Distributed Programming II

A.Y. 2018/19

Assignment n. 1 – 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]` working directory where you have extracted the archive for part a), so that the files developed in part a) of this Assignment remain under `[root]`. In particular, make sure that the file `doc.txt` remains under `[root]` and that the files `rnsInfo.xsd` and `rnsInfo.xml` remain under `[root]/xsd`.

The assignment consists of two sub-parts:

1. Using the *JAXB* framework, write a ***Java* application** for serialization called `RnsInfoSerializer` (in package `it.polito.dp2.RNS.sol1`) that serializes the data about the DP2-RNS system into an *XML* file **valid** against the *schema* designed in Assignment 1 part a). The output *XML* file must include all the information that can be retrieved by accessing the interfaces defined in package `it.polito.dp2.RNS`, and the implementation of the interfaces to be used as data source must be selected using the "abstract factory" pattern: the `RnsInfoSerializer` application must create the data source by instantiating `it.polito.dp2.RNS.RnsReaderFactory` by means of its static method `newInstance()`. The application must receive the name of the output *XML* file on the command line as its first and only argument.

   In practice, the application can be developed by modifying the sample *Java* application `it.polito.dp2.RNS.lab1.RnsInfo` (provided in source form in the *.zip* archive), which instantiates the interfaces as specified above, reads information from these interfaces and outputs it to the standard output. Your application can read the information in the same way but it must output it to a valid *XML* file (whereas the `RnsInfo` application outputs information to standard output). Note that the order for data serialization may change in the two applications. `RnsInfo` can be run from the `[root]` directory by running the provided ant script (which also compiles, adjusts classpaths, includes libraries as necessary, and sets the system property for specifying the data source that implements the interfaces). This can be done from the command line, by issuing the command

   ```
   $ ant RnsInfo
   ```

   By running the application in this way, a pseudo-random data source is used, included in one of the jar libraries provided with the assignment. By default, the pseudo-random data generator generates a random number of places and vehicles. The behavior of the generator can be changed by passing other command-line parameters, as shown in the following example:

   ```
   $ ant -Dseed=X -Dtestcase=Y RnsInfo
   ```

   where X is the seed of the pseudo-random generation engine (an integer number different from 0) and Y is the test case, which can be 0, 1, 2. When the test case is 0, a small number of places is generated with no vehicles. When the test case is 1, the number of places is still small, but there are vehicles. When the test case is 2 a larger number of places and vehicles is generated. The default is test case 1.

   All the sources of the application must be stored under the directory `[root]/src/it/polito/dp2/RNS/sol1/`. The ant script provided with the assignment material can also be used to generate the bindings, compile and run the developed application.

The command for generation of the bindings from your schema is

```
$ ant generate-bindings
```

The files will be generated into the `gen` directory.

The command for compilation is

```
$ ant build
```

whereas the command for execution is

```
$ ant -Doutput=file.xml RnsInfoSerializer
```

This command also calls the targets for binding generation and compilation. Of course, `file.xml` is the selected output file name. When running the developed application in this way, the pseudo-random data source is used, and the `seed` and `testcase` parameters can be set, as already shown for the `RnsInfo` program (setting the seed is useful for being able to repeat tests during debugging, otherwise the seed is selected randomly at each run). If the ant commands for compilation and execution fail, probably you did not follow the specifications given in the assignment strictly.

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. 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.RNS`, returning the data loaded from the file. The library must be entirely in the package `it.polito.dp2.RNS.sol1` and its sources must be stored in the *`[root]`*`/src/it/polito/dp2/RNS/sol1/` directory. The library must include a factory class named `it.polito.dp2.RNS.sol1.RnsReaderFactory`, which extends the abstract factory `it.polito.dp2.RNS.RnsReaderFactory` and, through the method `newRnsReader()`, creates an instance of your concrete class that implements the `RnsReader` interface. The name of the *XML* input file must be obtained by reading the `it.polito.dp2.RNS.sol1.RnsInfo.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

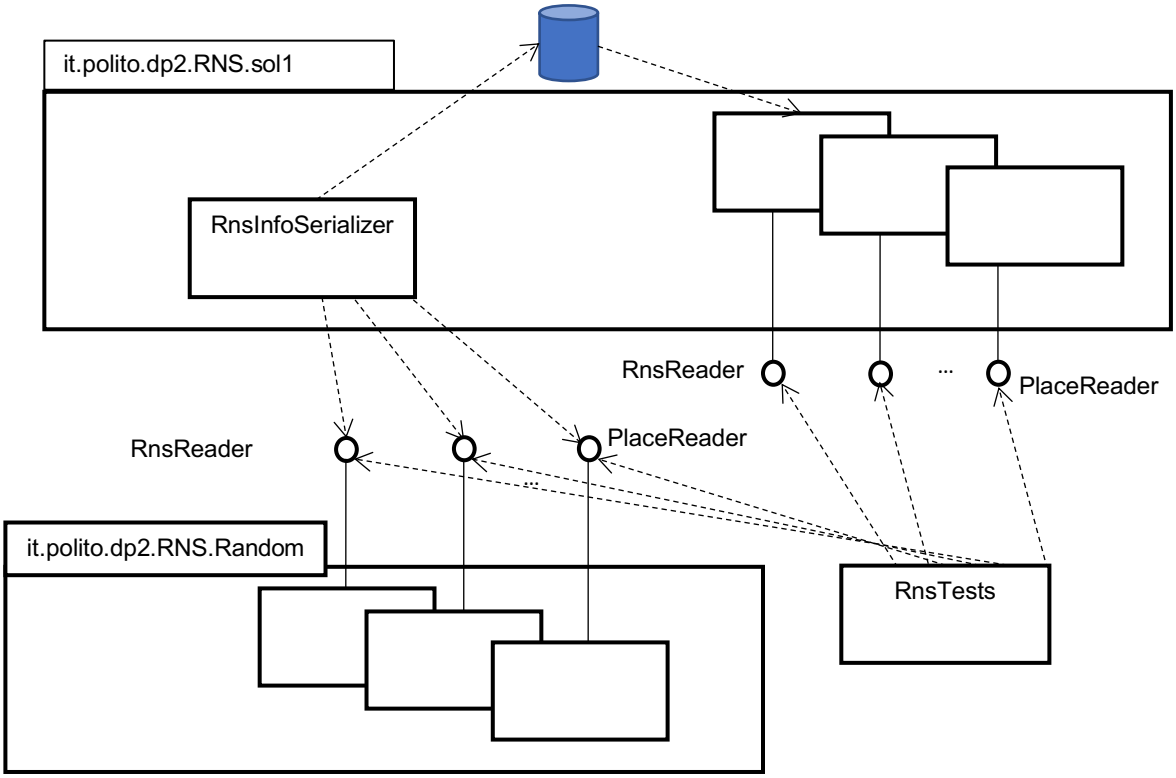
Before submitting your solution, you are expected to verify its correctness and adherence to all the specifications given here. In order to be acceptable for examination, your assignment must include both parts and must pass at least all the automatic mandatory tests. Note that these tests check just part of the functional specifications! In particular, they only check that:

- the `RnsInfoSerializer` application generates well-formed and valid *XML* files (with respect to the *schema*).

- the data stored by the `RnsInfoSerializer` 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 (e.g. programming style, adherence to guidelines) will be done at exam time (i.e. passing all tests does not guarantee the maximum of marks).

The way the automatic tests are organized is shown in the following figure:



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 have been written using the *Junit* system for unit testing. Their sources are available in the *.zip* file, package `it.polito.dp2.RNS.lab1.tests`.

Four different test cases can be used: 0, 1, 2, 3. Test cases 0, 1 and 2 differ only in the generated data (see what already said about the pseudo-random generator), while test case 3 uses the same data generated for test case 1 but it includes the extra check of time values (not checked with the other test cases). In order to be admitted to the final test it is mandatory to pass the tests with test cases 0 and 1. These tests will run on the server on submission, with two different seeds.

In order to locally run one of the test cases on your machine, you can issue the following `ant` command:

```
$ ant -Dtestcase=X -Dseed=Y runFuncTest
```

where `X` is the number of the test case (0, 1, 2, or 3) and `Y` is the seed for the pseudo-random data generator. If not specified, the seed is automatically computed from the current time. The results of the junit tests can be displayed graphically by double clicking on the `testout.xml` file in Eclipse.

Before trying the automatic tests, it is suggested that you test your applications thoroughly by yourself, by running your serializer (*ant* command with target `RnsInfoSerializer`) separately. Remember that automatic tests are partial, and that further evaluations (e.g. about robustness, portability, etc.) will be done at exam time on your solution.

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