Distributed Programming II

A.Y. 2017/18

Assignment n. 2

All the material needed for this assignment is included in the *.zip* archive where you have found this file, except the jars of the JAX-RS libraries and related dependencies that are provided separately. Please extract the archive to an empty directory (that will be called `[root]`) where you will work.

The additional jar files necessary for the assignment are available in the course web site. In the Labinf machines they are already installed under `/opt/dp2/shared/lib`. When setting the build path of your project, add all these jar files, in addition to the ones under `[root]/lib`. Also, you are adviced to attach the source jars found under `[root]/lib-src` to the the `junit-4.5` and `javax.ws.rs-api` libraries (right click on library, Properties, Java Source Attachment).

The assignment consists of developing a client for a RESTful web service named *Neo4JSimpleXML*, using the JAX-RS framework.

A documentation about the Neo4JSimpleXML service is available in the document file `[root]/Neo4JSimpleXML.pdf`.

The client to be developed must be able to: a) read the information about a set of NF-FGs and IN from the random generator already used in Assignment 1; b) load the nodes of these NF-FGs, the IN hosts, and their relationships into NEO4J by means of *Neo4JSimpleXML* as specified below; c) provide reachability information, by properly getting this information from *Neo4JSimpleXML* as specified below.

The nodes of an NF-FG have to be loaded into NEO4J by means of *Neo4JSimpleXML* as follows: a graph node has to be created for each **NF-FG node**, with a property named "name", whose value is the **NF-FG node name**, and label "Node"; a relationship has to be created for each **link** of the NF-FG, connecting the corresponding nodes, **with type "ForwardsTo"**.

The hosts of the IN have to be loaded into NEO4J by means of *Neo4JSimpleXML* as follows: a graph node has to be created for each IN **host**, with a property named "name", whose value is the **host name**, and label "Host"; a relationship has to be created for each **NF-FG node** allocated on a IN **host**, connecting the corresponding nodes of the graph, **with type "AllocatedOn"**.

Only the above mentioned data have to be stored in NOE4J. Any additional information, and the IDs of the graph nodes (necessary to perform further operations on the graph), have to be stored in the client.

The client to be developed must take the form of a Java library that implements the interface `it.polito.dp2.NFV.lab2.ReachabilityTester`, available in source form in the package of this assignment, **along with its documentation**. 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.NFV.sol2.ReachabilityTesterFactory`, which extends the abstract factory `it.polito.dp2.NFV.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.NFV.sol2` and their sources must be stored in the directory `[root]/src/it/polito/dp2/NFV/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`. An empty version of this file is already present in the Assignment zip archive. The main ant script `build.xml` will automatically call this 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 (if using WADL to java generation, in the ant task call you have to specify `autoSchemaPackage="false"` in order to force the package). 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 NF-FGs and IN 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.NFV.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. When developing the client, consider that the operation that gets the reachable nodes is a time-consuming one.

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. Before running the tests you must have started the servers as explained in the service documentation file. 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 some minutes when successful, depending on the seed, 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-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.