For this project, I created 6 classes; Route, Airport, Airline, Node, FileReader and FindRoute. The project required one to read from the route, airport and airline csv files and the records from these files were made into their respective objects. The input file was split by the new line and the source and destination were saved into variables. All this reading was done in the FileReader class by passing the files as parameters. The objects were then added to a hashmap to make search and retrieval among the different classes easier.

The node class creates the representations of the routes and airports that will make up the search tree. The state of the node is represented by an Airport object while its parent is the whole node. For the action, it records the Route object that led to it and the path cost attached is the number of flights that has been taken. In the FindRoute class, this maps out the search algorithm and the steps taking to record the information within the output file. The goal state is any airport that is found within the city and country of the destination that the user wants to get to and that is the concept being caught in the isGoal method.

In implementing the solution, I understood that any airport coming from the source airport could possibly be the root node, or the node the user would travel on. However, I was at a loss at how I could ensure that my algorithm searched for the goal from every “possible source” airport. By adding all the nodes to the frontier from the beginning, I was able to make sure that the algorithm searched through each airport to retrieve the solution. I made use of the rubber ducking technique for most of the project which helped me work through my thinking and translate that into code. Due to the many bugs that came up as a result of semantic errors usually, I learnt how to make use of the debug factor in IntelliJ which helped me step through my code and visualize what was actually happening.