Distributed Programming II

A.Y. 2016/17

Assignment n. 2

All the material needed for this assignment is included in the *.zip* archive where you have found this file. Please extract the archive to an empty directory (that will be called *`[root]`*) where you will work.

The assignment consists of developing a client for a RESTful web service named *Neo4JXML*, using the JAX-RS framework. The service, which is based on NEO4J (a graph-oriented DB), provides the possibility to create a graph, stored by NEO4J, by creating single *nodes* and *relationships*, and the possibility to query for the *paths* that connect a given source node to a given destination node in the graph that has been created. Each node can have properties (i.e. name-value pairs) and labels (i.e. strings).

The documentation about the available operations provided by the Neo4JXML service is detailed in the document file `[root]/neo4jxmlRESTapi.pdf.`

The service provider can be started on your own local machine (under Windows) by starting, firstly, the NEO4J database service and deploying, then, the Neo4JXML service.

The NEO4J database can be installed and started by: 1) downloading the *.zip* archive of the Neo4j service from the official website (https://neo4j.com/download/community-edition/); 2) extracting the archive to an empty directory (that will be called`[neo4j]`); 3) modifying the default authentication settings of Neo4j by uncommenting part of the Neo4J configuration file `[neo4j]/conf/neo4j.conf` (i.e, `dbms.security.auth_enabled=false`); 4) running the Neo4j service by issuing the following command (from the `[neo4j]/bin/` directory):

```
$ neo4j.bat console
```

The *Neo4JXML* service can be deployed and started into your instance of servlet container (e.g., Tomcat 8, installed under the `[catalina_home]` folder) by: 1) copying the .war file into the `webapps` folder under the Tomcat 8 webapps folder (i.e., `[catalina_home]/webapps`); 2) launching Tomcat by issuing the following command (from the `[catalina_home]/bin` folder):

```
$ startup.bat
```

Alternatively, Tomcat 8 can be started from XAMPP. Optionally, you can deploy/un-deploy/start/stop web-services through the Tomcat web interface at http://localhost:8080. In order to enable the management through the web interface, modify the Tomcat Manager configuration by editing the `[catalina_home]/conf/tomcat-users.xml` and adding the following lines of code under the `tomcat-user` tag:

```
<role rolename="tomcat"/>

<role rolename="role1"/>

<user username="root" password="root" roles="tomcat,manager-gui"/>

<user username="both" password="tomcat" roles="tomcat,role1"/>

<user username="role1" password="tomcat" roles="role1"/>
```

The client to be developed has to a) read the information about a set of NFFGs from the random generator already used in Assignment 1; b) load one of these NFFGs into NEO4J by means of *Neo4JXML*; c) test reachability between pairs of nodes in the loaded NFFG.

The selected NFFG has to be loaded into NEO4J by means of *Neo4JXML* as follows: a node has to be created for each NFFG node, with a property named “name” whose value is the NFFG name; a relationship has to be created for each link of the NFFG, connecting the corresponding nodes; at any time, only the nodes of a single NFFG must be present in the NEO4J DB (before loading a new NFFG, the client must delete all nodes that were present previously.

The client to be developed must take the form of a Java library that implements the interface `it.polito.dp2.NFFG.lab2.ReachabilityTester`, available in source form in the package of this assignment. This interface enables the operations a), b) and c) mentioned above. More precisely, the library to be developed must include a factory class named `it.polito.dp2.NFFG.sol2.ReachabilityTesterFactory`, which extends the abstract factory `it.polito.dp2.NFFG.lab2.ReachabilityTesterFactory` and, through the method `newReachabilityTester()`, creates an instance of your concrete class that implements the `ReachabilityTester` interface. All the classes of your solution must belong to the package `it.polito.dp2.NFFG.sol2` and their sources must be stored in the directory `[root]/src/it/polito/dp2/NFFG/sol2.`

If you want, you can use automatically generated classes for developing your solution. In this case you have to create an ant script named `sol_build.xml` and placed in `[root]`, with a target named `generate-artifacts`. This target, when invoked, must generate the source code of the classes in the folder `[root]/gen-src`. The main ant script `build.xml` will automatically call your script before compiling your solution and will automatically compile the generated files and include them in the classpath when running the final tests. Note that the generated classes must also belong to the same package as the solution. Custom files needed by your solution or ant script (e.g. a schema) have to be stored under `[root]/custom`.

An instance of the client library, when created, must read the information about the set of NFFGs from the random generator as already done in Assignment 1, then it must respond to method invocations.

The actual base URL used by the client class to contact the service must be customizable: the actual URL has to be read as the value of the `it.polito.dp2.NFFG.lab2.URL` system property.

The client classes must be robust and interoperable, without dependencies on locales. However, these classes are meant for single-thread use only, i.e. the classes will be used by a single thread, which means there cannot be concurrent calls to the methods of these classes.

Correctness verification

Before submitting your solution, you are expected to verify its correctness and adherence to all the specifications given here. In order to be acceptable for examination, your assignment must pass at least all the automatic mandatory tests. Note that these tests check just part of the functional specifications! In particular, they only check that the client behaviour is consistent with the data received from the data generator and with the status of the server in some scenarios.

Other checks and evaluations on the code will be done at exam time (i.e. passing all tests does not guarantee the maximum of marks).

The *.zip* file of this assignment includes a set of tests like the ones that will run on the server after submission.

The tests can be run by the ant script included in the *.zip* file, which also compiles your solution. Of course, before running the tests you must have started the server as explained above. Then, you can run the tests using the `runFuncTest` target, which also accepts the `–Dseed` and `–Dtestcase` options for controlling the random generation of data. Note that the execution of these tests may take up to 1-2 minutes when successful, and during the junit tests no output is produced. It is recommended that, before trying the test provided in the package, you create a main that lets you try the single calls to your client library and lets you debug your library.

Submission format

A single *.zip* file must be submitted, including all the files that have been produced. The *.zip* file to be submitted must be produced by issuing the following command (from the `[root]` directory):

```
$ ant make-final-zip
```

Do not create the *.zip* file in other ways, in order to avoid the contents of the zip file are not conformant to what is expected by the automatic submission system.