***Interface-Class Pairs Explanation***

* GraphInterface
  + - This contains all the methods exposed to the user. We have four methods that need to be exposed. The first 3 are called in UserView, whilst the last one is called in Controller.
      * getNode() returns the node object if one exists using a given string.
      * checkDuplicateNode() checks to see if there is more than 1 node with a given name and returns a boolean value
      * getNodeByLine() is used if there is a duplicate node and asks for the line of the intended node
      * getRoute() returns a LinkedList of edges that the UserView will use to display the route in the correct format.
  + Graph
    - Graph implements GraphInterface. It implements the inherited getRoute() method using a form of Dijkstra’s algorithm that uses an agenda and a dictionary. It returns the route as a LinkedList of edges with the help of a few other methods.
      * findShortest() finds the shortest route so far in the agenda
      * pathSoFar() returns how long the given path is
      * buildRoute() builds the final route and then converts it to a LinkedList of nodes with the proper names
      * checkEdge() checks whether there is an edge between two given nodes and returns that edge if it exists
      * getAdjacent() returns an ArrayList of all the nodes connected to the given one
    - There are a few other auxiliary methods not used in our implementation, however we felt Graph would be incomplete without it
      * getEdges() returns an array of all the connected edges to the given node
      * getDegree() returns the number of edges connected to a given node
    - Finally
      * checkNode() is used to determine if a certain node exists given the name of a station as a parameter. This is used when the user enters a node name. The name the user enters needs to be the same characters as the input file, however, as we process the input, the chars can be in any almost any format. This means that caps, no caps, spaces etc shouldn’t make a difference to getting the correct node.
* EdgeInterface
  + - EdgeInterface includes methods to return the IDs of nodes on either side of it, getLeftNode() and getRightNode(). It also has a method to return the weight of that edge, getWeight(). In theory the weight would be how long it takes to travel that edge, however it is set to a blanket value of 1 as we have no way to read in the real values.
  + Edge
    - Edge implements EdgeInterface. It includes getter and setters for all variables in the class.
    - ‘leaving’ and ‘arriving’ are used in UserView to print the names of the nodes you are leaving from and arriving at on your route. They are set in buildRoute()
    - leftNode and rightNode are the IDs of the nodes either side of the edge
    - line is just whatever colour line the edge is on
* NodeInterface
  + - This is the interface that Node implements. It includes methods to return the names and IDs of the node, getName() and getID().
  + Node
    - Node implements NodeInterface. It only has 1 method of its own, getWeight(). This, in theory, would be the amount of time it would take you to switch lines at that node, but much like with the edges we have no way of reading that information in. For this reason, weight is set to a blanket value of 15. In the case of this particular practical, it means that the algorithm will only switch lines when 100% necessary.

***Changes made***

There are quite a few changes that needed to be made from our initial block diagram to the final one. One of the biggest is that now we call the file parser from our main controller class instead of from within the Graph. The parser returns an object of Map<Node, ArrayList<Edge>> that we can then cast to Graph. This also allowed us to trim a few methods out of the Graph class such as setup() and addEdge(). However, in the process of testing we found there were two stations with the same name. This creates a problem when trying to travel between them. This prompted us to add a few more methods to GraphInterface to deal with the duplicate station names. These changes to graph are also reflected in a few extra methods in UserView to print and return the information needed. The Edge and Node interfaces were slightly changed in order to better suite them to how the program evolved. This meant that edge could hold strings instead of full node objects within it. Controller became our main, whereas we didn’t have a main previously, and with that gained the ability to terminate the program. UserView gained another method that assisted in exiting. Many method names were also changed in order to make them better match what they were designed to do.