Traffic Jam in Sarajevo Project

In collaboration with tech enthusiasts who built a glass sphere capable of accommodating a person and is propelled into the air by cannons, I managed to create the algorithm they needed. Specifically, they required an algorithm that would solve the shortest aerial path between locations represented by large initial letters of the alphabet.

My first task was to create a directed weighted graph capable of loading a file containing values such as:

'A B 10'

'B C 20'

'B D 30'

and so on. In this case, A would represent one location, B another location, and the number 10 would represent the time in seconds needed to travel from location A to location B.

I decided to create a graph using a matrix, which turned out to be both the path of greater resistance and the path of 'foolishness'. However, I challenged myself.

Firstly, from the input, I loaded all existing locations (places.txt) and created a hash map for them so that they could be accessed by letters (which is also the function of a hash map). However, the value of the key is not a String but a Place object, which will contain the name that would otherwise be represented by a string and PlaceID representing the ID of the place, which will actually be the index in the matrix. The same applies to constraints.

The project was implemented with two singletons. If you are not familiar with what a singleton is, it is a kind of global state management in Java (actually, it's just a design pattern).

One singleton stores all the locations that exist, while the other stores the constraints of those locations. The data from both singletons is needed in multiple functions and classes, which is why they are stored in singletons.



To create the graph, a simple command is used in the main function, which is createGraph. This command passes the edgeList to the Graph class, from which the matrix is created.

The edgeList is obtained by loading the input file for the graph, and then a certain degree of manipulation is performed on it, where each line is transformed into an edge and then pushed into the edgeList.

An Edge is actually an object that contains the start of the connection between two locations, the end of that connection, the duration of the connection, the constraint, and the name of the constraint.



In the constructor of the graph, the necessary values are initialized, and the matrix is created. At the same time, in that part, constraints' probability is immediately set, and it is determined whether a certain edge will fail or not.



In the Graph class, there is a createPlacesSet method that takes an edgeList and creates a set of unique values to precisely create the matrix. Otherwise, there would be a bug where it would create one element in the matrix for each edge.

The main code in this program is the Dijkstra's algorithm, which is used to calculate the shortest path. For simplification, I don't plan to include images because almost every Dijkstra implementation is identical.

For Dijkstra, I created a class called ShortestPath with three functions: shortestPathBetweenTwoVertices, \_shortestDistanceToAllVertices, and timeRequiredBetweenAllPlaces

\_shortestDistanceToAllVertices is indeed the most crucial one as it is the function that performs Dijkstra's algorithm and calculates the shortest path. shortestPathBetweenTwoVertices calculates the shortest route between two elements based on its algorithm.



timeRequiredBetweenAllPlaces calculates routes between all locations and writes them to a file.



To sum up the main functionalities of this program. There are also a couple of classes that are not so crucial to mention (for the sake of simplicity and abstraction).

It's important to note that the entire algorithm is covered by tests, which proves its reliability and trustworthiness.

For future updates, it is definitely necessary to eliminate the matrix and use an adjacency list or a similar method, as well as better organize the functions. All in all, the program does what is required of it in a neat and good way.