Distributed Programming II

A.Y. 2016/17

Final Test 1

All the material needed for this test is included in the folder where you have found this file. Copy your solution of Assignment 3 into this folder.

The test consists of a programming exercise (6 points) and a question (4 points). The exam can be passed only if the programming exercise solution passes the mandatory tests (at least test `testCachingBehavior`). In this case, the proposed mark will be the sum of the points got for the test and a base evaluation mark in the range 16-20, assigned on the basis of the evaluation of the submitted assignments (16 points granted because your assignments passed the mandatory tests at submission time and 4 extra points based on the evaluation of one extra aspect of your assignments).

Programming exercise

1. Modify your service developed in Assignment 3 so that it can perform the caching of policy verification results. At any time, caching can be enabled or disabled. When caching is enabled and a client requests the verification of a policy already loaded in the service, the service checks whether a verification result is available for that policy. If a verification result is available with a verification time strictly greater than the last update time of the NFFG the policy refers to, the service returns the available verification result, without computing a new verification result. In this case, the verification time of the policy is not updated. If instead a verification result with a verification time strictly greater than the last update time of the NFFG the policy refers to is not available, the service computes a new verification result for the policy and updates the old verification result. In this case, the verification time of the policy is also updated and set to the current time. If necessary, modify your service so that it also caches the NFFG names, the NFFG node names and the NFFG update times (i.e., this information must be available in the NffgService). The modified service must also offer the possibility for clients to enable or disable caching. The current status of caching represented as plain text (ASCII string "enabled" or ASCII string "disabled") should be readable via a GET of an extra resource of your service, with URL http://localhost:8080/NffgService/rest/caching. The same resource should be readable also with an application/xml representation chosen by you.

    All the other features of the service must remain unchanged.

2. Create a new client for your service, named client3, which implements the interface `it.polito.dp2.NFFG.lab3.test1.NFFGClient3`, given in source form (the interface is self-explaining, look at the comments). Create a factory class for your client3 named `it.polito.dp2.NFFG.sol3.test1.client3.NFFGClient3Factory` that extends the abstract factory `it.polito.dp2.NFFG.lab3.test1.NFFGClient3Factory` and, through the method `newNFFGClient3()`, creates an instance of your concrete class that implements the `it.polito.dp2.NFFG.lab3.test1.NFFGClient3` interface. All the classes of your client3 must be in the package `it.polito.dp2.NFFG.sol3.test1.client3`.

Apart from these new specifications, all the specifications given for Assignment 3 still apply.

Modify the *ant* script `[root]/sol_build.xml` so that it includes the compilation of your new client. As usual, you can assume that, when your client is compiled and run, your web service has already been deployed.

Question

Explain if and how, in the code of your service, you avoid race conditions on the representation of the status of caching and on the cached verification results.

The answer to this question must be written in a text file named `[root]/answer.txt`

Correctness verification

In order to pass the exam you have to implement at least points 1. and 2., and your solution must pass at least the mandatory tests that are included in the archive (the original tests of Assignment 3 plus the test `testCachingBehavior` from the additional junit test suite `it.polito.dp2.NFFG.lab3.test1.tests.NFFGTests1`). These tests can be run by the ant script `buildTest1.xml` included in the *.zip* file, which also compiles your solution, packages and deploys your service to Tomcat, and runs your clients.

The Tomcat server and Neo4J must be both running when the tests are launched. In particular, start Tomcat from the graphical interface of Xampp, in order to have the proper user configuration (i.e., user `root` – password `root` - with role "`manager-gui, manager-script`" at least). The Neo4J database, instead, must be started from command line. A distribution of the data-base is already available and properly configured under the `Z:\ neo4j-community-3.1.1` folder. The command to run Neo4J is (from the Neo4j folder):

`neo4j.bat console`

The command for running only the tests specific for this assignment (with the exclusion of the original ones of Assignment 3) is

`ant -f buildTest1.xml run-tests -Dseed=XXXX`

The total number of junit tests in this case is 3.

The full set of tests (including the original ones of Assignment 3) can be launched by running the `run-tests-full` target:

`ant -f buildTest1.xml run-tests-full -Dseed=XXXX`

As this target may take a long time to complete, initially you are suggested to use the `run-tests` target.

Submission format

A single *.zip* file must be submitted, including all the files that are part of your solution (including the files of your solution of Assignment 3 that you are re-using). The *.zip* file to be submitted must be produced by issuing the following command (from the `[root]` directory):

`ant -f buildTest1.xml make-final-zip`

Important: check the contents of the zip file named `solution.zip` after having run the command!