Object-Orientated Software Development

C13396956
James McCarthy

1. Function Specification

My program uses a relatively simple algorithm to calculate the lowest cost of a given list of airports. Once a home airport has been selected the order of the given list doesn't matter, the program will generate the route that gives the lowest cost. It can also give the cost of the shortest route. Often times this is more expensive than the shortest route. My program uses a very simple GUI for taking user input and displaying output or any errors that occur.

The main product features implemented are the GUI, the lowest cost calculation, shortest distance cost calculation and the breakdown of the flight (i.e cost/distance of each leg, fuel purchased at each airport). The GUI is very simple but is necessary not only to help the user enter the data and read the output but it makes error handling much easier as the input given to the program can be controlled. The lowest cost calculation is the main feature of the program so was a necessity. The shortest distance cost calculation was added so the user can compare the routes. Hypothetically if the user needed to transport good as quickly as possible and wanted to know if it was reasonable to choose the shortest distance, therefore the fastest, route over the cheapest, they could compare the costs of each trip and make an informed decision. Then the breakdown of the flight was added to help the user get a sense of how the cost was calculated.

The program is used by way of a GUI. Data is entered in the relevant sections and then the cost of both routes can be calculated using the relevant buttons. It was felt that this would make the user's experience easier and would help with error handling.

2. Design

1.1 Assumptions

There are number of assumptions made for this program:

- The initial fuel an aircraft is 0.
- Each aircraft has a minimum amount of fuel it must have at the end of each leg, set at 5% of the maximum fuel capacity of said aircraft. This plays into the algorithm that calculates the cost of the route
- The radius of the earth is assumed to be 6371 km.

1.2 Inputs/Outputs

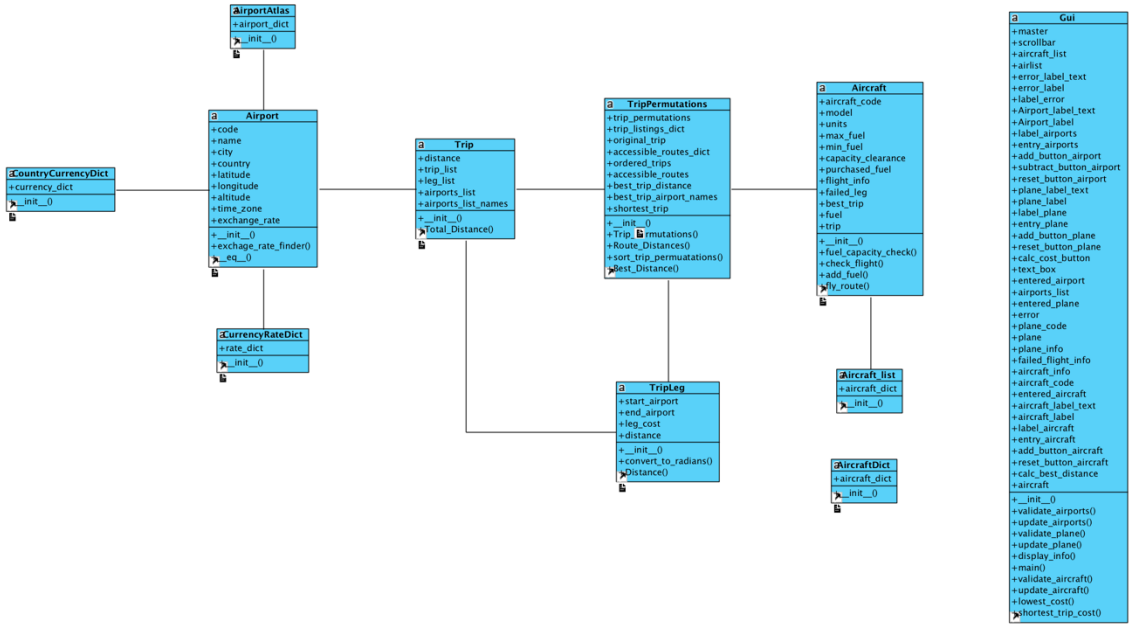
All inputs are passed to the program the implemented GUI. The first inputs are airport codes, they are passed as strings to the program that then reads them into a function that finds the relevant information from a file and creates an Airport object with this relevant information.

The second, and last input, is an aircraft code, which is handled in a similar manner as the airport codes, creating an Aircraft object with the relevant information.

The GUI displays all output of the program. It displays input errors in its own section and in a textbox, that displays the breakdown of the cost of the route, it displays any errors the program comes up with. For example, the route not being accessible with the chosen aircraft.

Once the User has entered all the correct information they can press one of two buttons. One to calculate the cost of the cheapest route and one to calculate the cost of the shortest route. The output displayed from these calculations is shown in the textbox and the bottom of the GUI.

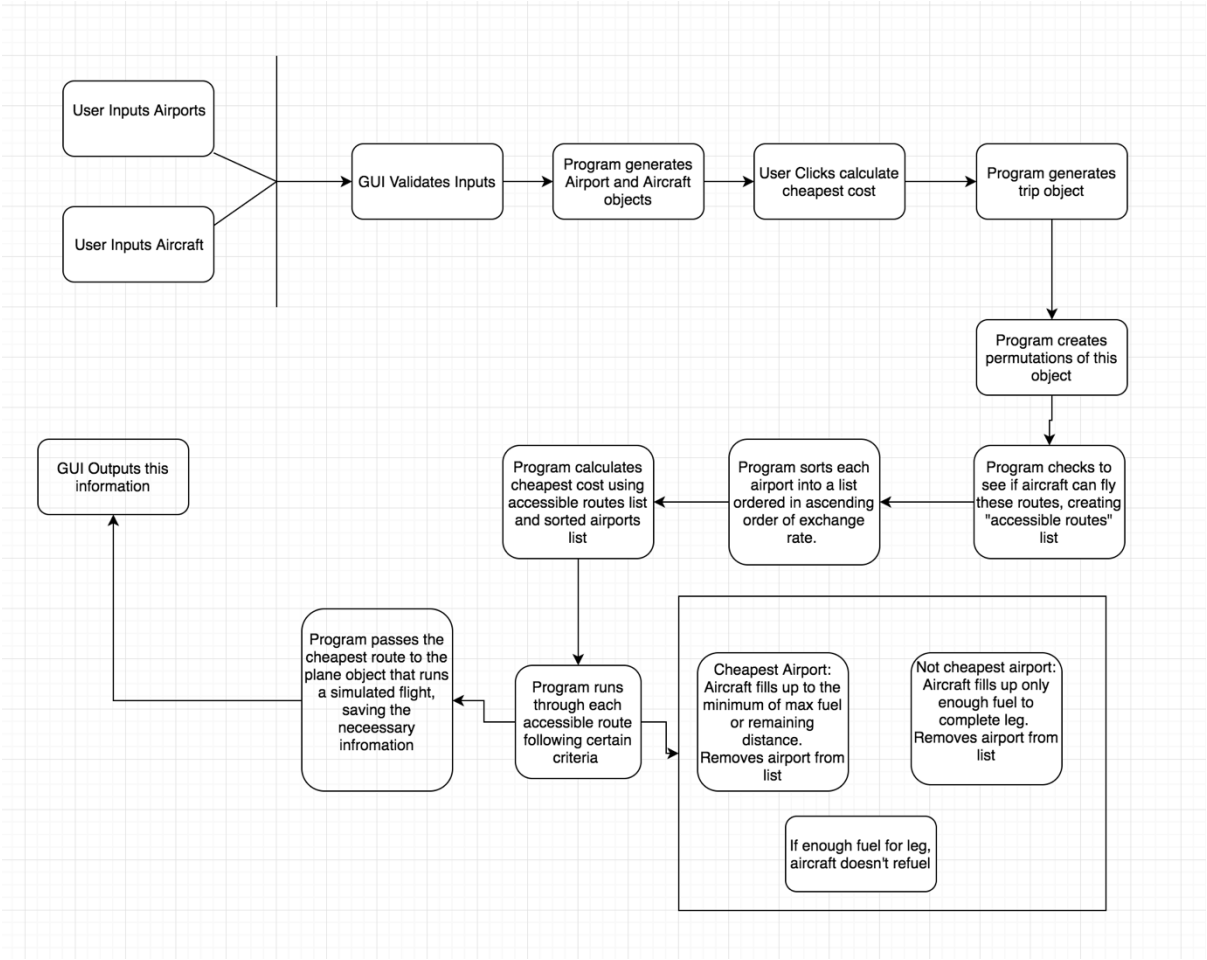
1.3 Class Diagram



The program consists of a number of classes. While there are functions that are outside of any class, these functions still rely on at least one of these classes. The aircraft_list, CurrencyRateDict, CountryCu and AirportAtlas are all classes that are used only to fetch the required information from the given CSV files and pass these to the objects. The Aircraft class creates aircraft objects. It mostly stands on its own but is used by the Trip Permutations class to create the accessible routes list. The Airport class creates airport objects and is required to create Trip, TripLeg and TripPermuations objects.

The GUI makes use of all of these classes.

1.4 Block Diagram of cheapest cost algorithm



The algorithm follows a fairly simple flow. The GUI takes input from the user, validates it to make sure the input is correct. It then passes this information to the program that generates the relevant objects. Then once the calculate cheapest cost button is pressed the program then creates a Trip object of the given list. It then creates permutations of this trip object by rearranging the order of the airports, excluding the home airport. The program checks these permutations to make sure the chosen plane can fly the route, creating an accessible routes list. The program then sorts the airports into a list ordered in ascending order of exchange rate. Using the accessible routes and the sorted airports the program calculates the cheapest route, running through Trip leg of each trip permutation and checking which leads to the lowest total cost using the following criteria:

- At the cheapest airport in the list the aircraft fills up to the minimum of max fuel or remaining distance. It then removes the airport from the list.

- If the airport is not the cheapest airport it will fill up only enough to finish the current leg
- If the aircraft has enough fuel reach the next airport with-out filling up it won't fill up. (This could be made more complex/accurate by checking to see if the aircraft will need to fill up at the airport after it finishes its current leg and to check if either the current airport or the next is cheaper to fill up at.)

Once the program finds the cheapest route it then passes this route to a function in the aircraft object that simply simulates a flight using these criteria and then saves all necessary information. The GUI then outputs this information.

3. Testing

No formal testing, in the form of test suites, has taken place during the design of this project. The testing was mainly done by giving the program different inputs, routes that can't be completed, incorrect information etc. The main error handling is done by the GUI by only accepting the correct information.