Overview

Goals

The goals of this assignment are to:

* Understand, implement, and use a graph to solve a conceptual problem.
* Design a graph implementation from scratch that solves the problem.
* Implement Dijkstra's algorithm for finding the shortest route in a graph.
* You may use Java's built-in collection types for your collection of nodes and edges. But, grossly inefficient code (exponential growth) may lose points. So, pay attention to whether your collection and algorithm choices are doing work for your program that can and should be avoided.

Background

In this assignment, you will be simulating a simple Navigation Map app, with the help of a Graph class that you write.

A real-life map is modeled as a directed weighted graph with non-negative edge costs in the following way:

* Different locations in the map correspond to distinct vertices in the graph.
* If there is a direct connection/road from one location to another, it manifests as an edge in the graph.
* When a user asks for the route from one location to another, Dijkstra's algorithm for the shortest path is used to retrieve the optimal route (least cost).

In real life, edges or paths often have multiple properties corresponding to them.  
For example, the length of the path is one property, while the estimated time taken to traverse that path, which would depend upon traffic and number of stop signs, is another property. A third property could be the number of tolls on the route, which would contribute to the estimated cost of traversing that path.

Hence, our graph interface supports storing multiple edge properties for each edge.  
The shortest route algorithm is run on one property per request, which the user specifies.

Specifications

Running the Program

The main application class (contains main method) is **MapApp.java**.  
The app starts by building an internal representation of the graph by reading edge information from a file. The path to this file is passed as the only command line argument.  Do not hard code file names.

java MapApp ./samples/sample1.txt

Input File Format

The first line of the input data file contains a header of column names. The remaining lines denote the edges. Each line has the same number of values, as indicated in the header. The file must have at least 3 columns, source, destination and property. The column delimiter in the file is a space character so any "two" word names are entered in camelCase.

Source Destination PropertyName1 PropertyName2 PropertyName3... PropertyNameN  
<srcName> <destName> <dbl prop1> <dbl prop2> ... <dbl propn>  
<srcName> <destName> <dbl prop1> <dbl prop2> ... <dbl propn>  
.  
.  
  
Sample input files are present in the samples folder in p5.zip.

If a negative property value is detected, store one as the property value with no message to the user.  Program testing will focus on valid sample input files. If 负值，读成1

All Classes

The [javadocs](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/javadocs)for the classes provided to you are here. 

**Location**:

This class is used to represent a location object that is identified by a case-insensitive name.   
Do not modify this class.

**Path**:

This class is used to represent a directed edge between two locations. It has a list of properties, each property being of the **Double** data type.   
The number of properties is fixed for a given graph instance, and can be derived from the graph's constructor.  
Do not modify this class.

**GraphADT**:

This is a generic interface to represent a graph and it contains methods to add vertices and edges of generic data types.  
Do not modify this interface.

**NavigationGraph**:

This is the most important class, which you will design and implement.

The implementation is left up to you, though we strongly recommend using the **GraphNode** class to implement an *Adjacency List* representation of the graph. You may add whatever *private* methods you see fit to in addition to implementing all of the required public methods even if your final solution does not use a particular operation.

This class must implement **GraphADT** and create a graph of **Location**s connected by **Path**s.   
i.e., it must not be generic.    
The constructor must accept an array of **String** property names (eg. Time, Cost) that correspond to the edge properties.

public NavigationGraph(String[] edgePropertyNames)

In addition to the **GraphADT** methods, your **NavigationGraph** class must contain the following:

public Location getLocationByName(String name)

These method signatures are provided in the NavigationGraph.java in p5.zip.

**Output Format**

Examples of the output format expected can be seen from the sample run files.

**Error Handling**

If any method in the **NavigationGraph** class is called with invalid parameters, throw an **IllegalArgumentException** with an appropriate message.

e.g. For getShortestRoute(Location src, Location dest, String propertyName),

* getShortestRoute(loc1, loc1, propName) ,
* getShortestRoute(null, loc1,propName),
* getShortestRoute(loc1, loc2,propName) where either of the locations is not in the graph.

These are not the only conditions that we will test error checking on!

**GraphNode**:

As discussed in the readings, one possible representation of a graph is as a collection of GraphNodes.    
We recommend a class **GraphNode<V,E>** that is a class that wraps a **Vertex** object (class V) and a List<E>, which are the *outEdges* leading out from this graph node's vertex. This models the adjacency list representation of a graph.

The **Graph** itself can be represented as any of Java's built-in collection types for the collection of GraphNodes.  Consider: List, Set, HashMap, etc    
e.g., Graph contains a ***private*** list of GraphNodes:  List<GraphNode<Location, Path>>

Note: Whenever you declare a variable of this type, remember to specify the parameter classes!  
e.g.   
GraphNode node1;   
GraphNode<Location, Path> node2;  
node1.getVertexData() will return an instance of class Object  
node2.getVertexData() will return an instance of class Location

**MapApp**:

This is the *driver* class that contains the main method. Most of this class has been implemented for you.

This app must first read/parse the input filename passed as command line argument and create a **NavigationGraph** object.

Note: The property names in the constructor of **NavigationGraph** must match the header line of the input file.

**InvalidFileException**:

This is an Exception which must be thrown when the input file format is invalid.  
Do not modify this class.

Javadocs

See [javadocs](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/javadocs/index.html) for the classes provided to you.

**Other than the optionally overriding the toString() method,   
you may *not* add any other public methods that are not listed in the provided files.    
You may *not* modify any class other than NavigationGraph in any way except where noted as "//TODO".**

**You may add package level (not public, not private) helper classes like GraphNode as desired.**

SampleRuns

[Input1](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/sampleruns/sample1.txt) --> [Output1](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/sampleruns/output1.txt)  
[Input2](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/sampleruns/sample2.txt) --> [Output2](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/sampleruns/output2.txt)  
[Input3](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/sampleruns/sample3.txt) --> [Output3](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/sampleruns/output3.txt)

Steps

After you have read this program page and given thought to the problem we suggest the following steps:

* **Your p5 team will be responsible for a single submission.**  Be sure to get all team members joined to the team by Monday April 24th, 2017.  All teammates are responsible for checking that your submission is complete and that you have submitted the correct version of your work before the due date to ensure that there are no sad faces -- if the one who said they would submit does not submit the correct files successfully.
* Review the [commenting](https://canvas.wisc.edu/courses/23073/pages/program-commenting-guide) and [style](https://canvas.wisc.edu/courses/23073/pages/program-style-guide) standards that are used to evaluate your program.
* You may use the Java development environment of your choice in CS 367. *However, all programs must compile and run on the lab computers for grading.* If you are going to use the CS lab computers, we recommend that you use Eclipse. You may want to review the [Eclipse tutorial](http://pages.cs.wisc.edu/~cs302/labs/EclipseTutorial/) to learn the basics. Note that on the Linux lab computers, you should enter "eclipse&" at the prompt instead of what is described in the tutorial.
* **Download this**[**p5.zip**](http://pages.cs.wisc.edu/~cs367-1/assignments/p5/files/p5.zip)**file to a programming assignment p5 folder that you make.** Unzip to place all files in your p5 program folder.
* Write test classes and test methods to see if the classes and methods that you are defining work as you intend.
  + Test\_Graph class
    - define class with a main method that calls test methods for each public method of the GraphADT type.
    - add code to create several different instances of your NavigationGraph and then checks that the expected results are achieved for as many different types of conditions as you can identify.
      * empty graph
      * one node graph
      * two nodes and one edge
      * 5 nodes and complete
      * multiple connected components (all nodes are NOT connected to all other nodes)
* Work on code that is the least dependent upon other code working first.  A good order for this program is:
  + NavigationGraph
  + MapApp
* There is no need to complete an entire class before testing.  It is best if you work on small parts and completely test those parts before moving on to other parts.
* If you are not using the lab computers to develop your program, make sure you compile and run your program to ensure that it works on the Linux lab computers. You can compile your Java source using javac in a terminal window as in this example:
  + javac \*.java
* Submit your work for grading. Be sure to submit required files on each submission and to double-check that you have submitted the correct file versions.  We will thoroughly test individual classes and run end-to-end tests on your entire program (using the files we provided).

Submitting Your Work

* Late work is not accepted.
* Make sure your code follows the [style](https://canvas.wisc.edu/courses/23073/pages/program-style-guide) and [commenting](https://canvas.wisc.edu/courses/23073/pages/program-commenting-guide) standards.
* Make sure your program runs on the CS Linux workstations (Rm 1366 CS)
* All classes must belong to the default package. (No package declaration at top of class).
* Submit the following files:
  + NavigationGraph.java
  + MapApp.java
  + Any other classes you write in support of the two required public class.  Other classes should not be public or private.  They should be package level access, but without a package declaration so that they are resolved in the default package.
* Note that you must implement the graph data structure yourself, and you will be graded on this. Using existing **graph packages / classes** is not allowed and will lose at a minimum 1/2 of the points for entire assignment.
* Grading will focus on the implementation and working of the GraphADT interface, not just on end to end application correctness.