***Block Diagram Explanation***

* GraphInterface
  + - This contains all the methods exposed to the user. In this case the only thing the user needs is a method that takes where they are and where they want to go and returns a route to do that. This is done in the getRoute() method.
  + 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 tracks 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 stations with the proper names
      * checkEdge() checks whether there is a track between two given nodes and returns that track if it exists
      * getAdjacent() returns an ArrayList of all the stations connected to the given one
    - There are a few other methods. Some are auxiliary methods not used in our implementation, however we felt Graph would be incomplete without it
      * getEdges() returns an array of all the connected tracks to the given station
      * getDegree() returns the number of tracks connected to a given station
    - 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 station name
* EdgeInterface
  + - EdgeInterface includes methods to return the IDs nodes on either side of it, getLeftNode() and getRightNode(), as well as a method return the weight of that edge, getWeight(). In theory the weight would be how long it takes to travel that track, however it is set to a blanket value of 1 as we have no way to read in the real values.
  + Track
    - Track 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 stations you are leaving from and arriving at on your route. They are set in buildRoute()
    - leftNode and rightNode are the IDs of the stations either side of the track
    - line is just whatever colour line the track is on
* NodeInterface
  + - This is the interface that Station implements. It includes methods to return the names and IDs of the station, getName() and getID().
  + Station
    - Station 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 station, 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.

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<Station, ArrayList<Track>> 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(), which in turn completely decoupled the Graph ADT from the rest of the system. 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.