

**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, as well as the removal of Nodes and Edges. 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 or the name of the node.

**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.

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

Station: This provides an implementation of the INode interface.

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).

Metro: This class acts as the Model within the Model-View-Controller. It will initialise and store the Multigraph representing the Boston Metro system.

UserPrompt: 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.

**Relationships between interfaces and classes**

Interfaces: IMultigraph, IEdge, INode

Implementations of Interfaces: MultiGraph, Track, 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 classes MetroMapParser, UserPrompt, and Metro.

Metro will store the multigraph, and supply it with the Tracks and Stations representing the Boston Metro system.

**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.

IEdge/Track:

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.

INode/Station:

getID() – returns the ID of the node

Unsure if there will be methods under UserPrompt/Controller/MetroApp/MetroMapParser

Notes/Highlighted parts to change/consider:

**Document requirements: A brief description of your diagram (approximately two pages) that:**

1. **explains the role of each interface and each class in the design**
2. **explains the relationships between the interfaces and classes that you have identified**
3. **includes a brief description of each method that you have identified in each interface and class**

“Removal of Nodes and Edges”: Likely redundant and will probably be removed (needs to be removed from UML). Same as other setters like under iEdge.

“or the name of the node.”: Do not know if name of stations is a necessity; not currently in UML.

“It will initialise and store the Multigraph representing the Boston Metro system.”: Not sure if it was intended for the graph to be made in Controller and the stations/tracks to be added at that point, and then passed into Metro. Or if the Multigraph is made under Metro, and then Controller passes in stations/tracks.

“and supply it with the Tracks and Stations representing the Boston Metro system.”: Same issue as above.