Interfaces:

MultiGraphADT- This interface contains method declarations that allow for the handling of data in the context of a MultiGraph.

Edge-The Edge interface contains method declarations for retrieving data for an edge between two Nodes.

Node-The Node interface represents a typical vertex in a graph, it only has two methods for retrieving its name and unique identifier.

Classes:

MultiGraph- Implements MultiGraphADT and provides implementation of each function from the interface. It creates a collection of edges and nodes, additional methods are also provided to aid the functionality for the searching and creation of routes.

BostonMetro- Creates a new MultiGraph using the Parser createMap (see below) method, this class also contains all the console interaction for a user to find a route between stations. BostonMetro has a FindRoute attribute which is supplied with a source and destination, and returns an ArrayList of “directions”. (Subject to change).

Line-This class implements Edge, taking in the name of the line, the source vertex and the destination vertex. BostonMetro sends Lines to the MultiGraph it uses, as a Line implements Edge, and, as such, is an accepted parameter for its methods (e.g addEdge). Line has an overridden hashCode and equals method, partly for testing purposes, and partly for future HashMap implementation requirements.

Station-Implements Node. A station has the exact same information “blueprint” as a Node. Station also has an overridden equals and hashCode method, use for future changes involving hashmaps. BostonMetro also sends Stations to the MultiGraph it uses, as a Station implements Node, and, as such, is also an accepted parameter for its methods (e.g addNode).

FindRoute – This class takes a source node and destination node and from that finds a walk between the two. Its main method for finding routes returns an ArrayList of lines.

Parser- Parses information from a given file about stations and lines. BostonMetro uses this to return a MultiGraph which is used later on for when a user wants to find routes. The classes main createMap method also checks for errors in the grammar of the file.

b) One relationship is that BostonMetro uses a MultiGraph which implements MultiGraphADT, as the BostonMetro and MultiGraph have the same concept of nodes and edges (in the context of the BostonMetro; stations and lines).

Line also implements Edge, as they contain the same components but Line in the context of subway lines. This is the same case for Station implementing Node, the stations of the Subway being like the vertices of a graph. Line and Station have some extra overridden hashCode and equals methods, for testing and future implementation.

c)

MultiGraph:

nNodes -Returns the size of the array list of nodes, ie the number of nodes nEdges-returns the size of the array list of edges, ie the number of edges

addEdge- Adds the given edge to the list of edges.

addNode- Adds the given node to the list of nodes

isEdge- Verifies whether or not the given edge is an edge in the list

successors- Returns a list of nodes that are adjacent to a supplied node.

getEdges- Returns all edges

BostonMetro:

run()- creates a new MultiGraph

Line: (implements Edge)

getLabel- Returns the colour of the line

getSrcNode- Returns the station that a train is coming from.

getDestNode- returns the station that the train will be going to next.

hashCode- Used so that a HashMap (or any method which uses hashing) correctly knows how to compare two values to another(yet to be fully implemented)

equals- Compares 2 nodes to check if they are equal

toString- For testing purposes, self explanatory.

Station: (implements Node)

getName – Returns the stations name/label.

getId – Returns the stations ID.

hashCode- Used so that map correctly knows how to map values to another(yet to be fully implemented).

Parser:

createMap- Reads a file of lines and stations to create a MultiGraph Returns a value which must implement MultiGraphADT as it’s in the method signature.