Course code and name:	F28DA – Data Structures and Algorithms
Type of assessment:	**Individual**
Coursework Title:	Flying Planner
Student Name:	Akshay Arunkumar Garg
Student ID Number:	H00338776

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature: *AkshayArunkumarGarg*

Date: *14/04/2022*

SCHOOL OF MATHEMATICS & COMPUTER SCIENCE

Flying Planner

Prepared by:
Akshay Arunkumar Garg

Prepared on:
14th April 2022

Person Identification:
H00338776

Programme:
BSc Computer Science (AI)

Course :
Data Structures and Algorithms

Campus:
Dubai

TABLE OF CONTENTS

OVERVIEW

In this coursework, we had to create a flying planner using the graph data structure and the jGraphT library. The data set was given to us in the form of CSV files which we had to load into the data structure and use to validate and analyse data. In part A, we tried using the library with some airports as vertices and the cost of the flight as edge weight. And using the inbuilt class to find the least cost of the journey. In part A, we didn't load the data from the CSV which we will do in part B and use that data to evaluate it. The implementation of part B is explained below.

DESIGN CHOICES

In this coursework on implementing flying planner, the main aim was to use the graph data structure and use that graph to validate and analyse data. So, I have used the graph data structure to store the data of the airport and the flights. I have used HashSet to store the data from the CSV files so that there is no duplicate data and this set is used to populate the graph with the data in the files. In the graph, I have used vertices to store the airport details and edges to store the flight details. Also, I have used inbuilt classes like List and Set to store data and go through that lists and sets to analyse data. These were the main structures that I used in this coursework.

IMPLEMENTATION

Part B Methods

1. Populating the Data

In FlyingPlanner class, there are two overloaded functions named populate but the parameters are different. In the first method, we are passing the object of FlightsReader class which has the data from the CSV files. Whereas the second method takes the HashSet of string arrays of airports and the flights. Using these methods I am populating the graph data structures to store the data in the graph. I am taking, vertex as the Airport and edge as the Flight details. We are storing the objects of these classes respectively which helps us to store more data. For making my code more modular and clean, I have created two private functions, one for populating the Airport data and another for populating the flight data. The function used in populating the flights uses the airports function to find the airport code from the string code. The implementation of the function is explained below.

2. Converting string codes into object

As we know, a graph stores the object in the vertices as well as the edges. The type of vertices is Airport whereas the type of edge in the graph is Flight. Data in the CSV files is given in the form of the string, so to connect an edge to the vertices we had to know the airport code which we are evaluating in the airports' method. In the airports' method, code in the string type is passed and we are checking the code with the vertices in the graph. If the code exists in the graph, then we are returning the respective object of the Airport class. For, this we are storing the list of vertices into a Set of type Airport for which I am using the inbuilt class. Also, I am using the logic of "for each" loop to access the data in the Set. The same logic has been used in the flight function to find the object of Flight class which has the specific code.

3. Least Cost Function

This function is used to find the journey with the least cost from one airport to another. The implementation for this was easy but the place where it got tricky is when I had to add the edge weight which I didn't do in the populate functions. So I created a function that I used to add the edge weight for all the edges in the graph. After adding the weight of the edges, I simply created the object of the DijkstraShortestPath class and passed the weighted graph into the constructor. Now, I called the built-in function of the library to find the least cost of the journey. Before this, I am creating the object of the Journey class and storing the result of the leastHop function whose implementation is explained below.

4. Least Hop Function

This function is used to create the journey with the least hops using the graph and the parameters passed in the function. First, I am converting the string codes into objects of Airport class and then we use these objects to find the shortest path. For finding the shortest path we are using the function from the DijkstraShortestPath class and by using the result of this method we are creating the journey and setting the vertex list and the edge list in the journey. For making my code modular, I have created two functions for setting these in journey class in which I am passing the result of the shortest path in the function as an argument. After that, I am setting the value of the least hop using the setter function in Journey class.

5. Air Time functionality

I found this function tricky as we had to calculate the air time during the flight. First, I computed the time where arriving time was more than the departure time. After that, I used an if condition to check if the arrival time is less than the departure time, then I am calculating the time and then adding it to the total hours. For minutes I am calculating by getting it from the string and subtracting it. If the minutes are less than zero then I am decreasing the hour by one and adding sixty minutes to the total minutes to equalize it. In the end, this function returns the total time in minutes.

Part C Methods

1. Connecting Time and Total Time

In this part, we had to calculate the time between the connecting flights and the total time of the journey. For this, I used the implementation of the air time but, I changed the time for the calculation. I have taken the arrival time of the first flight and the departing time of the flight which is next to it. This function returns the time in total connecting time and then sets it in the journey. Also, I have implemented the total time of the journey using the air time and the connecting time.

2. Excluding The Airports

In this part, we had to find the journey which didn't include the excluded airports entered by the user. For implementing this, I had to change my code a lot, as we had to remove the vertices and the edges from the graph to exclude those airports as well as the flight from and to that airport. Also, when setting the weight of the edges I didn't have to add the weight of the edges of those airports. So, with the help of the built function of the library, first I removed the vertices of the airport and then the edges. For removing edges, I am getting the list of all the edges from the specific vertex and then removing all those edges using the library function. Now, for calculating the journey with the least stops, first I called the function which removes the vertices and the edges, and then I call the leastHop function which we used in Part B. But for the least cost, I had to change my code of part B function as I didn't want the code to be repeated, and also I wanted to make my code more modular. So, first I created one common function which finds the least cost taking the parameter whether we have airports to exclude or not, and then I am calling the function accordingly. If we don't have to exclude any airports, I am passing null instead of the airports' list. Also, I had to change the populate edge weight function, so that code doesn't get repeated. For this, I created a common function again for both of them and then set the weights. Finally, it returns the result as per the user's requirement and this feature would help the user to add the filter in their search of the journey.

SCREEN SHOTS

Part B Implementation

1. Starting interface

Here, We are asking the user to input the codes of the airport of departure and the arrival. Also, after taking the input I am asking the user's preference whether they want the cost to be minimal or the least stops during their journey.

```
========================================
==            Flying Planner          ==
========================================
Please enter the departure Airport code : NCL
Please enter the arrival Airport code   : NTL
Please choose your preference :
1. Least Cost
2. Least Stops
Enter Your Choice : 1
```

2. Details of the journey – With Least Cost

After the user has entered the details requested by the program, it shows the path of the journey which includes the airport as well as flight details. After these details are printed, then the total least cost and the total air time are printed.

```
Journey from NCL and NTL is :
------------------------------------------------------------------------
| Leg  | Leave           | At    | On      | Arrive          | At    |
------------------------------------------------------------------------
| 1    | Newcastle (NCL) | 19:18 | KL7893  | Amsterdam (AMS) | 20:04 |
| 2    | Amsterdam (AMS) | 07:47 | CX0831  | Hong Kong (HKG) | 17:02 |
| 3    | Hong Kong (HKG) | 07:48 | CX7100  | Brisbane (BNE)  | 14:27 |
| 4    | Brisbane (BNE)  | 16:28 | QF0640  | Newcastle (NTL) | 17:29 |
------------------------------------------------------------------------

Total Cost of the journey is     : £1035
Total Air Time of the journey is : 1061 minutes.
```

3. Details of the journey – With Least Hop

When the user selects the least stops mode, the program displays the details of the journey with the airport as well as flight details. As cost doesn't matter here, the result of the journey will be different here.

```
Journey from NCL and NTL is :
------------------------------------------------------------------------
| Leg  | Leave           | At    | On      | Arrive          | At    |
------------------------------------------------------------------------
| 1    | Newcastle (NCL) | 07:14 | EK1294  | Dubai (DXB)     | 13:17 |
| 2    | Dubai (DXB)     | 07:54 | EK1309  | Brisbane (BNE)  | 21:26 |
| 3    | Brisbane (BNE)  | 16:28 | QF0640  | Newcastle (NTL) | 17:29 |
------------------------------------------------------------------------

Total Cost of the journey is     : £1070
Total Air Time of the journey is : 1236 minutes.
```

Part C Implementation

1. Connecting Time and Total Time

In this screenshot, I am showing the journey details with the connecting time and the total time of the journey.

```
Journey from NCL and NTL is :
-------------------------------------------------------------------------
| Leg  | Leave           | At    | On      | Arrive          | At    |
-------------------------------------------------------------------------
| 1    | Newcastle (NCL) | 07:14 | EK1294  | Dubai (DXB)     | 13:17 |
| 2    | Dubai (DXB)     | 07:54 | EK1309  | Brisbane (BNE)  | 21:26 |
| 3    | Brisbane (BNE)  | 16:28 | QF0640  | Newcastle (NTL) | 17:29 |
-------------------------------------------------------------------------

Total Cost of the journey is     : £1070
Total Air Time of the journey is : 1236 minutes.
Total Connecting Time is         : 2259 minutes.
Total Time of the journey is     : 3495 minutes.
```

2. Excluding Airports in the search

In this screenshot, I am showing the journey from Edinburgh to Dubai but I am excluding one airport which is Paris. So, As a result, Paris won't come on the journey. And, the program prints the journey with all the details.

```
=========================================
==             Flying Planner          ==
=========================================
Please enter the departure Airport code : EDI
Please enter the arrival Airport code   : DXB
Please choose your preference :
1. Least Cost
2. Least Stops
Enter Your Choice : 2
Do you want to exclude any aiport(Y/N) :
Y
Please enter the codes of the airports you want to exclude :
Please enter * to stop entering the code
CDG
*
Thanks for entering the codes.
Journey from EDI and DXB is :
-------------------------------------------------------------------------
| Leg  | Leave           | At    | On      | Arrive          | At    |
-------------------------------------------------------------------------
| 1    | Edinburgh (EDI) | 16:53 | BA2448  | London (LGW)    | 17:43 |
| 2    | London (LGW)    | 12:53 | EK1354  | Dubai (DXB)     | 18:49 |
-------------------------------------------------------------------------

Total Cost of the journey is     : £414
Total Air Time of the journey is : 406 minutes.
Total Connecting Time is         : 1150 minutes.
Total Time of the journey is     : 1556 minutes.
```

3. Printing Error when user enters wrong input

In this screenshot, I am showing that when the user inputs the wrong departure or arrival airport, then I am throwing an error and then exiting the code. So the code won't proceed with the wrong airport codes.

```
========================================
==             Flying Planner         ==
========================================
Please enter the departure Airport code : EDL
Please enter the arrival Airport code   : DXB
Invalid Input!!
```

4. Printing Error when user enters wrong input

In this screenshot, I am showing that when the user inputs the wrong excluding airport, then I am printing an error and continue asking the user for the next airport code. But I am not adding that airport code to the list.

```
Do you want to exclude any aiport(Y/N) :
y
Please enter the codes of the airports you want to exclude :
Please enter * to stop entering the code
SFG
Invalid Input!!
123
Invalid Input!!
SG4
Invalid Input!!
EDI
DXB
*
Thanks for entering the codes.
```

KNOWN LIMITATIONS

For this code, known limitations can be that when the user enters Edinburgh (EDI) as the departing airport and the Dubai (DXB) as the arriving airport, and the least stops option is selected then the journey with the least cost is not printed. Because there are many options available for both airports and the least stops option is selected. The problem might be that we are not finding the all possible paths for this and then checking the cost. If we do this, then we can check the cost and print the journey with the least cost as well as the least stops which would be a better result. Also, the known limitations can be that there are some features given in the coursework specification which we have not implemented in the code. But, I would like to try implementing these features in the future for making this code more advanced and modular.

TEST DATA

Testing the code is the most important part of the development and I have used Test Driven Development(TDD) for this project which helped me to do the project. For testing the data, I first used the provided test, and then I created my test for testing my code. For creating my tests, I have used the data given in the coursework specifications and for some tests, I have referred to the results of some of my peers. I have written around ten tests to check my code in many different ways. The code of the departure airport and the arriving airport are NCL and NTL respectively. After writing the code, when I executed my tests they all got passed and I assume that it will pass for other airport codes also.