PROJECT HIGH LEVEL ARCHITECTURE

The approach to this project was by modelling a graph from the given data set with the airports as the nodes or vertices of the graph and the routes as the edges that connects the airports.

After the forming the graph, **A*** search algorithm is use to search for an optimal path between a given city and a destination city. The **A* search algorithm**, builds on the principles of Dijkstra's shortest path algorithm to provide a faster solution when faced with the problem of finding the shortest path between two nodes. It achieves this by introducing a heuristic element to help decide the next node to consider as it moves along the path. I used priority queue with a comparator that that assigns priority to Airports on the frontier with the smallest distance to the given destination by using the haversine approach to compute the distances using longitude and latitude. The heuristic approach was to pre-estimate the distance from a generated child airport or vertex to the destination and compare with other nodes generated to decide which to process at a given a given instance of the search.

A breadth first search algorithm was also implemented to help make a quick search for the route that leads from the given airport city to the defined destination airport city.

Classes and their methods aside accessors and mutators

- **Route Class:**
  - *calRouteCost:* takes the sourceairport and destinationairport coordinates of a route and calculates the cost of, cost here is taken as the distance, the route using the Haversine formula
  - *toString :* overrides the object toString method and returns a string representation of a Route instance
- **Airport Class:**
  - *toStrig:* overrides the object toString method and returns a string representation of a Route instance
  - *equals:* compares a given object to the other and returns a boolean value
- **Airline Class:**
  - *No other methods aside setters and mutators*
- **FlightRouteSearcher Class:**
  - *destinationTest:* takes an airport parameter and compares to see if it is the destination of a given instance of FligthRoutSearcher and returns a Boolean value.
- **Vertex Class:**
  - ***It implements the comparable java interface***
  - *compareTo:* It compares the distance of the current vertex to the distance of the vertex passed in as a parameter and returns *0* if there are equal, returns *1* if the formal is greater than the latter, returns -1 otherwise.
  - *toString:* It overrides the object toString method and returns a string representation
  - *getPathToDestination:* backtracks to compute for all routes that lead to a found destination and returns a returns a deque of routes that are the shortest path from the source to the destination
  - *writeOutputPathToFile:* takes the name of the inputFile and writes the output of the *getPathToDestination* to a file with the name of the inputFile concat "_output".

- **PathFinder Class:**
  - *readInputFile:* It reads the airports.csv file and sets the start and end cities and countries .
  - *getStartEndAirport:* It reads the airports.csv file and array of two objects of type Airport (Airport startAirport, Airport endAirport)
  - *generateAirportByIATAC:* It takes airportId and IATACode as parameters and it reads a csv file and returns an object of type Airport if the airportId and IATACode match specific airport details in the csv file
  - *getAirportNeighbors:* takes an airport object and generates all the nieghboring vertices of airports.
  - *optimalPathSearchByAstar:* Uses the **A*** search algorithm to find the optimal route from a start city to a destination city.
  - *breadthFirstSearch: Uses the breadth first search algorithm to search for a route from a start city to a destination city*

Other important facts about this project are found in the readme.md file of this project repo.