Homework 4 : Problem 1

Graph Class:

- Public Graph(){}
  // Base case constructor
  // shown as an adjacency list where is displays a connection to a named node and
  // the edge weight in each list index
  @ modifies initializes nodes and their edge values
  @ effects will initialize a new graph

- Public Graph(Graph existing)
  // copy constructor
  @ parameter takes in an existing graph
  @ requires existing graph to not be NULL and existing is a Graph object
  @ modifies nodes and held values for each node
  @ effects copies existing to the new graph, they should be identical
  @ throws RunTimeException when a NULL object is inputted.

- Public Void AddNode(string NewNode)
  // adds a new node to the graph
  @ parameter the NewNode string
  @ requires NewNode be a valid string and not NULL
  @ modifies list of node names and list of node connections
  @ effects adds a node to the graph
  @ throws RunTimeExecption if the string is NULL

- Public Void AddEdge(string start, string end, string weight)
  // adds a connection (edge) between to existing nodes in the graph
  @ parameter start and stop nodes, as well as the string weight
  @ requires start and stop nodes are already in the graph, and weight is not NULL
  @ modifies graph adjacency list at start node index by adding a 2 string array
  displaying an edge's destination node, and weight
  @ effects
  @ throws

- Public String GetWeight(string start, string end)
  // returns the weight of an existing node, or -1 if there is no existing edge
  @ parameter the start and stop nodes
  @ requires both nodes must be contained in the graph
  @ return string the represents edge's weight value

- Public Boolean GraphContains(string node)
  // returns true if node is in graph and false otherwise
  @ parameter node
  @ return true if lists contain node, else false

- Public List<string> FindParents(string Node)
  // returns all nodes that connect to this particular node
  @ parameter Node (child node)
  @ requires Node is not NULL
  @ return list of all nodes that connect to Node where Node is their child

- Public List<string> FindChildren(string Node)
  // returns all the nodes that this node connects to (list of children nodes)
  @ parameter Node (parent node)
  @ requires Node is not NULL
  @ return list of nodes where this Node connects to (list of children nodes)

- Public List<string> allNodes()
  // returns the names of all nodes in the graph
  @ requires graph is not null
  @ returns ArrayList<String> of all node names sorted alphabetically

- Public List<string> allChildren(String nodeName)
  // returns all the names of the nodes that have directed edges coming from nodeName
  @ parameter nodeName exists in the graph
  @ requires node Name to exist in the graph
  @ returns list of node names of the nodes that have directed edges coming from nodeName alphabetically

Node Class:

- Node(String name)
  // constructor for node class, takes in a name by default
  @ parameter string name
  @ modifies, initializes node name and its connection ArrayList
  @ effects will initialize a new node with the name inputted by name

- addEdge(String To, String Label)
  // stores information about a directed edge going to node To, titled Label.
  @ parameters strings To, and Label
  @ requires neither string is null
  @ modifies connection arraylist
  @ effect adds an index to the connection ListArray representing an edge.

- getName()
  // returns name of the node
  @ requires node is not null
  @ returns name of the node

- getWeight(String nodeName)
  // returns the Label/weight of a desired edge going to node nodeName
  @ parameters nodeName
  @ requires nodeName to be the name of a node that exists in the graph & connects to this node
  @ returns edge's label

- kids()
  // returns the list of all child nodes to this node in particular
  @ requires node is not null
  @ returns ArrayList of all child nodes

- allKids()
  // returns the list of edges where each index is [node_name, label/weight]
  @ requires node is not null
  @ returns ArrayList of an ArrayList of all child nodes and their labels

- connects(String nodeName)
  // sees if node object has an edge going to nodeName
  @parameter nodeName
  @ requires string is not null and there is a node that has this name
  @ returns true or false depending on if the node shares an edge to nodeName