Collibra Backend Interview Exercise

Rules of Engagement

- The exercise must be completed and submitted within 1 week maximum of receiving the assignment.
- Collibra delivers a JVM-based test client that acts as the final testing framework.
- Needs to run on a Java SE Runtime Environment 8.
- Within the above boundaries, all libraries or languages can be used.
- The exercise is divided into multiple phases. The test client will test these phases one by one. It is not mandatory (but recommended) to finish all the phases.
- Contact dev-backend-interview@collibra.com in case of questions.

Testclient

The testclient can be ran by doing: **java -jar testclient.jar**
This will show the different test phases and if they pass or not.
To show debug output for the testclient you can do: **java -jar testclient.jar debug**

Deliverable

- The code
- The full built package with all dependencies needed to run on a plain JVM.
- A readme file on how to execute and optional comments

Requirements

General

- The goal is to implement a simple string protocol communicating over sockets. Below, all messages coming from the server (your code) are shown in **bold red**, all client messages are shown in **bold blue**.
- The client will only send and understand standard ASCII characters.
- Every command or response in the protocol is delimited by a newline character, also known as '\n' or ASCII 10.

PHASE 1

For the first phase, setup the TCP socket server application listening on port 50000 and handling the simple commands for greetings and saying goodbye as explained below.
Only one session will be active at a certain time.

When the client doesn't send a message for more than 30 seconds, the connection should time-out and the session should be closed. When closing the session, the server should send a last goodbye message to the client (see later).
Only the current session should be closed. The server should remain ready to accept new sessions.

On connection, the server starts a new session and generates a unique ID for the session. This ID should be a valid UUID.

The server sends a first message back to the client:

HI, I'M <session-id>

Where the <session-id> is replaced by the generated session id.

The client will then respond with:

HI, I'M <name>

Where <name> can be any string of alphanumeric characters + the dash character (-).

The server will respond with:

HI <name>

To end the session, the client will send:

BYE MATE!

After which the server will respond with

BYE <name>, WE SPOKE FOR <X> MS

Where <name> is the name the client mentioned in its first message and <X> is the number of milliseconds the session.
Note: this message should also be sent to the client when the session times out (after not receiving any message from the client for 30 seconds).

When a command is sent which is not supported, the server should respond with:

SORRY, I DIDN'T UNDERSTAND THAT

PHASE 2

For the second phase, we'll be adding a few commands to be handled by the server. These commands are for building a directed graph. The graph is shared amongst all the sessions during the lifetime of the running server. No persistence on disk is needed.
These commands can occur 0 or multiple times in any order.

Commands

ADD NODE <X>

Adds a new node with name <X> to the graph. The response must be:

- If succeeded: **NODE ADDED**
- If already exists: **ERROR: NODE ALREADY EXISTS**

Note: node names will always contain only alphanumeric characters + the dash character (-).

ADD EDGE <X> <Y> <WEIGHT>

Adds an edge from node <X> to node <Y> with the given <WEIGHT>. The <WEIGHT> is a simple integer indicating the weight of the edge.
The response must be:

- If succeeded: **EDGE ADDED**
- If a node not found: **ERROR: NODE NOT FOUND**

Note: edges can occur multiple times (even with the same weight).
Note: the weights, will always be non-negative positive integers.

REMOVE NODE <X>

Removed node <X> from the graph. The response must be:

- If succeeded: **NODE REMOVED**
- If not found: **ERROR: NODE NOT FOUND**

Note: this should also cleanup all the edges that are connected to this node.

REMOVE EDGE <X> <Y>

Remove all the edges from node <X> to node<Y>.
The response must be:

- If succeeded: **EDGE REMOVED**
- If a node not found: **ERROR: NODE NOT FOUND**

Note: edges are directed and cannot be traversed in the other direction.
Note: no error should be given if there were no edges from node X to node Y. The operation should just success.

PHASE 3

For the third phase, we add the calculation of the shortest path between two nodes in the directed graph.

Commands

SHORTEST PATH <X> <Y>
Finds the shortest (weighted) path from node <X> to node <Y>.
The response must be:

- If succeeded: **<WEIGHT>**
- If a node not found: **ERROR: NODE NOT FOUND**

Where <WEIGHT> is the sum of the weights of the shortest path.

Note: when there is no connection between the two nodes. Integer.MAX_VALUE must be returned.

PHASE 4

For the fourth phase, we add the calculation to find the nodes which are 'closer' to the given node than the given weight.

Commands

CLOSER THAN <WEIGHT> <X>
Finds all the nodes that are closer to node X than the given weight. The response must be:

- If succeeded: **<NODES>**
- If a node not found: **ERROR: NODE NOT FOUND**

Where <NODES> is a comma separated list (no spaces) of the found nodes, sorted alphabetically by name, not including the starting point

For example:
Support you have this very simple graph: Mark - 5 -> Michael - 2 -> Madeleine - 8 -> Mufasa
CLOSER THAN 8 Mark
Would return
Madeleine,Michael
Because Michael is at weight 5 from Mark and Madeleine is at weight 7 (5+2) from Mark.

PHASE 5 & 6

For the fifth and sixth phase, the test client will open multiple sessions at the same time, all adding and removing nodes and edges.
At the end (6th phase) it will do some random checks with the ‘shortest path’ and ‘closer than’ commands to check the consistency of the graph (also multiple in parallel).