# Distributed Programming II

A.Y. 2014/15

## *Assignment n. 1 – part b)*

All the material needed for this assignment is included in the *.zip* archive where you have found this file. Unzip the archive into the same working directory where you have unzipped the archive for part a), so that the files developed in part a) of this Assignment (`flightInfo.dtd`, `doc.txt`, and `flightInfo.xml`) remain under *`[root]`*`/dtd`, where *`[root]`* is the working directory.

This part consists of two sub-parts:

1. Write a *Java* **application** for serialization called `FDSInfoSerializer` (in package `it.polito.dp2.FDS.sol1`), that serializes the data about the DP2-FDS flight management system into a **valid** *XML* file referencing the *DTD* designed in part a). For simplicity, the output *XML* file must reference a local external *DTD* named `flightInfo.dtd`. Assume that this file is always in the same directory as the generated *XML* file. The generated *XML* file must include all the information about the aircrafts, flights, flight instances, and passengers of the DP2-FDS system that can be retrieved by accessing the interfaces defined in package `it.polito.dp2.FDS`. The implementation of the interfaces to be used as data source must be selected via the "abstract factory" pattern: the `FDSInfoSerializer` application must create the data source by instantiating `FlightMonitorFactory` by the static method `newInstance()`. In practice, the application can be developed by modifying the sample *Java* application `it.polito.dp2.FDS.lab1.FDSInfo` (provided in source form in the *.zip* archive), in such a way that it outputs information to a valid *XML* file that references your *DTD* (whereas the `FDSInfo` application outputs information to standard output). Note that the order for data serialization may change in the two applications. `FDSInfo` can be run from the *`[root]`* directory by the provided ant script (which compiles, adjusts classpaths and includes libraries as necessary), by issuing the command

   ```
   $ ant FDSInfo
   ```

   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 variable number of flights, flight instances, and passengers. 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 FDSInfo
   ```

   where X is the seed of the pseudo-random generation engine (an integer number) and Y can be 0 (the default value: the generator generates only flight instances in booking state and no passengers) or 1 (like case 0, but with passengers) or 2 (no restrictions).

   The `FDSInfoSerializer` 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 `[root]/src/it/polito/dp2/FDS/sol1/`.

   The ant script provided with the assignment material can also be used to compile and run the developed application. The command for compilation is

   ```
   $ ant build
   ```

whereas the command for execution is

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

where, of course, `file.xml` is the selected output file name. When running the developed application, the pseudo-random data source is used, and the `seed` and `testcase` parameters can be set, as already shown for the `FDSInfo` 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.
Note: If you use an *XSLT* transformer to write the *XML* file, it normally does not write the *doctype* declaration. In order to have this declaration generated, it is necessary to set the *transformer* property `OutputKeys.DOCTYPE_SYSTEM` to the name of the external file that contains the *DTD*.

2. Using the *DOM* interface, 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 package `it.polito.dp2.FDS`, returning the data loaded from the file. The library must be entirely in package `it.polito.dp2.FDS.sol1` and its sources must be stored in the `[root]/src/it/polito/dp2/FDS/sol1/` directory. The library must include a factory class named `it.polito.dp2.FDS.sol1.FlightMonitorFactory`, which extends the abstract factory `it.polito.dp2.FDS.FlightMonitorFactory` and, through the method `newFlightMonitor()`, creates an instance of your concrete class implementing the `FlightMonitor` interface. The name of the *XML* input file must be obtained by reading the `it.polito.dp2.FDS.sol1.FlightInfo.file` system property. To build the library, use the command:

   ```
   $ ant build
   ```

   If this command fails, check that you have strictly followed all the specifications in this assignment.

The serializer application and the parsing 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 sub-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 `FDSInfoSerializer` application generates well-formed and valid *XML* files (referencing the DTD submitted for part a)).
- the data stored by the `FDSInfoSerializer` application in the output *XML* file are loaded by the classes of the library developed in subpart 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 pseudo-random data generator provided with this assignment. The *.zip* file of this part 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.FDS.lab1.tests`.

Three different test cases will run on the server: test case 0 (which generates all flights in the booking state and no passengers), test case 1 (like test case 0 but with passengers) and test case 2 (no restrictions). Each test case runs once. Passing all of them is mandatory.

In order to locally run one of the three 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, or 2) and `Y` is the seed for the pseudo-random data generator. If not specified, the seed is automatically computed from the current time.

Before trying the automatic tests it is suggested that you test your applications thoroughly by yourself, with the aid of the random data generator (*ant* command with target `FDSInfoSerializer`). 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-final-zip
```

In order to make sure the content of the zip file is as expected by the automatic submission system, do not create the *.zip* file in other ways.