Lab 08B - 80 points – Airline Route Simulator (Weighted Graph)

A *weighted graph* is a graph in which each branch is given a numerical weight. A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive).

For this exercise you will modify the adjacency matrix you created in Lab 08A. Each node represents a city and the edges connecting the nodes represent the cost of a ticket. The graph will be implemented using a matrix of `int` values. **0** means there is no connection.

In addition to the **public String findRoute(int length, String start, String end)** method required by the **AirlineGraph interface** you will need a **public String toString()** method. The **toString()** method should return a String representation of the matrix. If a Graph object were printed it would look like this:

	BOS	CHI	DFW	DEN	HNL	IAH	MIA	JFK	PHX	SFO
BOS	-	-	-	-	-	-	-	113	-	-
CHI	-	-	-	89	-	131	-	113	-	-
DFW	-	-	-	-	388	165	-	-	-	123
DEN	-	198	-	-	-	-	-	-	-	194
HNL	-	-	351	-	-	388	-	-	-	352
IAH	-	98	165	-	407	-	128	-	99	-
MIA	-	-	-	-	-	148	-	156	-	-
JFK	113	113	-	-	-	-	148	-	-	-
PHX	-	-	-	-	-	68	-	-	-	-
SFO	-	-	141	99	294	-	-	-	-	-

The Graph class will need to implement a two-dimensional matrix and a Stack.

Field Summary

private static int[][]	**graph** A two dimensional *adjacency matrix*.
private static Stack<Integer>	**stack** A last in first out data structure.

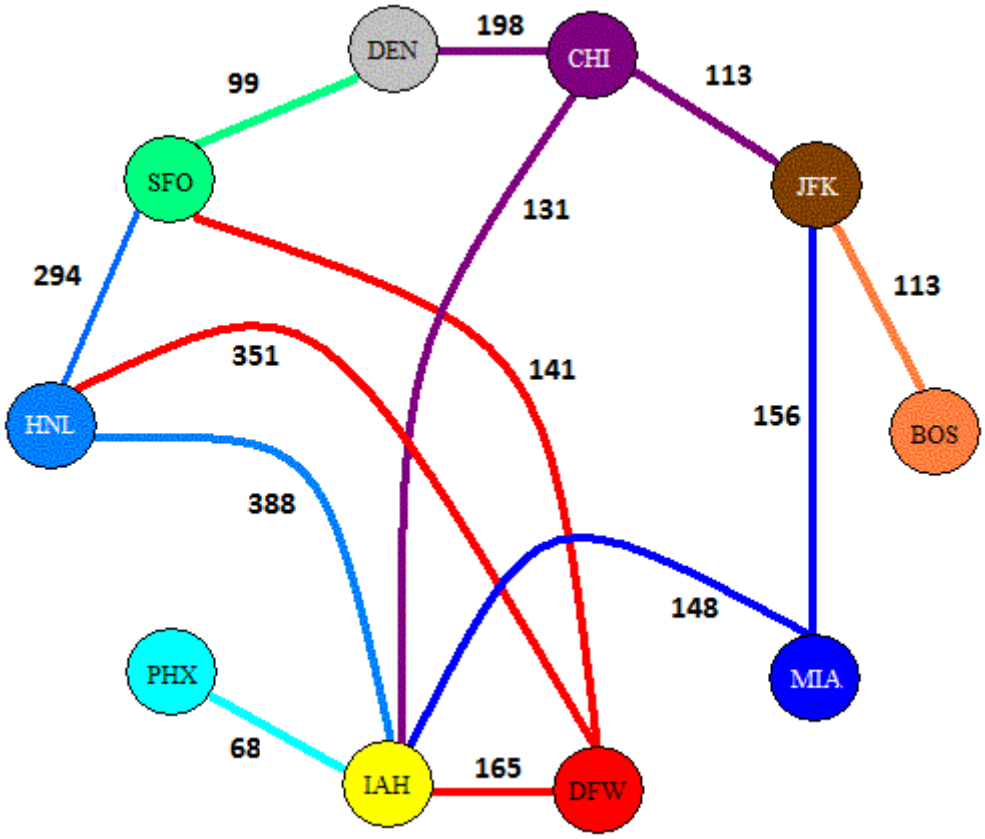
Constructor Summary

Graph()
Initializes a newly created Graph object. Initializes the *adjacency matrix* to be a **10 X 10** grid and initializes the grid by setting the points contained in the data file "connections.dat" to be the designated value.

Method Summary

public String	**findRoute(int length, String start, String end)** Returns a String containing all the cities visited from <*start*> to <*end*> including <*start*> and <*end*> in the order visited followed by the total fare.
public String	**toString()** Returns a String representation of this Graph object.
private int	**findAirportCode(String airportCode)** Returns the index position of the specified airportCode.
private boolean	**adjacent(Point edge)** Returns true if Point edge is connected, false otherwise.

private boolean	**findPath(int length, Edge p)** Returns true if a path of length <length> exists between the two points in Edge <p>. If a path exists, the index values for each airport visited is pushed onto the stack. **Algorithm:** **if length equals 1** **push the current city onto the stack** **return adjacent(p)** **else** **for every *node* in the graph** **if adjacent(p) && findPath(length – 1, Edge(*node*, p.x))** **push the current city onto the stack** **return true**



Airline Graph

	BOS	CHI	DFW	DEN	HNL	IAH	MIA	JFK	PHX	SFO
BOS								113		
CHI				89		131		113		
DFW					388	165				123
DEN		198								194
HNL			351			388				352
IAH		98	165		407		128		99	
MIA						148		156		
JFK	113	113					148			
PHX						68				
SFO			141	99	294					

A Matrix representation of the Airline Graph

Run **AirlineRoutes** to test your class.

OUTPUT

```
      BOS   CHI   DFW   DEN   HNL   IAH   MIA   JFK   PHX   SFO
BOS    -     -     -     -     -     -     -    113    -     -

CHI    -     -     -    89     -   131     -    113    -     -

DFW    -     -     -     -   388   165     -     -     -   123

DEN    -   198     -     -     -     -     -     -     -   194

HNL    -     -   351     -     -   388     -     -     -   352

IAH    -    98   165     -   407     -   128     -    99     -

MIA    -     -     -     -     -   148     -   156     -     -

JFK  113   113     -     -     -     -   148     -     -     -

PHX    -     -     -     -     -    68     -     -     -     -

SFO    -     -   141    99   294     -     -     -     -     -


There is no direct connection!
Phoenix, AZ -> Houston, TX -> Miami, FL  $196.00
San Francisco, CA -> Denver, CO -> Chicago, IL -> New York, NY  $410.00
Honolulu, HI -> Houston, TX -> Chicago, IL -> New York, NY -> Boston, MA  $712.00
```