Problem 1: Write a Specification

```
package hw4;
import java.util.ArrayList;
import java.util.Dictionary;
import java.util.Hashtable;
import java.util.Enumeration;


//A Graph is a directed multigraph. This means that there can be an edge going
// from node A to node B and an edge going from node B to node A. There
// also has the possibility to have multiple edges going from A to node B as
// long as those nodes have different edge weights ("labels")

public class Graph {
    // Abstraction Function:
    // A Graph is made up of nodes and edges. When two or more nodes are added
    // it is possible to connect them with an edge. All edges must have an edge weight
    // thus when an the edge weight is removed it should remove that particular edge
    // although it is possible to have multiple edges of different edge weights connecting
    // those two nodes. The adjacencyList is the particular number of edges that a particular
    // node can explore to.

    // Representation invariant for every Graph g:
    //   (g.getNodes().size() >= 0) &&
    //   (g.getEdges().size() >= 0) &&
    //   (g.edgeInGraph(node1, node2) == true ==> g.get(node1).get(node2).size() > 0) &&
    //   for node in g.getNodes() ==> g.getEdges().size() == sum(g.getAdjacencyList(node).size())())
    //
    //   In other words,
    //     * number of nodes in graph is non-negative
    //     * number of edges in graph is non-negative
    //     * number of edge weights is never zero
    //     * The sum of all the adjacency lists (including edgeweights) must equal the number of edges


    //constructor
    public Graph() {}

    // @requires nothing
    // @param String value of the node to add
    // @effects Adds the node to the Graph and does nothing if the node already exists
    // @modifies Graph
    // @throws nothing
    // @returns nothing
    public void addNode(String node) {}

    // @requires nothing
    // @param String value of the source node and the destination node and value of the edge
    // @effects Adds the edge to the Graph and does nothing if the nodes are not in the graph or
    //          is a repeat edge weight
    // @throws nothing
    // @returns nothing
    public void addEdge(String node1, String node2, String value){}

    // @requires nothing
    // @param String value of the node to remove
    // @effects Removes the node from the Graph and does nothing if the node is the node
    //          is not in the graph
    // @effects It will also remove all edges going into that node
    // @modifies Graph
```

```
// @throws nothing
// @returns nothing
public void removeNode(String node) {}

// @requires nothing
// @param String of the source node and the destination node
// @effects Removes all edge weights and the connection between the source node
//          to the destination node
// @effects It does not effect the connection from the destination node to the source node
// @effects It will do nothing if the edge does not exist
// @modifies Graph
// @throws nothing
// @returns nothing
public void removeEdge(String node1, String node2) {}

// @requires nothing
// @param String of the source node and the destination node and the weight to remove
// @effects Removes the edge weight for that edge from the Graph
// @effects Removes the edge if there are no more edge weights for that edge
// @effects Does nothing if the edge weight does not exist
// @modifies Graph
// @throws nothing
// @returns nothing
public void removeEdgeWeight(String node1, String node2, String value) {}

// @requires nothing
// @param nothing
// @effects nothing
// @modifies nothing
// @throws nothing
// @returns ArrayList<String> of nodes
public ArrayList<String> getNodes(){}

// @requires nothing
// @param nothing
// @effects nothing
// @modifies nothing
// @throws nothing
// @returns ArrayList<String[]> (each edge will be an array of size 2 [source, destination]
public ArrayList<String[]> getEdges(){}

// @requires nothing
// @param String value of the source node and the destination node
// @effects nothing
// @modifies nothing
// @throws nothing
// @returns ArrayList<String> of edge weights
public ArrayList<String> getEdgeWeight(String node1, String node2) {}

// @requires nothing
// @param String value of the node
// @effects nothing
// @modifies nothing
// @throws nothing
// @returns ArrayList<String> of adjacent nodes
public ArrayList<String> getAdjacencyList(String node){}

// @requires nothing
// @param String value of a node
// @effects nothing
// @modifies nothing
```

```
    // @throws nothing
    // @returns true if the node is in the graph
    // @returns false if the node is not in the graph
    private boolean nodeInGraph(String node) {}

    // @requires nothing
    // @param String value of the source node and the destination node
    // @effects nothing
    // @modifies nothing
    // @throws nothing
    // @returns true if the edge is in the graph
    // @returns true if the edge is not in the graph
    private boolean edgeInGraph(String node1, String node2) {}

    // @requires nothing
    // @param String value of the source node and the destination node and the edge weight
    // @effects nothing
    // @modifies nothing
    // @throws nothing
    // @returns true if the edgeWeight is in the graph
    // @returns false if the edgeWeight is not in the graph
    private boolean edgeWeightInGraph(String node1, String node2, String value){}
}
```