Distributed Programming II

A.Y. 2015/16

Assignment n. 2 – part b)

All the material needed for this assignment is included in the *.zip* archive where you have found this file. Please extract the archive to the same `[root]` directory where you have extracted the archive for part a). The material is similar to the one of Assignment 1. After having extracted the archive, make sure that the files developed in part a) of this Assignment (`WFInfo.xsd`, `doc.txt`, and `WFInfo.xml`) remain under the `[root]/xsd` directory.

The assignment consists of two sub-parts:

1. Using the *JAXB* framework, write a *Java* **application** for serialization called `WFInfoSerializer` (in package `it.polito.dp2.WF.sol2`) that serializes the data about the DP2-WF workflow management system into an *XML* file **valid** with respect to the *schema* designed in Assignment 2 part a). As already done for Assignment 1, the *XML* file must include all the information that can be retrieved by accessing the interfaces defined in package `it.polito.dp2.WF` and the implementation of the interfaces to be used as data source must be selected using the "abstract factory" pattern: the `WFInfoSerializer` application must create the data source by instantiating `WorkflowMonitorFactory` by the static method `newInstance()`. The application must receive the name of the output *XML* file on the command line as its first and only argument.

   All the sources of the application must be stored under the directory `[root]/src/it/polito/dp2/WF/sol2/`. The ant script provided with the assignment can be used to compile and run the application exactly as with Assignment 1. In addition, this script includes a target called `generate-bindings`, which must be used to generate the *Java* bindings for the developed *schema*. Using this target, the files will be generated into the `generated` directory. The target will be called automatically when calling `build` and `WFInfoSerializer`.

2. Using the *JAXB* framework, write a *Java* **library** that can be used to load and validate an *XML* file like the one generated by the program developed in the previous part of the assignment. As in Assignment 1, the library must be robust enough to be used within a server: it must consider the input document as "unreliable" (being something that comes from a public network), and it must never throw runtime exceptions (such as for example `NullPointerException`). The library must implement all the interfaces and abstract classes defined in the package `it.polito.dp2.WF`, returning the data loaded from the file. The library must be entirely in the package `it.polito.dp2.WF.sol2` and its sources must be stored in the `[root]/src/it/polito/dp2/WF/sol2/` directory. As in Assignment 1, the library must include a factory class named `it.polito.dp2.WF.sol2.WorkflowMonitorFactory`, which extends the abstract factory `it.polito.dp2.WF.WorkflowMonitorFactory` and, through the method `newWorkflowMonitor()`, creates an instance of your concrete class that implements the `WorkflowMonitor` interface. The name of the *XML* input file must be obtained by reading the `it.polito.dp2.WF.sol2.WorkflowInfo.file` system property.

   To build the library, use the command:

   ```
   $ ant build
   ```

   which automatically calls the target `generate-bindings`. If this command fails, check

that you have strictly followed all the specifications in this assignment.

The serializer and the library must be portable and interoperable, even when executed in a distributed environment (there must be no dependency on the local machine, location, and settings).

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 include both parts and 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 `WFInfoSerializer` application generates well-formed and valid *XML* files (with respect to the *schema*).
- the data stored by the `WFInfoSerializer` application in the output *XML* file are loaded by the classes of the library developed in sub-part 2 without errors.
- the chain *serializer+library* does not alter data (if the library receives an *XML* file generated by the serializer, the data extracted by the library are the same that were given to the serializer for the generation of that file).

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).

All the automatic tests use the random data generator provided with this assignment. The *.zip* file of this assignment includes a set of tests like the ones that will run on the server after submission. The tests are the same used for Assignment 1 and they have the same test cases. Tests can be run by the same command used for Assignment 1.

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. Note that the *.zip* file **will not** include the files generated automatically.