**HOMEWORK 2**

The class ***TraceAnalyser*** is written to analyze and compare graphs. It uses the class ***Dijkstra*** which is not implemented yet. For the purpose of testing, the class ***Dijkstra*** has to be replaced with test double. Pay attention that in future the real implementation of ***Dijkstra*** has to be used without additional changing of code of ***TraceAnalyser***.

Graph information is saved in the database in the table ***graphs***.

Description of functions of the class ***Dijkstra*** can be found in Javadoc (folder **doc** inside the project).

**Unit testing** in Java

1. Class under testing ***TraceAnalyser*** depends on the class ***Dijkstra*** and on the database. Refactor the code of the class ***TraceAnalyser*** (from the attached project **TraceAnalyser**) to isolate these dependencies. Take care that functionality was not changed!
2. Write a complete Junit test suit for the methods ***calculateDiameter(), firstGreater()*** and ***buildGraph()*** by using constructor dependency injection approach (assume that you have no real implementation of the class ***Dijkstra***; test cases have to be independent from the database).

Present test case results in a table of the following structure:

|  |  |  |
| --- | --- | --- |
| Test case name | Description | Result (pass/failed) |
|  |  |  |

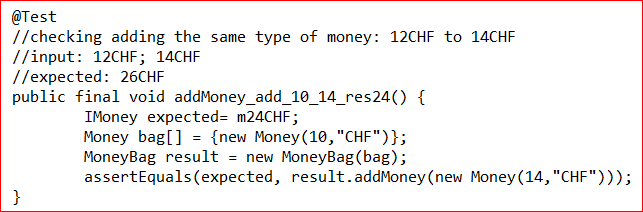
1. Using TDD approach implement the boolean function

***isClose(****sourceNode, evaluationNode, Threshold****)***

of the class ***TraceAnalyser*** that check if there exists a path from *sourceNode* to *evaluationNode* of length less than the *Threshold*.

If there is no path from *sourceNode* to *evaluationNode*, the function throws an exception.

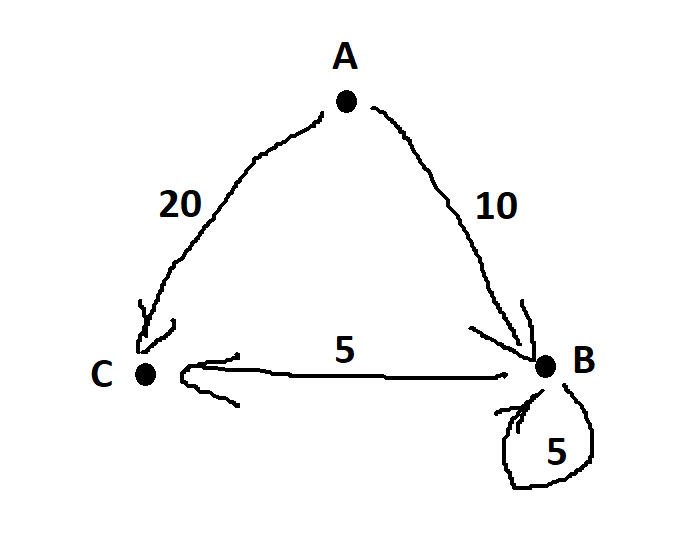
**Note1: Don't forget code notes about performed changes and test cases description according to the following example:**



**Note2:** The table **graphs** is saved in the backup file hw2.sql.

**Note3:** Alpha characteristic of the graph is calculated as follows:

where *N* is a number of nodes in the graph; *D* is diameter of the graph (the maximal length of shortest paths between every two nodes in the graph) and *L* is a number of simple loops in the graph.



**Note 4:** In the graph G1 from the database:

the result of calculation of

Dijkstra.*calculateShortestPathFromSource*(g, A)

will be the graph structure, containing:

A.distance = Integer.***MAX\_VALUE***

B.distance = 10

C.distance = 15

**Format of submission:**

Homework is submitted on Moodle as a ".zip" file called G<#>-<first submitter ID1>\_<second submitter ID2>\_HW2.zip or G<#>-<submitter ID>.zip if you are alone.

The file has to include:

* Zip file including full Eclipse project

It has to include **full eclipse project** and not single files like <>.java or <>.jar.

* Word/Excel document including test case results in a table of the structure presented above.

**Notes:** Delay in submission will be fined 5% of the score for each day of delay.

**Submission date**: 24.05.2021