Table of Contents

[1. Function Specification 2](#_Toc481349687)

[2. Design 3](#_Toc481349688)

[2.1 Class Diagram 4](#_Toc481349689)

[2.2 Block Diagrams 5](#_Toc481349690)

[2.2.1 Find Route with possible stopover algorithm - flowchart 5](#_Toc481349691)

[2.2.2 Find Route saving fuel algorithm - flowchart 7](#_Toc481349692)

[3. Test 9](#_Toc481349693)

# Function Specification

The project develops a fuel management software system to manage a fuel purchasing strategy. The software has been requested to find the most economic route for a given airport among a list of five airports. The route will start and end to the same airport and can occasionally visit another airport twice. Therefore the route will be composed by five or maximum six trips.

The program gathers csv input files to acquire the necessary data and for each file creates class where the data is stored in key-value pair data structures (e.g. dictionary). Then it creates classes which represent a template for each single unit included in the csv files (i.e. Airport, Currency classes). After gathering all the necessary data the program develops some particular feature for each classes that would be essential to the purpose. For instance, it implements in the AirportAtlas class methods to calculate the great circle distance on sphere between two airports from their latitudes and longitudes using the Haversine formula.

|  |  |
| --- | --- |
| a = sin²(Δφ/2) + cos φ1 ⋅ cos φ2 ⋅ sin²(Δλ/2)  c = 2 ⋅ atan2( √a, √(1−a) )  d = R ⋅ c | φ = latitude  λ = longitude  a = square of half the cord length between the two points  c = angular distance in radians  R = radius of Earth  d = distance |

Therefore the software develops three algorithms to find the most economic route. These three algorithms are then comparing to find the cheapest route among their solutions for the given list of airports and aircraft.

The choice to have three algorithms instead of one is due to the difficult to find a correct algorithm that could satisfy any conditions. The three algorithms are briefly described below:

Shortest Route Algorithm. It’s the simple one that finds the best route considering always the minimum distance between airports. Basically from home airport the aircraft will fly to the closest airport and from this airport to its closest one and so on. The aircraft will be filled it up to the amount of fuel needed to cover the distance.

Route with possible stopover. This route has been developed trying to find if there was a possibility that stopping twice in one of the given airports would be cheaper than flying only once to each airports. In this algorithm the airplane will try to fly to the airport at its maximum distance, but if there is any airport in the list which flying through it would be cheaper than flying directly to the airport at maximum distance the algorithm will follow this condition. Also in this algorithm, it will be added to the aircraft only the needed fuel to fly to its next destination.

Route saving fuel. This algorithm calculates the cheapest route try to fill it up the aircraft tank when the cheapest airport is visited. As filling the tank up to max fuel capacity could be very expensive in the case of a short route, the tank will be filled it up to a value which is calculated considering a sum of maximum distances between airports. This will avoid waste of fuel.

For a given input data the software will find the cheapest route based on these algorithms.

The software must be used through a Graphic User Interface (GUI) which displays inputs and outputs on same window.

On the top part of the window there are three comboboxes, six text-boxes and three buttons.

The three comboboxes allow the user to select first the aircraft from a list of predefined aircrafts, then select airport code based on country where airport is located.

The six text-boxes are there to allocate the selected inputs which will need to run the application and find the cheapest route.

The three buttons are, one for running the application and therefore display the outputs, the second to reset the list of selected airport to be able to insert a new selection and last one to save route found on CSV file.

The bottom part of the window displays several text boxes which represent the overall route. Each row shows the details about each trip, including the departure and arrival airports and the covered distance (in Km), used fuel (in litre) and cost of fuel (in Euro) which it’s affected by different currency rates.

At the very bottom and on right side, the application shows the total amount of kilometres, euros and litres used for the whole route.

I previously specified that the application must be run by GUI because it has been decided to handle all the exceptions from the GUI. In fact, instead handling the exceptions in the actual code, some custom exceptions are raised to then be handled in the GUI and therefore allow the user to be informed about the type of error.

# Design

Any CSV file imported in the application have been stored in a dictionary data structure inside a class. Therefore for aircraft.csv there is a class called AircraftDictionary which open the file and store the data in a key-value pair data structure where the key is the aircraft model and value is an object of class Aircraft. Basically each row of CSV file represents an object of the class associated to that file (i.e. Aircraft object). The same concept is applied to the other files that contain airports data or country currencies data. For currency rates CSV file, the associated class dictionary called CurrencyRatesDictionary is created but it has not been created the object class as the file contains only two attributes.

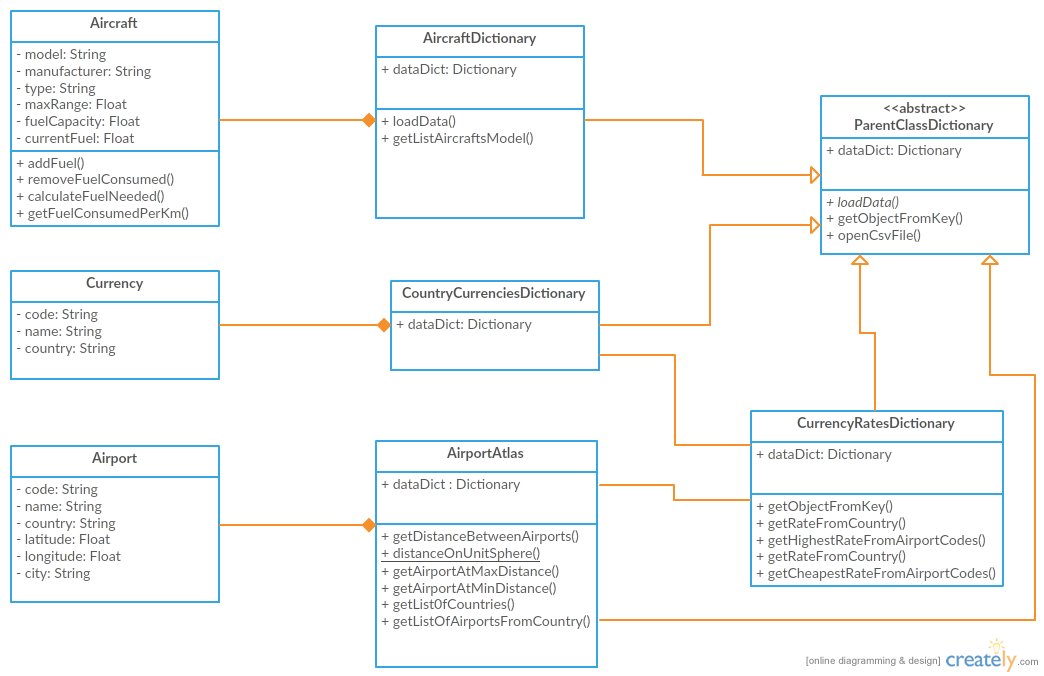
Some modifications to the original input have been applied to manage in a better way the data.

Aircraft CSV file has been modified and it has been added a column with the max fuel capacity of each aircraft. To make this new property consistent with max range, the latter has been modified for some aircrafts. As a consequence of these modifications the unit property has been switched to metric for every aircraft.

Regarding the currency rates, it has been considered the rate equal to 1 in case no currency rate is found for a given country.

In conclusion all the outputs are measured in kilometres for distances, euro for currency and litres for fuel volume.

## Class Diagram



The class diagram shows the overall static structure of the system. There is an abstract class called ParentClassDictionary (unfortunately the tool used to draw the diagram did not allow to set Italic font for name class as it supposed to be for abstract classes) which allow to reuse some methods that otherwise will be repeated in many dictionary classes. In fact all dictionary classes carry out the same functionalities, such as open CSV file and get the object (value in dictionary) from a selected attribute (key in dictionary). Therefore it has been applied one of the principles of object oriented paradigm which is the Inheritance.

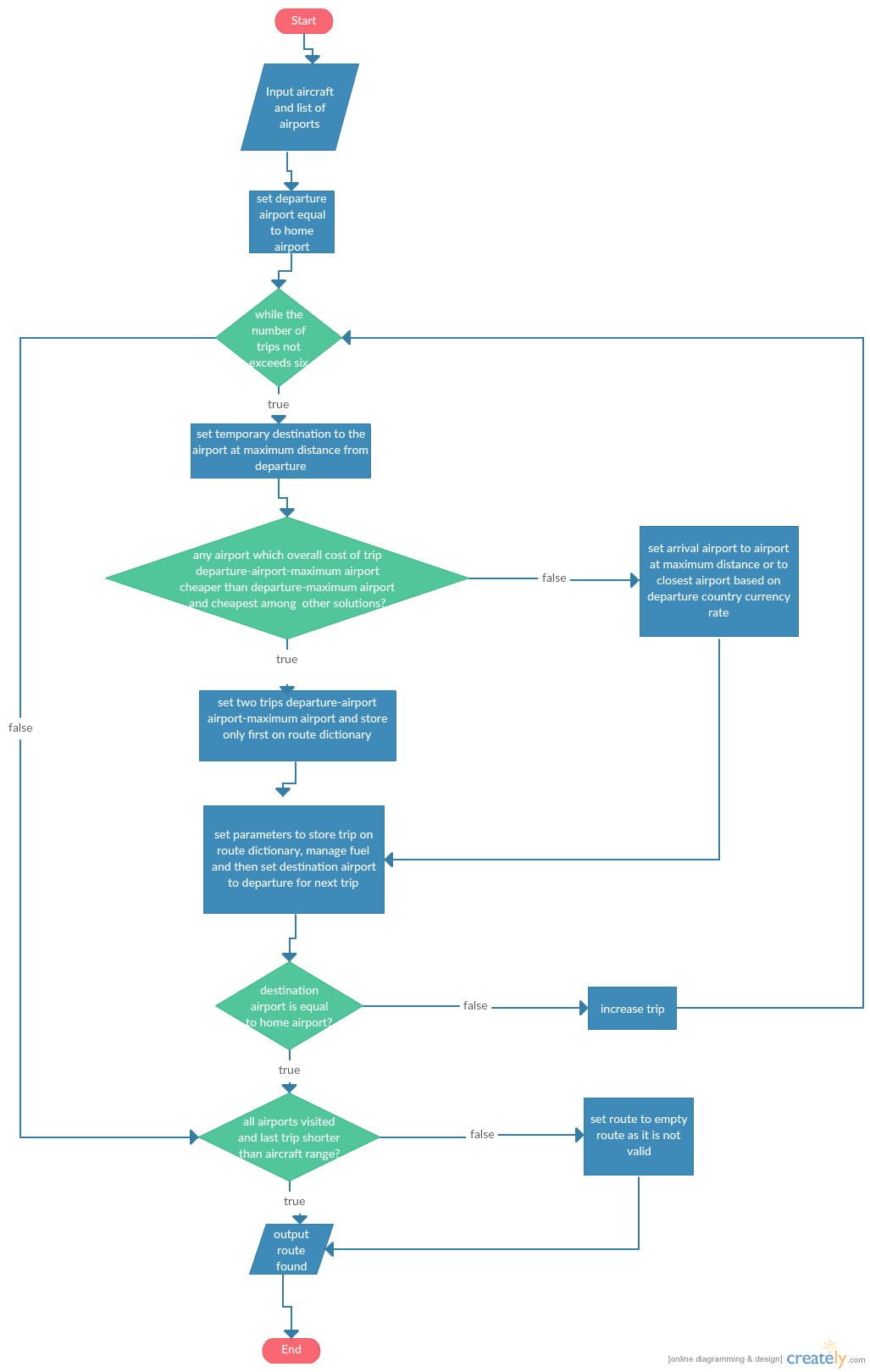
Furthermore the diagram shows some relationships between classes. All dictionary classes are composed by a set of associated object class (e.g. AirportAtlas is composed by several Airport objects). Also CurrencyRatesDictionary has relationships to either AirportAtlas or CountryCurrencyDictionary.

## 2.2 Block Diagrams

### 2.2.1 Find Route with possible stopover algorithm - flowchart

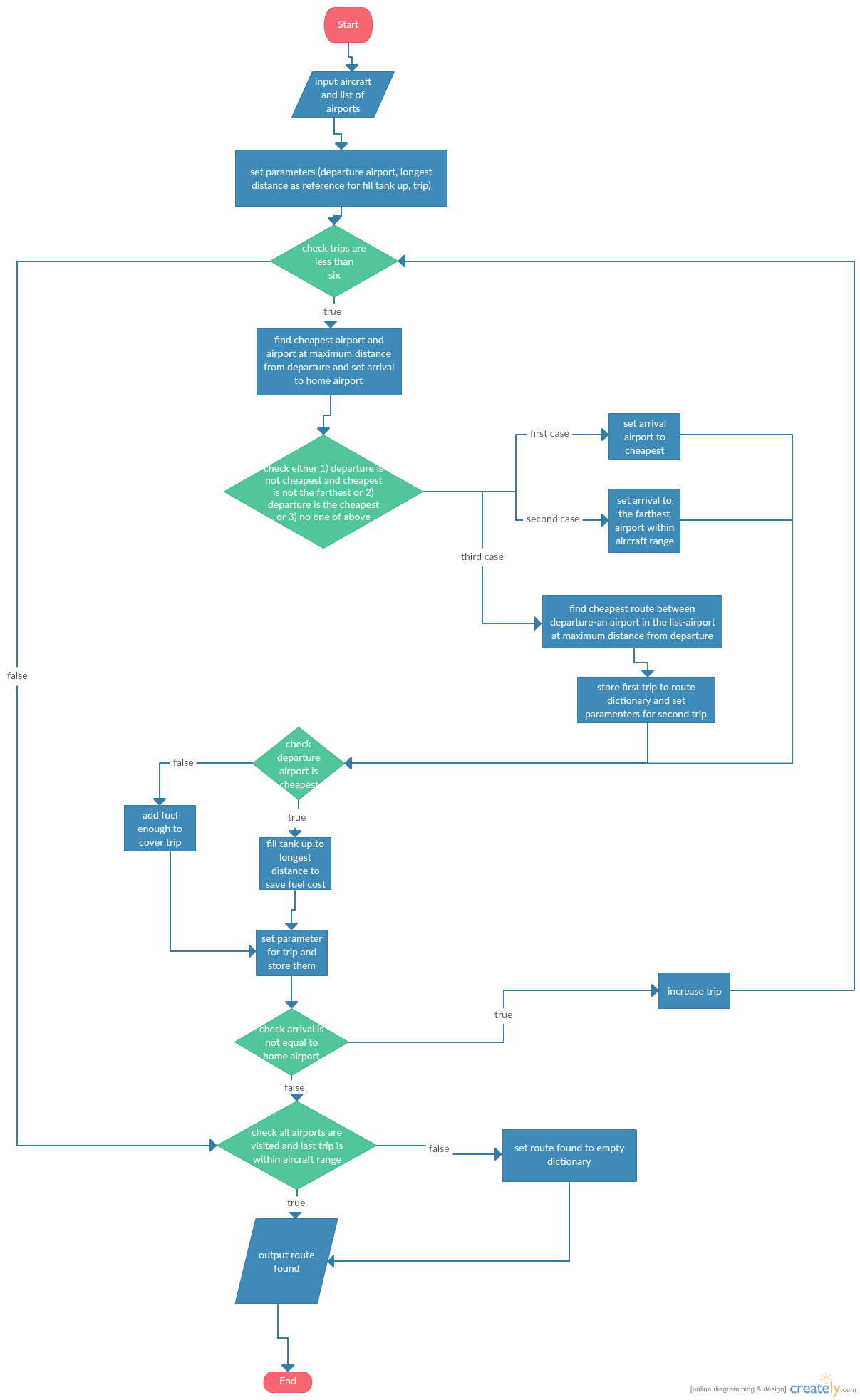
The diagram shows the flow of the algorithm. It starts gathering input data (aircraft and list of airports) and then it will begin to search for a route until the number of trips is not greater than six.

Then it finds the airport at maximum distance from departure airport. If there is any airport in the list which the route, considering itself as stopover, is cheapest than flying straight to airport at max distance, then this route will be selected and store it. Otherwise, it will decide if flying to max distance airport or to a closest airport based on the departure airport currency rates. The algorithm will keep search and store trips in the same way and finally it will give as output the found route unless the route found is not valid, which means the aircraft has not enough range to reach any airports.



### 2.2.2 Find Route saving fuel algorithm - flowchart

This algorithm will include also the fuel condition, filling the aircraft tank as much as possible (it cannot be the max fuel capacity as in some cases it is much greater than the needed volume of fuel) at the cheapest airports. Therefore the scope of this algorithm is to fly always towards the cheapest airport. This condition it would not be applied if the departure airport is already the cheapest or the cheapest airport is the farthest from the departure airport, as in this case it would not be so convenient to fly so distant and to add a great volume of fuel at higher price.



# Test

Tests have been performed on any classes involved in the system. There are only few tests for airport class, aircraft class and currency as they do not have too many functionalities.

The most tested components of the system are AirportAtlas, CurrencyRatesDictionary and the route algorithms file that contain not only the three algorithms but also supporting methods.

For instance, in airport atlas I tested the distance between some airports, then I checked if the distance between same airport is equal to zero . While developing the application some airport gave me an error for measuring the distance to themselves. Therefore the great circle distance formula has been changed and the test has been applied to all the airports inside the airport.csv file and the test has been passed for all airports. Then I tested if some invalid input is given, either for airport codes or for the csv file itself.

For route algorithm file, I tested the main supporting methods and the three algorithms. I do not consider very appropriate the tests on three algorithms as it is not easy to predict the expected result of them, especially for the stopover and saving fuel algorithms.

Finally the AirportAtlas, CurrencyRatesDictionary and route algorithms have been grouped into a test suite to run all the test together.