Implementation/list of Classes:

I. Interfaces/Classes/Methods
   A. Node (Airport)
      1. Represents a node on a "graph"
   B. Addable (Interface)
      1. The interface for the object cost takes
   C. ConnectionGroup from a node (Flight groups)
      1. Represents a group of connections/paths from a single origin Node on the graph to other Nodes
   D. Connection between nodes (with cost)(Flights, bus route, etc.) (Interface)
      1. Represents a connection between two Nodes on a graph
   E. Cost (different from RouteTime)
      1. represents a "cost", either to traverse a certain Node, or to cross a path. Has the ability to deal with "times", etc.
   F. ConnectionType
      1. (Optional) Represents different variation of "connection" e.g. walking, running, different kinds of flights, etc.
   G. PathNode
      1. Represents a node on a path (has information about total cost)
   H. PathState (Route State)
      1. Represents a graph that has the shortest found path from an origin point to other points on the graph
   I. PathFinder (Route Finder)
      1. Represents a graph, and has functionality to find minimum cost path from one point to another

Error Handling for Program (overall):

- Error handling and checking occurs throughout the program
  - Handle globally: keep passing errors up in more generic form, so then can handle/deal with them appropriately at higher levels
- Issues and errors are logged via Logger as they appear with more detailed information
- The types of errors that will be passed are:
  - `NullPointerException`
  - `IllegalArgumentException`
  - `IllegalStateException`
  - The cases in which both of these are called is described below/in method descriptions

- These are errors that we expect to be thrown when people use our package incorrectly

Class Node is comparable

Represents a node in the graph

Variables:

private String ID - identifies the node

private Cost nodeCost - Cost of visiting the node (it can be 0, similar to minimum layover time at an airport)

private ConnectionGroup outConnections - contains nodes which this node is able to connect to

Methods

of/builder

- throws illegal argument exception if it has invalid inputs (e.g. String is empty, etc.)
- Throws null pointer exception if null inputs

String getID()

- getter for ID

Cost getNodeCost

- getter for nodeCost

boolean equals (Object other)

- compares two nodes by their IDs. Two nodes are equal if their IDs are equal.

int hashCode()

- uses the ID value for the node's hashcode

int compareTo(Node otherNode)

- uses ID value to compare Nodes, a node is larger than another node if it's ID is larger than the other node's ID

final boolean addConnection(Connection)

- Takes a Connection and adds it to outConnections, returns true if successful, false otherwise

final boolean removeConnection(Connection)

- Takes a Connection and removes it from outConnections, returns true if successful, false otherwise

Set<Connection> availableConnections()

- returns set of connections available from the node

Set<Connection> availableConnections(ConnectionType)

- takes a connection and returns a set of connections of the given connection type

Set<Connection> availableConnections(Cost)

- takes in a local time and returns a set of connections larger than the given cost

Set<Connection> availableConnections(ConnectionType, Cost)

- takes a connection and returns a set of connections of the given connection larger than the given cost

- Uses Connection Group’s methods

class ConnectionGroup

Represents a set of Connections that have the same origin node.

Variables:

private variable Map<LocalTime, Set<Connection>> connections

- a map of sets of Connections matched with their available times

private variable originNode - the node the connections originate from

Methods:

boolean add (Connection)

- Checks connection is not null
- If the connection has the same origin node as the connection group, adds the connection to the connection group
- otherwise, does not add the connection and logs the issue

boolean remove (Connection)

- Checks connection is not null
- assumes Connection exists in the connection group
- removes the connection from the connection group

Set<Connection> connections()

- returns the connections (getter method)

Set<Connection> connectionsAtOrAfter(Cost cutOff)

- Checks cost is not null
- returns the Connections with a greater cost

Set<Connection> allConnections()

- returns all connections

getOriginNode() - getter method for origin

abstract class Connection

A connection represents a non-stop path from start node to end node

private final Node origin

private final Node destination

private final Cost cost

private final ConnectionType connectionType

Methods:

private Connection(Node origin, Node destination, Cost cost, ConnectionType connectionType)

- Private constructor for connection

static Connection of(Node origin, Node destination, Cost cost)

static Connection of(Node origin, Node destination, Cost cost, ConnectionType connectionType)

- creates a new connection and adds it to the connection group of the origin airport
- Throws null pointer exception if input is null
- Throws an illegal argument exception if the input bad arguments (e.g. origin and destination are the same)

getOrigin()

- Return startNode

getDestination()

- Return endNode

getCost()

- Return cost

getConnectionType()

- Return connection type
- Throws illegal state exception if this doesn't exist/if it's a null pointer

Boolean isLowerCost (Connection, Object)

- Check connection and Object are not null
- Return whether this connection's cost is lower than the other one when the input is the Object (cost is function)

Boolean isLowerCost (Connection)

- Check connection is not null
- Return whether this connection's cost is lower than the other one with no input (cost is constant)

int hashCode

- returns the hash code of the connection

class Cost

Represents costs path. Since cost is not a constant in all cases, we need to modify the RouteTime.

- is comparable
- has a generic type T which is Addable

Methods:

static final Cost UNKNOWN

- Represents unknown path time, with null cost/internal big integer

T cost()

- Returns a cost based on private variables only
- If cost cannot be computed only via private variables, then this should return some invalid value

Cost<T> of(T)

- returns a new cost object, of type T

T cost(T)

- Check Object is not null/is of the appropriate type
- Return a cost based on the input and private variable(s), if any
- If the input object is irrelevant, just return the value of cost(), and do not check for null
- It should throw an Illegal Argument Exception if the Object isn't the appropriate type

Cost plus(Cost)

- returns the new cost after adding the cost
- It should throw an Illegal Argument Exception if the Object isn't the appropriate type (e.g. Object is a BigInteger when we expected a LocalTime, etc)

compareTo(Object)

- compares two Cost objects, throws an illegal argument exception if input is not a Cost object

class ConnectionType

Represents a certain type of connection

Variables:

private final String identifier

Methods:

String getIndentifier()

- public getter for the identifier variable

boolean equals(Object other)

- compares two connection types by identifier. Two connection types are equal if their identifiers are equal

int hashCode()

- returns the hashcode of the identifier

(Eg. in Multimodal Route Finder, connection types are private vehicles, public transportation, and simply walking)

final class PathNode implements Comparable<PathNode>
represents a node in the path from the original departure to the final destination

Variables:
private final Node node - the node on this path
private final Cost cost - the cost of the path from the first Node (going to previous until you reach null)

private final PathNode previous - the path node directly before this path node, null previous denotes that this node is the original node

Methods:
Node getNode() - getter for node
Cost getCost - getter for cost of the path
PathNode getPrevious() - getter for previous
private PathNode (Node node, Cost cost, PathNode previous)
- initializes values

static final PathNode of (Node node, Cost cost, PathNode previous)
- checks for null values for node and cost
- previous can be null

static final PathNode of(Connection connection, PathNode previous)
- creates new PathNode with the destination node of the connection

static final PathNode of(Node node)
- creates a new PathNode with the given node, a Cost of UNKNOWN, with null previous value

static final PathNode of(Node node, Cost cost)
- creates a new PathNode with the given node, a Cost of cost, with null previous value

final Cost totalCost()
- returns a new Cost out of the addition of the node's cost and this cost

final Set<Node> availableConnections()
- assumes cost is known
- returns all Connections available

final Set<Node> availableConnections(ConnectionType)
- assumes cost is known
- returns the set of Connections available at this time with the given connection type

class PathState (Route State)

In the process of searching for a path, the path finder keeps an intermediate path state variable

Variables:

Map<Node, PathNode> pathNodes

NavigableSet<PathNode> unreached

Methods:

private PathState(Set<Node> nodes, Node origin)

private PathState(Set<Node> nodes, Node origin, Object obj)

static final PathState of(Set<Nodes> nodes, Node origin)

- Use this constructor when the input object is not needed to get cost

static final PathState of(Set<Nodes> nodes, Node origin, Object obj)

- Use this constructor when an input object is needed to get cost

Both constructors throw:

- Null Pointer Exception if input is null
- Invalid Argument Exception if arguments are invalid in some way (origin is not in the set of nodes, etc.)

void replaceNode(PathNode pathNode)

- replaces the path node for it corresponding node. Assumes the node is in the path state and is unreached
- Should throw null pointer exception if appropriate
- Should throw an illegal state exception if the pathNode is not in unreached

boolean allReached()

- returns true if all nodes are reached

PathNode closestUnreached()

- returns the closest unreached path node and removes it from the unreached set
- Throws invalid state exception if allReached() is true

PathNode pathNode(Node)

- returns the path node corresponding to the given node
- Throws appropriate null pointer exception if needed
- Throws illegal argument exception if Node isn't in the set of nodes

class PathFinder (Route Finder)

Variables:

- private final Set<Nodes> nodes

Methods:

private PathFinder(Set<Nodes> nodes)

- private constructor that sets internal node variable to argument

public static PathFinder of(Set<Nodes> nodes)

- public builder method for constructor
- validates input by ensuring that it is not null (throws appropriate exception)

public final PathNode bestPath

Input: Node start, Node end, ConnectionType

- finds and returns the shortest path from the start node to the end node
- Throws null pointer exception if necessary
- Throws invalid argument exception if appropriate (start/end are not in the map, etc.)

```
pathState <— new PathState in which all nodes have no previous
node and min reach times are unknown, except for the start node,
which has a time of Zero/some departure time value (improve on
this?)

while (there is an unreached node)

   current node <— unreached node with lowest cost

   if (currentNode has unknown connection time)
      Then we don't actually have a working path to it, so return
      null because there is no valid path

   if (currentNode = goal) return currentNode

   for all available paths from "currentNode" with proper
   connectionType:
      if that (destination node's total cost via the path) <
      (previous cost of the node)
         replace that node with a node that has "currentNode" as
         it's "previous"
end while

return null (no route was found)
```

Classes Implemented for Testing Only

These will be the simplest implementation possible of the classes that they extend, so that way we can test the functionality of each interface/abstract class

- SimpleAbbable extends Addable
    - Contains integers for the comparison value
- SimpleConnection extends Connection
    - Connection has a “start” and “end” only