F28DA CW2 REPORT

Abstract

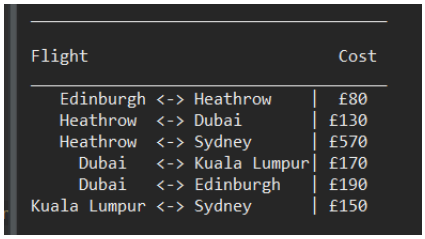
Report of my implementation of CW2 including Part A & B. Implementation, testing, errors and known limitations.

Callum Forsyth

H00275277

Part A:

In the figure below you can see the graph representing shortest flight and least cost connections. I used String.format, to get the graph layout that I wanted, I set up each flight with its cost as the edge weight as shown in the course work sheet, giving the layout shown below (Figure 1), each flight and the price as the edge weight.

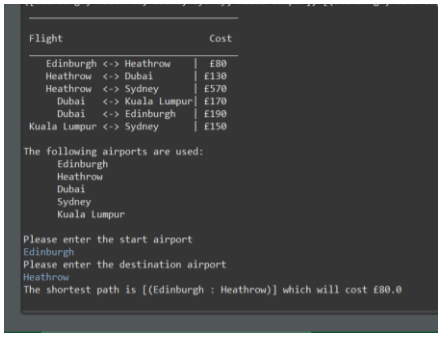


```
Flight                              Cost
-------------------------------------------
   Edinburgh <-> Heathrow     |   £80
   Heathrow  <-> Dubai        |  £130
   Heathrow  <-> Sydney       |  £570
     Dubai   <-> Kuala Lumpur |  £170
     Dubai   <-> Edinburgh    |  £190
Kuala Lumpur <-> Sydney       |  £150
```

Figure 1

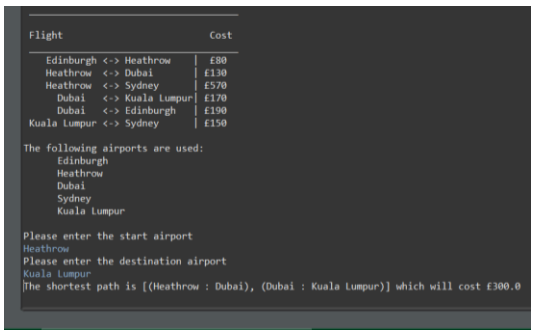
Test 1: Edinburgh to Heathrow

First, I tested the graph to check that the direct flights are showing the correct output, with the flight path and then the cost. When testing a direct flight from Edinburgh to Heathrow I was given the output shown below (Figure 2). This is the correct output for a direct flight. I then begun to test to find the least cost journey between 2 cities consisting of 1 or more direct flights



```
Flight                              Cost
-------------------------------------------
   Edinburgh <-> Heathrow     |   £80
   Heathrow  <-> Dubai        |  £130
   Heathrow  <-> Sydney       |  £570
     Dubai   <-> Kuala Lumpur |  £170
     Dubai   <-> Edinburgh    |  £190
Kuala Lumpur <-> Sydney       |  £150

The following airports are used:
     Edinburgh
     Heathrow
     Dubai
     Sydney
     Kuala Lumpur

Please enter the start airport
Edinburgh
Please enter the destination airport
Heathrow
The shortest path is [(Edinburgh : Heathrow)] which will cost £80.0
```

Figure 2

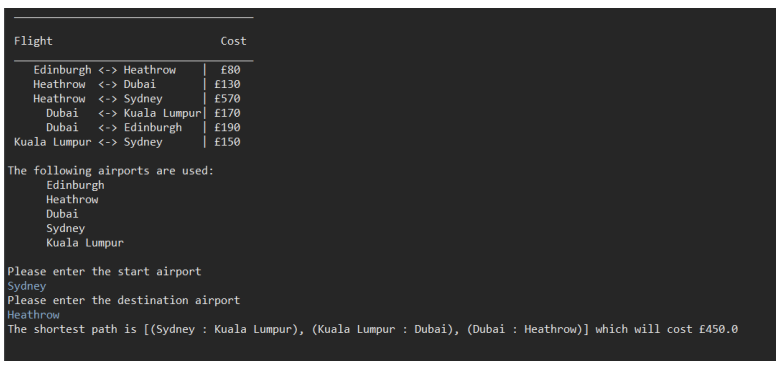


```
Flight                              Cost
-------------------------------------------
   Edinburgh <-> Heathrow     |   £80
   Heathrow  <-> Dubai        |  £130
   Heathrow  <-> Sydney       |  £570
     Dubai   <-> Kuala Lumpur |  £170
     Dubai   <-> Edinburgh    |  £190
Kuala Lumpur <-> Sydney       |  £150

The following airports are used:
     Edinburgh
     Heathrow
     Dubai
     Sydney
     Kuala Lumpur

Please enter the start airport
Heathrow
Please enter the destination airport
Kuala Lumpur
The shortest path is [(Heathrow : Dubai), (Dubai : Kuala Lumpur)] which will cost £300.0
```

Figure 3

I then begun to test my graph for a journey that was direct but could be done cheaper by adding connections, as the DijkstraShortestPath uses the edge weight, in this case the edge weight is the cost so it should be finding the cheapest flight not the one with the shortest path.

Sydney to Heathrow is a direct flight but cost's £570, as shown below (figure 4) you can take a longer route with more connecting flights but get a cheaper price, this concluded the testing on part A and gave the output I was expecting.



```
Flight                              Cost
-------------------------------------------
   Edinburgh <-> Heathrow     |   £80
   Heathrow  <-> Dubai        |  £130
   Heathrow  <-> Sydney       |  £570
     Dubai   <-> Kuala Lumpur |  £170
     Dubai   <-> Edinburgh    |  £190
Kuala Lumpur <-> Sydney       |  £150

The following airports are used:
     Edinburgh
     Heathrow
     Dubai
     Sydney
     Kuala Lumpur

Please enter the start airport
Sydney
Please enter the destination airport
Heathrow
The shortest path is [(Sydney : Kuala Lumpur), (Kuala Lumpur : Dubai), (Dubai : Heathrow)] which will cost £450.0
```

Figure 4

Part B:

In the graph below (figure 5) you can see the user interface for my part B showing a full user testing, I have shown the journey name, using the .getName() method in the IAirportPartB interface which returns the name of the airport when given its flight code. I have then shown the connections for the flight using the .getFlights() method which returns a list of flight codes in the journey (ie. The connections). I have then shown the total journey cost using the .totalCost() method which returns the total cost for the journey and finally I have shown the total time in the air using the .airTime() method which returns the total time in flight for the journey.

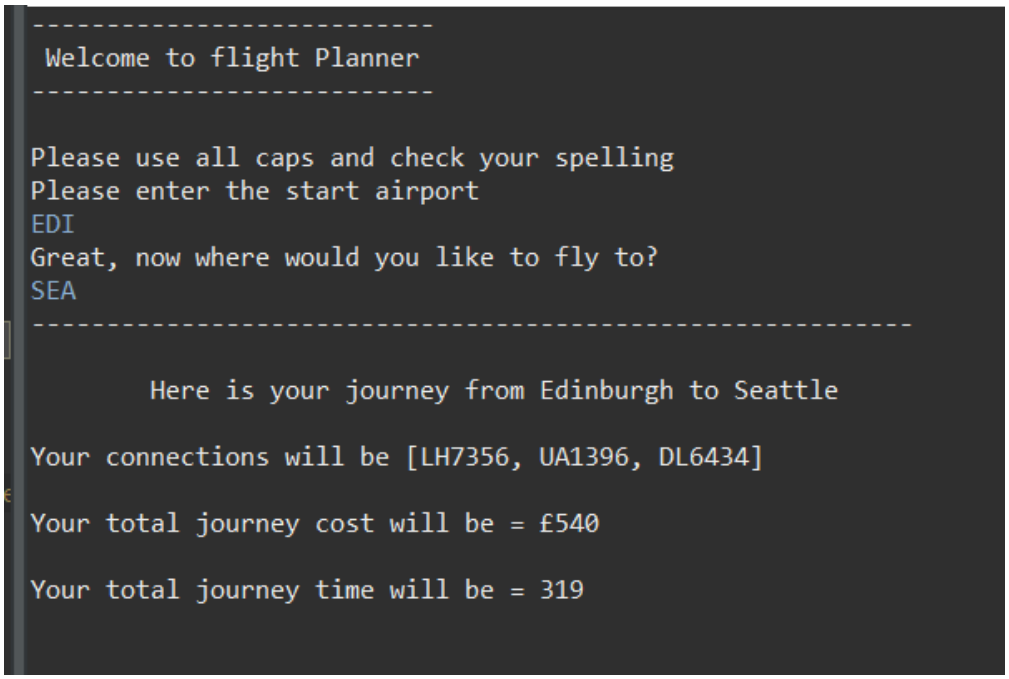


Figure 5

I have chosen a simple user interface (figure 6) as I wanted to keep the program as simple as possible without having to explain every part to the user beforehand. Using string.format to keep the information alongside each other. I think this looks better, as well as using string.format I also used \n to create a new line as well as \t to bring the information in without needing to use a large messy print statement.

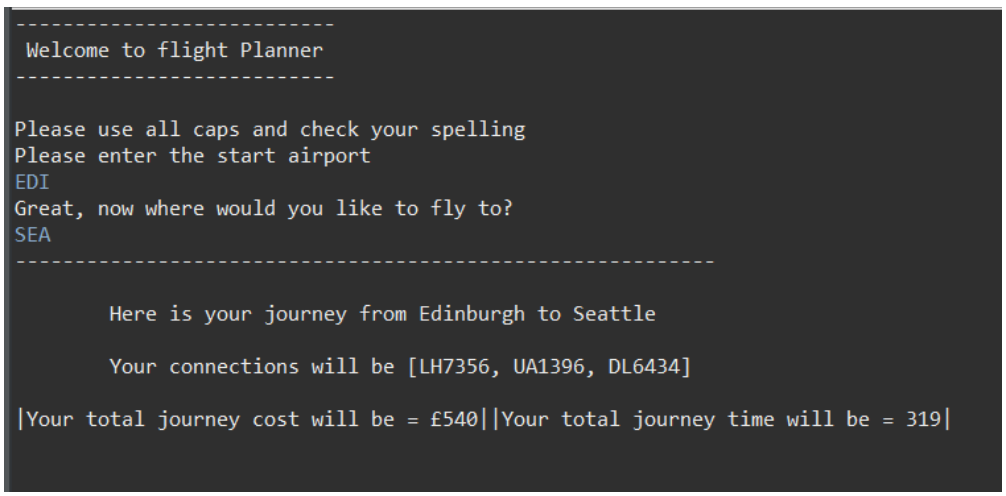


Figure 6

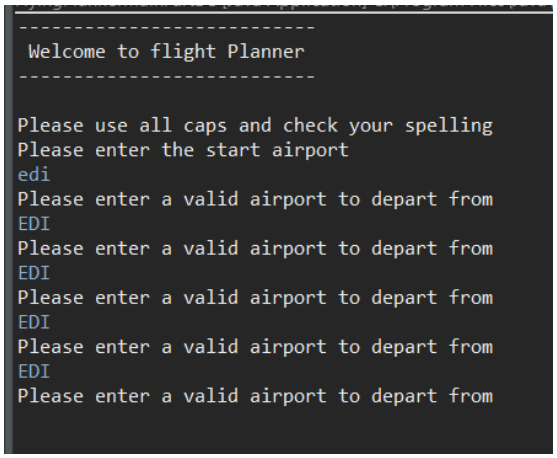
Implementation:

I first started by implementing the helper classes, I started with Airport.java as I thought this was the easiest, I got the hashset for airport code and airport name from checking the csv file as well as the FlightsReader.java file. I then moved on to implementing the Flight.java file, again using the hashset from the FlightReader.java file and the csv file. I then moved on to Journey.java file and began implementing the methods one by one, I have extensively commented my code line by line as well as java doc comments so I will avoid talking too much about my exact coding as it is a lot easier to explain when looking at the code. I then moved on to the FlyingPlanner.java class and finally begun to implement my user interface and finished testing. I tested thoroughly throughout my implementation using the provided tests, my own tests and the console to check the user interface part of the program.

I have written my own tests in the FlyingPlannerTest.java file as well as writing my own custom test which uses a custom graph made within the test rather than the graph we have used throughout the program.

Errors:

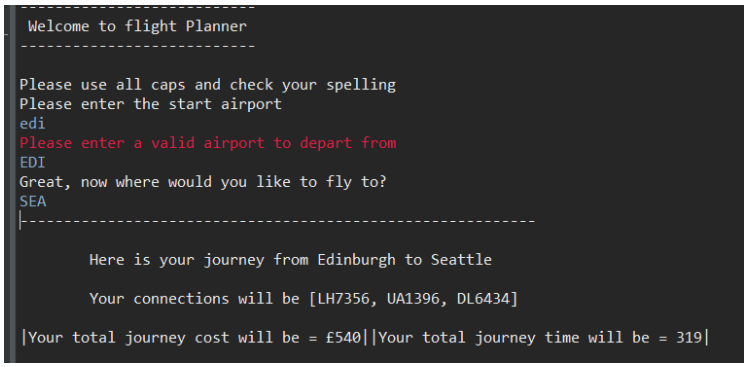
There are some errors that I know of within my program, one is regarding the user input on the interface. If the user inputs an error when trying to put their airport in the code gets stuck in an infinite loop and continues to ask the user to put a new airport in, even if the airport they put in is correct the second time.




```
22          try {
23
24              fi.populate(new FlightsReader());
25
26              Scanner scan = new Scanner(System.in);
27
28              System.out.println("--------------------------");
29              System.out.println(" Welcome to flight Planner ");
30              System.out.println("--------------------------");
31              System.out.println();
32              System.out.println("Please use all caps and check your spelling");
33              System.out.println("Please enter the start airport");
34
35              String startAirport = scan.nextLine();
36              Airport airStart = fi.airport(startAirport);
37              while(fi.graph.containsVertex(airStart) == false) {
38                  System.out.println("Please enter a valid airport to depart from");
39                  startAirport = scan.nextLine();
40              }
41
42
```

Fixed

I have now fixed this error, I realised I was only updating the string ‘startAirport’ and not the airport object, this meant the program would keep taking in the first input even if the user inputted the correct airport code a second time.



Know limitations:

There are some limitations that I know of, firstly the program doesn’t output the exact times of each flight, this is a design problem but not essential for the program implementation. To improve the program, I would include the individual times for each flight, both arrival and departure time.

Another known limitation is the program does not print out the leg of the trip that the user is on, this can be shown through the order of the connections but I think it is more user friendly to have each leg printed out in a line with the airport, the flight code and the time of departure and arrival.

An example of a model user interface is shown below.

```
Journey for Newcastle (NCL) to Newcastle (NTL)
Leg Leave              At   On      Arrive             At
1   Newcastle (NCL)    1918 KL7893  Amsterdam (AMS)   2004
2   Amsterdam (AMS)    0747 CX0831  Hong Kong (HKG)   1702
3   Hong Kong (HKG)    0748 CX7100  Brisbane (BNE)    1427
4   Brisbane (BNE)     1628 QF0640  Newcastle (NTL)   1729
Total Journey Cost      = £1035
Total Time in the Air = 1061
```