

Graph:

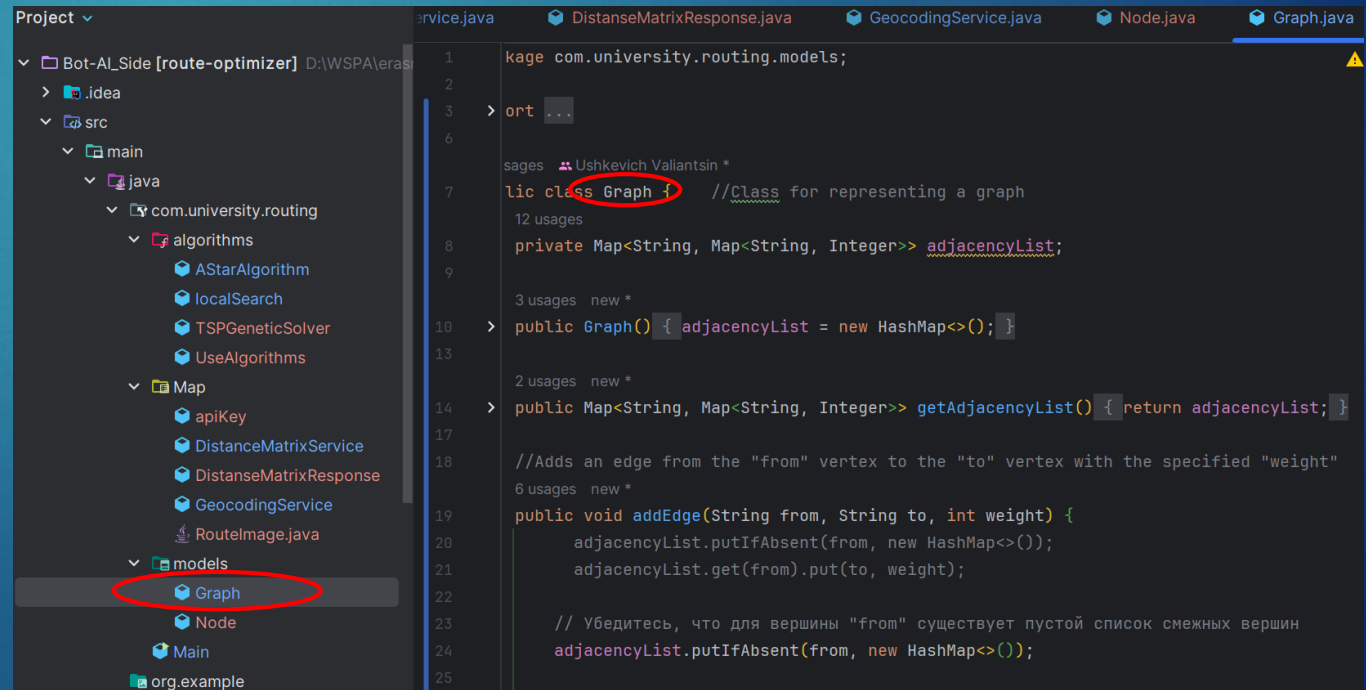
The Graph class represents the full graph structure.

This is a set of vertices (Node) and connections between them (edges with weights).

The graph is implemented through a contiguous list **adjacencyList** : a map where the key is the vertex id, and the value is the neighbours of this vertex and the distance to them.

► Key methods:

1. **addEdge(from, to, weight)**: adds an edge between the vertices from and to with the specified weight. If the reverse edge already exists, it is used.
2. **getDistance(from, to)**: returns the weight of an edge between two vertices.
3. **getNeighbors(node)**: returns all neighbors of the specified vertex.
4. **getNodes()**: returns the set of all vertices of the graph.



The screenshot shows an IDE with a project structure on the left and the source code of the `Graph` class on the right. The project structure includes a package `com.university.routing` with sub-packages `algorithms` and `Map`, and a `models` package containing `Graph` and `Node`. The `Graph` class is highlighted in the project structure. The source code of `Graph` is as follows:

```
1 package com.university.routing.models;
2
3 import ...
4
5
6
7 //Class for representing a graph
8 public class Graph {
9     private Map<String, Map<String, Integer>> adjacencyList;
10
11     3 usages new *
12     public Graph() { adjacencyList = new HashMap<>(); }
13
14     2 usages new *
15     public Map<String, Map<String, Integer>> getAdjacencyList() { return adjacencyList; }
16
17     //Adds an edge from the "from" vertex to the "to" vertex with the specified "weight"
18     6 usages new *
19     public void addEdge(String from, String to, int weight) {
20         adjacencyList.putIfAbsent(from, new HashMap<>());
21         adjacencyList.get(from).put(to, weight);
22
23         // Убедитесь, что для вершины "from" существует пустой список смежных вершин
24         adjacencyList.putIfAbsent(from, new HashMap<>());
25     }
26 }
```