Object Oriented Programming

Name: Seamus Meehan

Student No.: D16124533

1. BRIEF DESCRIPTION

1.1 Instructions for Running the Code

The program can be running the *findBestRouteCodeInput.py* with arguments of five space separated codes followed by a aircraft code. A valid example is:

python findBestRouteCodeInput.py DUB LHR SFO JFK EIN 747

This will print output to the terminal showing the most fuel efficient route along with the fuel cost for that route,

It can also be run using *findBestRouteCSVInput.py*. This file will by default look in the current directory for a file called *testroutes.csv*. It will process each itinerary in the file. The itineraries should be on single lines and be comma-separated. A valid line in this file looks like:

DUB,LHR,SFO,JFK,EIN,747

We can also specify to read a non default CSV file by passing it as an argument on the command line. Like so:

python findBestRouteCSVInput.py adifferentCSV.csv

The CSV method will output a comma-separated string to a file called *bestroutes.csv*. This file will look like for the example above:

DUB,JFK,SFO,EIN,LHR,DUB,33655.65

where the airports are ordered in the most efficient route and the number at the end is total fuel cost. The terminal will also display how this number was arrived at by displaying how much fuel was bought in each airport.

It should be noted here that this fuel number is to have a full tank at the end of the route. The reasons for this will be discussed later when we consider the *getOptimumStrategy()* method of the Aircraft class.

1.2 Included Functionality

Our program includes functionality to work on the command line and via CSV input. We calculate the most fuel efficient route and return it to the user.

1.3 Missing Functionality

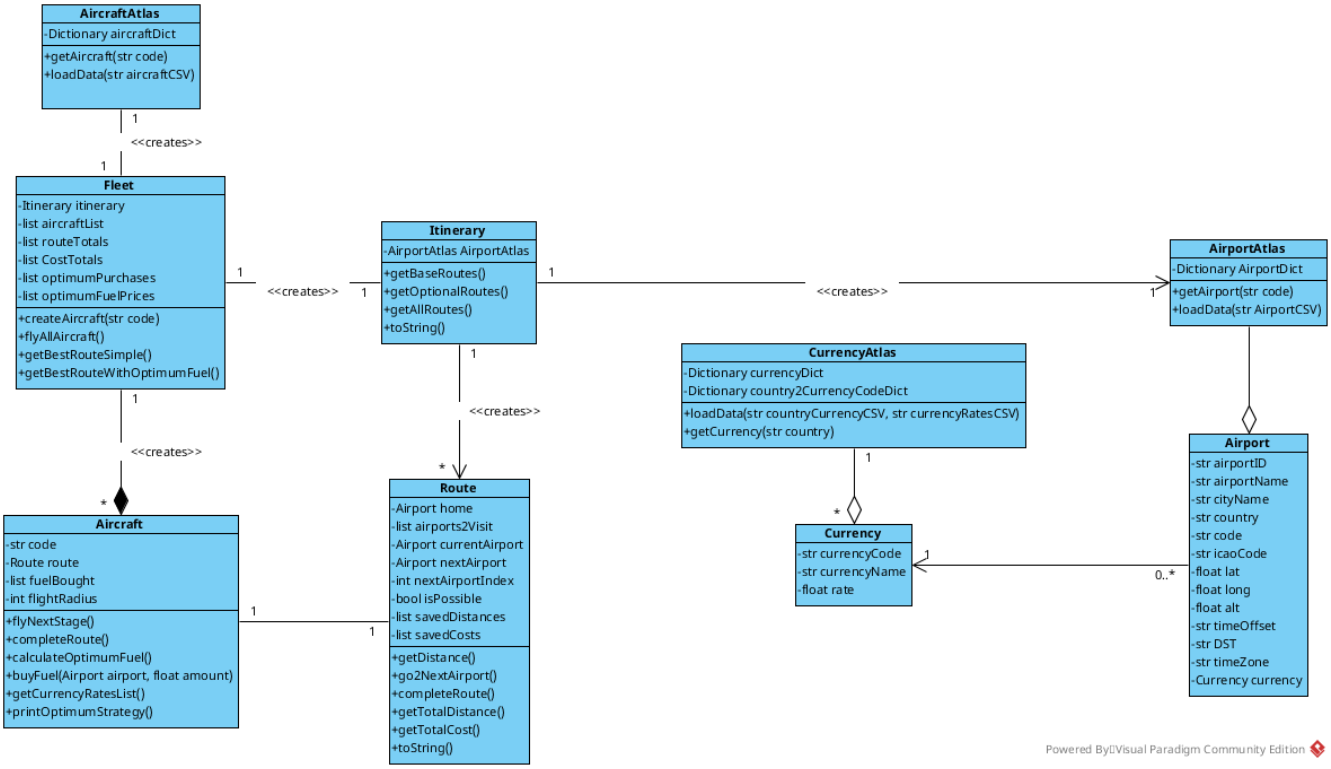
The user interface is absent as there was not enough time to complete a working version. While the project was designed to have a Model-View-Controller structure the View remains absent.

Due to time constraints it was decided to focus on the actual functionality of the program, calculating the most fuel efficient path through a list of airports. The UI was considered “nice to have” but not required for a fully functional program.

Proper encapsulation of classes is also missing. Most classes just directly access the attributes of other classes

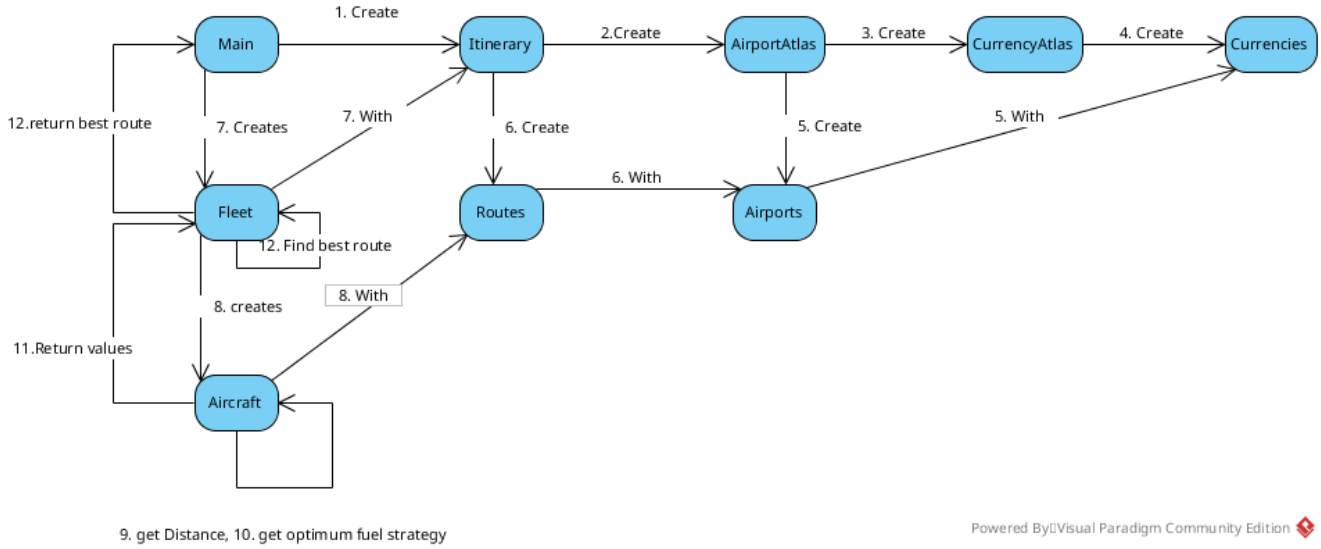
2 Design

2.1 Class Diagram



A class diagram was developed to model the problem. It was iterated upon through development and the diagram above was arrived at. Classes were considered to have a single responsibility and their interactions were designed to minimize file I/O. We do not want to create an airport atlas every time we want to create a route.

2.1 Main Execution



The code executes as above with more detailed steps below:

1. Main method is called with appropriate arguments.

2. Main then deals with the arguments to obtain a valid route

3. Main then instantiates a itinerary

4. Itinerary checks if it is a valid route (no duplicates)

5. Itinerary generates all permutations of the itinerary

6. Itinerary generates an airport atlas

7. The AirportAtlas generates airport objects from a file

8. It creates a currency atlas which creates currency objects.

9. These currency objects are then given to airports as an attribute

10. Itinerary uses it's permutations to create route objects which are a list of airport objects

11. Main then creates a Fleet object by passing the itinerary object to it.

12. The fleet object creates an aircraft atlas in order to enable creation of aircraft objects

13. The fleet object then creates new aircraft objects for each route. It passes a route, the plane code along with the max flight range for that code obtained from the atlas to the aircraft constructor.

13. The fleet object then tells the aircraft to complete their routes.

14. Aircraft store distances travelled for each leg in an attribute.

15. The fleet object then tells the aircraft to calculate their most efficient refuelling pattern

16. The aircraft do this by considering all future airports and attempt to buy fuel to cover distance until a cheaper airport is arrived at. Otherwise they buy the minimum fuel. If they arrive at an airport with no airport later in the route with a cheaper price. Finally to make it fair they fill their tank when they arrive home.

17. The fleet object then sorts through the aircraft to find which has the cheapest optimal route.

18. The fleet object will then return the cheapest route to the main function

19. The main function will then decide what to do with the returned route based on the input method of the user.

3 Test Strategy

The program was tested by writing suites of unit tests for the main classes. This was consider to be the aircraft and route classes. For these unit tests all the methods in the aircraft were tested. These in turn call the route methods so we are testing them too. For the aircraft tests we use a placeholder route which we generate and use for all aircraft generated. These tests are contained in the file *aircrafttest.py*.

For the system wide tests we input multiple routes likely to cause errors into the testroutes.csv file and we run each of these itineraries through. Examples of test routes used and what they are likely to break are as follows:

DUB,LHR,SFO,JFK,EIN,747: Valid route

DUB,SFO,LHR,JFK,EIN,747: Should return the same as above

DUB,SFO,LHR,JFK,EIN,DC8: 300 planes fail

DUB,LHR,SFO,JFK,EIN,F50: no valid routes

DUB,LHR,SFO,JFK,747: too short

DUB,LHR,SFO,747: too short

DUB,LHR,SFO,JFK,EIN,PAR,747: too long

dub,LHR,SFO,JFK,EIN,747: lower case airport

DUB,EIN,BFS,LHR,PAR,dc8: lower case plane