**Air Route Planner**

Software Design Specification

*Version 2.1*

**That “One” Team**

SE300, Section 1

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **A/M/D** | **Author** | **Description** |
| 1.0 | 02/26 | A | Brian Powell | Skeleton layout, purpose, problem statement, UML |
| 1.1 | 03/26 | A | B. Rompa  C. Wilkerson | HLD, UML, planned methods |
| 1.2 | 04/02/13 | AMD | B. Rompa  C. Wilkerson | Update and revision of HLD, UML, methods |
| 2.0 | 04/11/13 | AMD | B. Rompa | Update and revision of methods, HLD, UML |
| 2.1 | 04/21/2013 | AM | B. Rompa | Modify methods, HLD, UML |

` \*A - Added, M - Modified, D - Deleted

**Table of Contents**

[Purpose 4](#_Toc349673747)

[Problem Statement 4](#_Toc349673748)

[Team Project Information 4](#_Toc349673749)

[High Level Design 5](#_Toc349673750)

[Detailed Design 13](#_Toc349673751)

# **Purpose**

The purpose of this document is to specify the design details for the Air Route Planner (ARP). The ARP will contain classes that break down the functionality into several key components: user interface, file handling, search algorithms and the manner in which the output will be formatted. A detailed description of what methods and features will be included in this software will be outlined in the later sections of this document.

# **Problem Statement**

The goal of this project is to create an Air Route Planner (ARP) that will allow a user to select an origin and destination based on a database of airports. The system will then search for a route based on time, cost, or airline most frequently used. This information will then be displayed back to the user.

# **Team Project Information**

Course: Spring 2013 SE300 Section 1

Team: That “One” Team

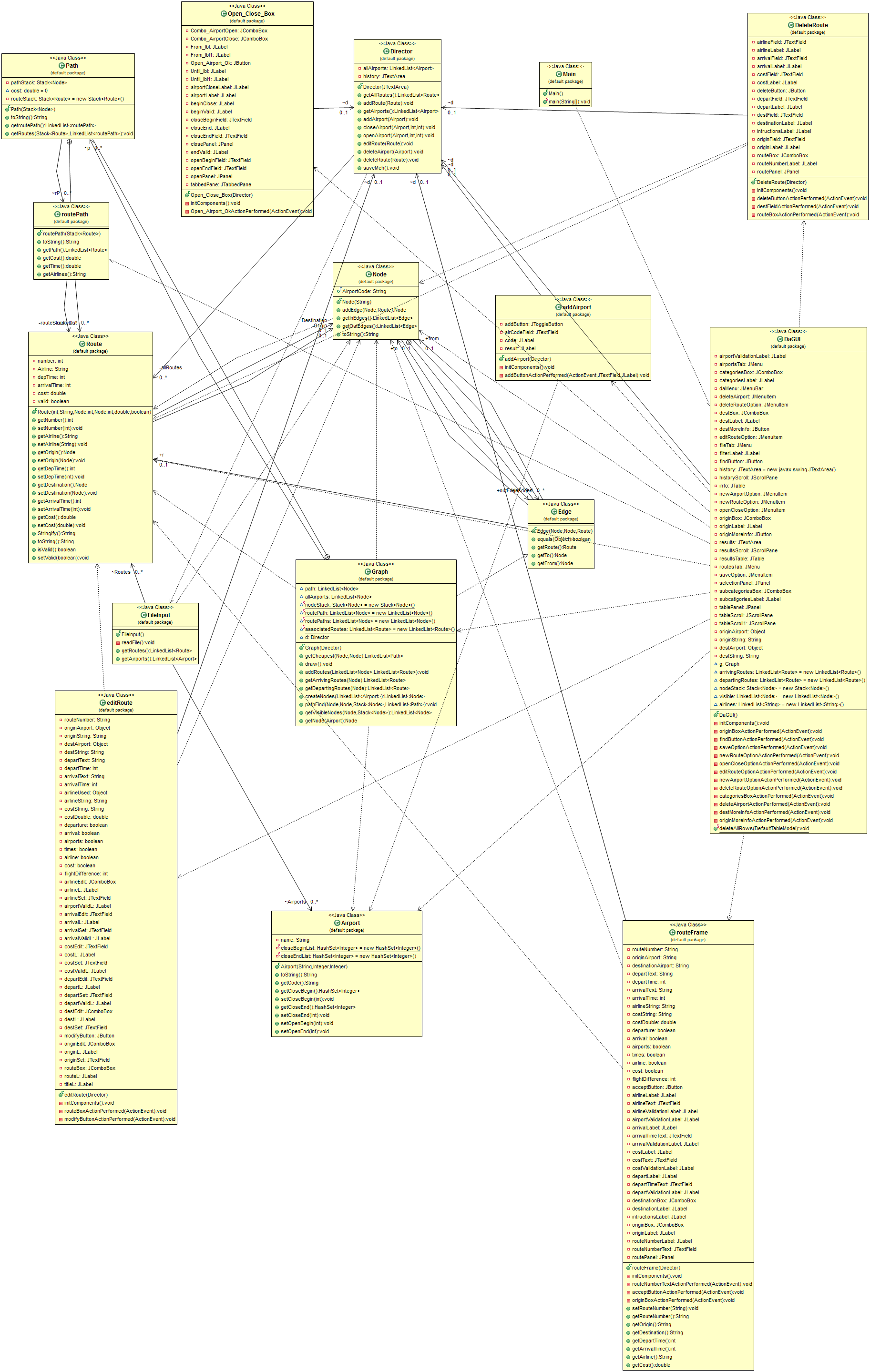
Members/Roles:

* **Team Leader:** Brian Powell
* **Development Manager:** Brittany Rompa
* **Planning Manager:** Craig Wilkerson
* **Quality Manager:** Yutong Zhu
* **Req/Support Manager:** Muraad Khan

# **High Level Design**

**Introduction**

The following is an explanation of the high level design. The Class Diagram illustrates the way each class functions and how they work together. A detailed list of each class, which includes their purpose(s) and methods, shall be defined later in the document.

**Class Diagram**

**Class Specifications**

* Director
  + Hold data
  + Manage changes of lists (of type route or airport)
  + Provide data for other classes
* FileInput
  + Read input file
  + Sort input file
  + Pass information to Director class
* Graph
  + Return route depending on filter
  + Connects flight paths
* Node
  + Return route to
  + Return route from
* Airport
  + Open airport
  + Close airport
  + Return airport names
* Route
  + Return/set route number
  + Return/set arrival time
  + Return/set departure time
  + Return/set origin
  + Return/set destination
  + Return/set airline
  + Return/set cost
* Main
  + Initialize and run DaGUI
* DaGUI
  + Main graphical user interface
  + Majority of functionality
  + Display route information
* DeleteRoute
  + Delete a route
* deleteAirport
  + Delete an airport
* editRoute
  + Edit origin airport
  + Edit destination airport
  + Edit airline
  + Edit depart time
  + Edit arrival time
  + Edit cost
* addAirport
  + Add an airport
* routeFrame
  + Create a new route
* Open\_Close\_Box
  + Open an airport
  + Close an airport

**User Interfaces**

The following figures display the entire graphical user interface for the ARP.

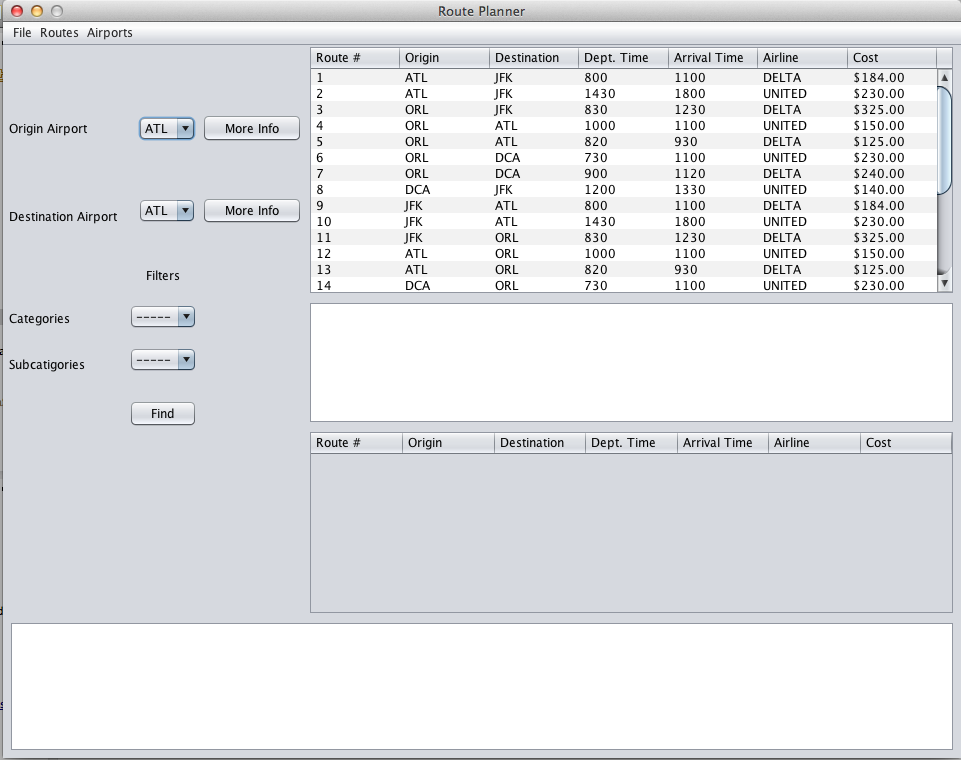


Figure – Class DaGUI – ARP Main GUI

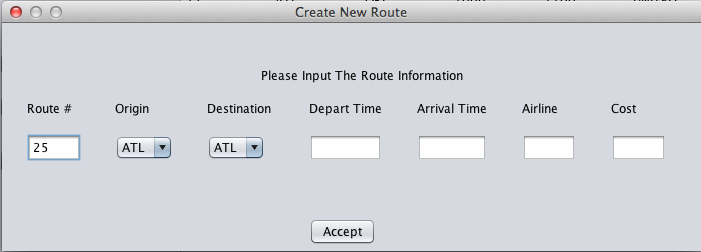


Figure – Class: routeFrame – Creating a New Route

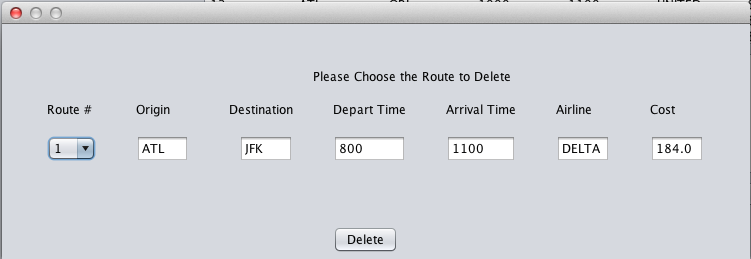


Figure – Class: DeleteRoute – Deleting a Route

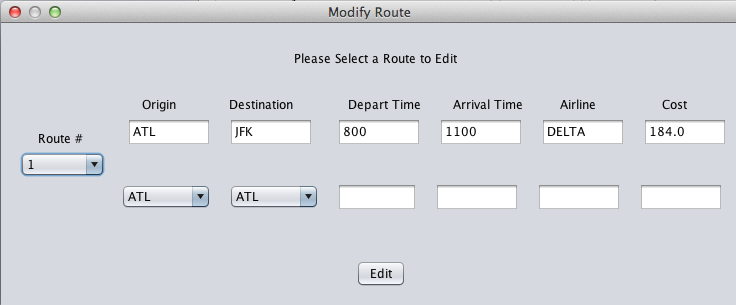


Figure – Class: editRoute –Editing a Route

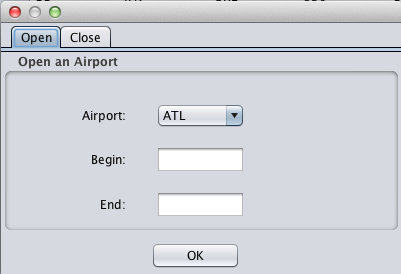


Figure – Class: Open\_Close\_Box – Open/Close an Airport

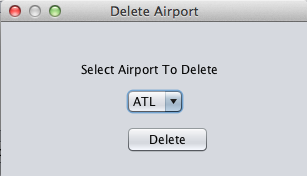


Figure – Class: deleteAirport – Delete an Airport

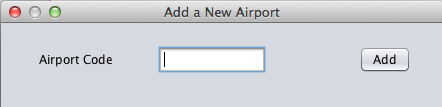


Figure – Class: addAirport – Add a New Airport

**File and Report Formats**

#comment \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#comment Example Input file

#comment Version 1.0

#comment Authors: Evan Richardson, Keith Garfield

#comment

#comment This file represents the state of the air network upon first system start-up.

#comment Use this file to ensure that your system has basic functionality with respect to loading files.

#comment Use this file as an example of how to create other input files you may need during development and test.

#comment \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#comment All airports must be listed first, per three-letter code, one airport per line

#comment Format: XXX

#airports

ATL

DCA

ORL

JFK

#comment All routes to be listed here, one route per line.

#comment Format: route\_id, carrier\_id, ap\_dep, time\_dep, ap\_arr, time\_arr, price, [val8, val9, val10, ...]

#routes

R001, DELTA, ATL, 0800, JFK, 1100, 184.00

R002, UNITED, ATL, 1430, JFK, 1800, 230.00

R003, DELTA, ORL, 0830, JFK, 1230, 325.00

R004, UNITED, ORL, 1000, ATL, 1100, 150.00

R005, DELTA, ORL, 0820, ATL, 0930, 125.00

R006, UNITED, ORL, 0730, DCA, 1100, 230.00

R007, DELTA, ORL, 0900, DCA, 1120, 240.00

R008, UNITED, DCA, 1200, JFK, 1330, 140.00

R009, DELTA, JFK, 0800, ATL, 1100, 184.00

R010, UNITED, JFK, 1430, ATL, 1800, 230.00

R011, DELTA, JFK, 0830, ORL, 1230, 325.00

R012, UNITED, ATL, 1000, ORL, 1100, 150.00

R013, DELTA, ATL, 0820, ORL, 0930, 125.00

R014, UNITED, DCA, 0730, ORL, 1100, 230.00

R015, DELTA, DCA, 0900, ORL, 1120, 240.00

R016, UNITED, JFK, 1200, DCA, 1330, 140.00

#comment All closures to be listed here.

#comment Format: airport\_code, time\_c, time\_o

#closures

#comment No closures in initial data set.

#comment Example: ATL, 0800, 1000

#end

Files are to be in .txt format.

# **Detailed Design**

**Introduction**

This portion of the document describes the purpose and function of each method. This is to be used to develop an understanding for how each method will interact with another method.

**Method Specification**

* getCloseBegin – Time an airport will begin closure
* getCloseEnd() – Time an airport will end closure
* getCode() – Returns 3 letter airport designation
* setCloseBegin() – Set time an airport will begin closure
* setCloseEnd() – Set time an airport will end closure
* setOpen() – Set an airport to open
* deleteAllRows() – Removes all rows of data from table
* addAirport() – Creates an airport
* addRoute() – Creates a route
* getAirports() – Returns all airports
* getAllRoutes() – Returns all routes
* editRoute() – Edits route information
* deleteRoute() – Deletes a route
* deleteAirport() – Deletes an airport
* saveMeh() – Saves the array to a text file
* addEdge() – Adds an edge (route)
* getInEdge() – Returns inbound edge
* getOutEdge() – Returns outbound edge
* getAirline() – Returns airline
* getArrivalTime() – Returns arrival time
* getCost() – Returns cost of flight
* getDepTime() – Returns departure time
* getDestination() – Returns the destination
* getNumber() – Returns the route number
* getOrigin() – Returns the origin airport
* setAirline() – Sets the airline
* setArrivalTime() – Set flight arrival time
* setCost() – Set flight cost
* setDepTime() – Set flight departure time
* setDestination() – Set flight destination
* setNumber() – Set flight route number
* setOrigin() – Set the origin airport
* Stringify() – Change object in array to string
* getRouteNumber() – Returns the route number
* setRouteNumber() – Sets the route number
* getDepartTime() – Returns the departure time
* getPath() – Returns path between origin and destination Nodes
* getArrivingRoutes() – Returns any routes associated with arriving node (airport)
* getDepartingRoutes() – Returns any routes associated with departing node (airport)
* addRoutes() – Adds linked list of Node and Route
* createNodes() – Creates nodes for airports in system
* pathFind() – Returns path of nodes between two nodes, in sequential order
* getVisibleNodes() – Returns reachable nodes from current node
* getNode() – Returns a node
* passDirector() – Pass information to Director class
* UpdateInfo() – Updates the table
* updateDirector() – Updates the Director