

**Overview**

In order to promote decoupling within the project, and potential future re-usage of certain classes (such as those for the Multigraph), the project has been separated into two halves. The two halves consist of those representing functionality of the Metro System, and another outlining the Multigraph. Furthermore, to decouple the user’s interactions with the program for the Boston Metro System, the Model-View-Controller pattern will be implemented.

**Role of Interfaces**

IMultigraph: Outlines the functionality of a multigraph. Public methods will allow for the addition of Nodes and Edges to the multigraph. Multigraphs will also be capable of returning a List of Edges representing a path between two Nodes. The List being an implementation of the interface java.util.List.

IEdge: Outlines the functionality of Edges, used within multigraphs. Methods may include returning the label of the edge, or returning one of the nodes that the edge connects to (two separate methods are required, one for each node).

INode: Outlines the functionality of Nodes, used within multigraphs. This may include returning the values of the ID.

**Role of Classes**

MultiGraph: A concrete implementation of the IMultigraph interface. This would allow the program to instantiate an instance of Multigraph and use it in place of IMultigraph references.

Line: Likewise, this provides an implementation of the IEdge interface.

Station: This provides an implementation of the INode interface. Stores the name.

MetroApp: This class must be run by the user to initialise the program. It contains the only main method within the project.

MetroMapParser: This class (which is provided) will parse the given text file in order to fill a MultiGraph with information regarding the Boston Metro system (such as the stations and track lines).

ConsoleIO: This class acts as the View within the Model-View-Controller; it will manage user messages and retrieve user-input through the console.

Controller: This class acts as the Controller within the Model-View-Controller; it will parse the user input from an instance of User Prompt, and manage the Metro system app in accordance to the input. Controller now holds Multigraph rather than the initial design of using Metro.

**Relationships between interfaces and classes**

Interfaces: IMultigraph, IEdge, INode

Implementations of Interfaces: MultiGraph, Line, Station (respectively)

Within the Multigraph half of the program: Multigraph (the implementation of IMultigraph) will store a List of INodes and IEdges.

IEdges will store two INodes.

Within the Metro half:

MetroApp creates a Controller which will have an instance of the class MetroMapParser.

Controller will hold the Multigraph rather than the initial design of Metro holding it. We removed the metro class.

**Method descriptions**

IMultigraph/MultiGraph:

addNode(INode) – Allows for the multigraph to be populated with nodes.

addEdge(IEdge) – Allows for the multigraph to be populated with edges.

getRoute(INode, INode) – Given two nodes, it will return a List (java.util.List) which

contains a series of edges in a path between the two nodes.

getNodes() – returns a list of nodes

getNodesWithName(String) – returns nodes with a specific name.

getNode(int) – Returns a node with a given ID.

Successors(Node) – Returns a list of successor nodes to a given node.

IEdge/Line:

getNode1() – returns the INode of one of the nodes the edge connects to.

getNode2() – returns the INode of the second node that the edge connects to.

getlabel() – returns the label of the edge.

getOtherNode(int) – returns the node the user is interested in when looking at an edge connecting two nodes.

INode/Station:

getID() – returns the ID of the node

setName(String) – sets the name of a station

getName() – sets the name of a station

ConsoleIO:

prompt(String) – prompts the user with a given message past in as a parameter then returns the user input.

printRoute(List<IEdge>) – formats the route of edges past in

formatRouteList(List<IEdge>)- constructs the Route list so that it can be output.

printNumberOfStops(String, int) – prints the number of stops between a section of the journey.

printDirectionOfTravel(int, IEdge)- prints the direction of travel from a connecting node so the user knows which way to travel when changing lines.

printLineChange(INode,IEdge) – prints then change of line required to reach the destination.

printStationDetails(IEdge,int) – prints the information about the given station.

PrintList(List) – Used to print out a collection of all elements.

Controller:

InitialiseMetro – calls metromapParser and returns that in the constructor of metro

Run() – where the main loop for the reading user input is based, it was placed in here as it includes some validation(of whether it was a valid station) and wanted to make sure the ConsoleIO didn’t know about metro or the controller.

validateInputStation(String)- validates the input from the user and provides some error checking. If the user doesn’t provide the station with a capital letter it will automatically change it.

manageStationNotValid(String) – If the station it not valid an error message will be printed so the user is made aware.

manageDuplicateStation(List<Node>) – Used to provide help when a duplicated station is detected.

MetroMapParser:

generateGraphFromFile() – this returns some concrete implementation of IMultigraph filled with Stations and Lines based on some txt file that was already loaded.

createStationIfNotPresent(int, IMultigraph) – Creates a station if it is not already present in the graph.

**Changes from initial design**

* We removed the Metro class from our initial design and allowed controller to hold the multigraph as the class wasn’t really required.
* Add several methods to different classes in order to split up the work.
* Provided more error checking so that the users input would work even if the put spaces between input and didn’t capitalize their entry.
* Improve the look of the console output so it is easier for the user to use, and also easier for them to understand the journey plan.