods work correctly.
    - Next, write the code for the FlightMap class. Again, test the methods thoroughly as you write them. Notice that some of these methods can be written by calling on other methods in the class (for example the *loadFlightMap* method can make use of the *insertAdjacent* method).
    - Once all of the data structures have been THOROUGHLY tested, then proceed to write the real main program.

- **ANY PROGRAM THAT DOES NOT COMPILE WILL RECEIVE AN AUTOMATIC GRADE OF "F"**, no matter how simple the error may be!

To submit your programs, you need to submit your programs electronically on Blackboard. Please also bring a hard copy of your programs to the class to submit. Make sure you put all your source code files (and only source code files) in one package and name the package as assg7_Yourlastname. When you submit your files to Blackboard, please submit your package folder (with source code only, i.e., .java files) as one zip file.