Distributed Programming II

A.Y. 2019/20

Lab3

All the material needed for this lab 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 archives for Lab1 and Lab2, so that the files developed in Lab1 and Lab2 remain under `[root]`.

Purpose

The aim of this Lab is to experiment with XML processing in Java, using the JAXB framework, and with the abstract factory pattern. This lab focuses specifically on the unmarshalling operations and on the implementation of the abstract factory pattern.

Exercise 1

1. Using the *JAXB* framework, write a *Java* **library** that can be used to load and validate an *XML* file written according to the *schema* designed in Lab1 - Exercise 2 (file `[root]/xsd/biblio_e.xsd`). 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.BIB`, returning the data loaded from the file. The library must be entirely in the package `it.polito.dp2.BIB.sol1` and its sources must be stored in the *`[root]`*`/src/it/polito/dp2/BIB/sol1/` directory. The library is one of many possible implementations of the above mentioned interfaces, according to the abstract factory pattern: it must include a factory class named `it.polito.dp2.BIB.sol1.BibReaderFactory`, which extends the abstract factory `it.polito.dp2.BIB.BibReaderFactory` and, through the method `newBibReader()`, creates an instance of your concrete class that implements the `BibReader` interface. The name of the *XML* input file must be obtained by reading the `it.polito.dp2.BIB.sol1.BibInfo.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.

   As with the serializer developed in Lab 2, the library must be portable, even when executed in a distributed environment (there must be no dependency on the local machine, location, and settings).

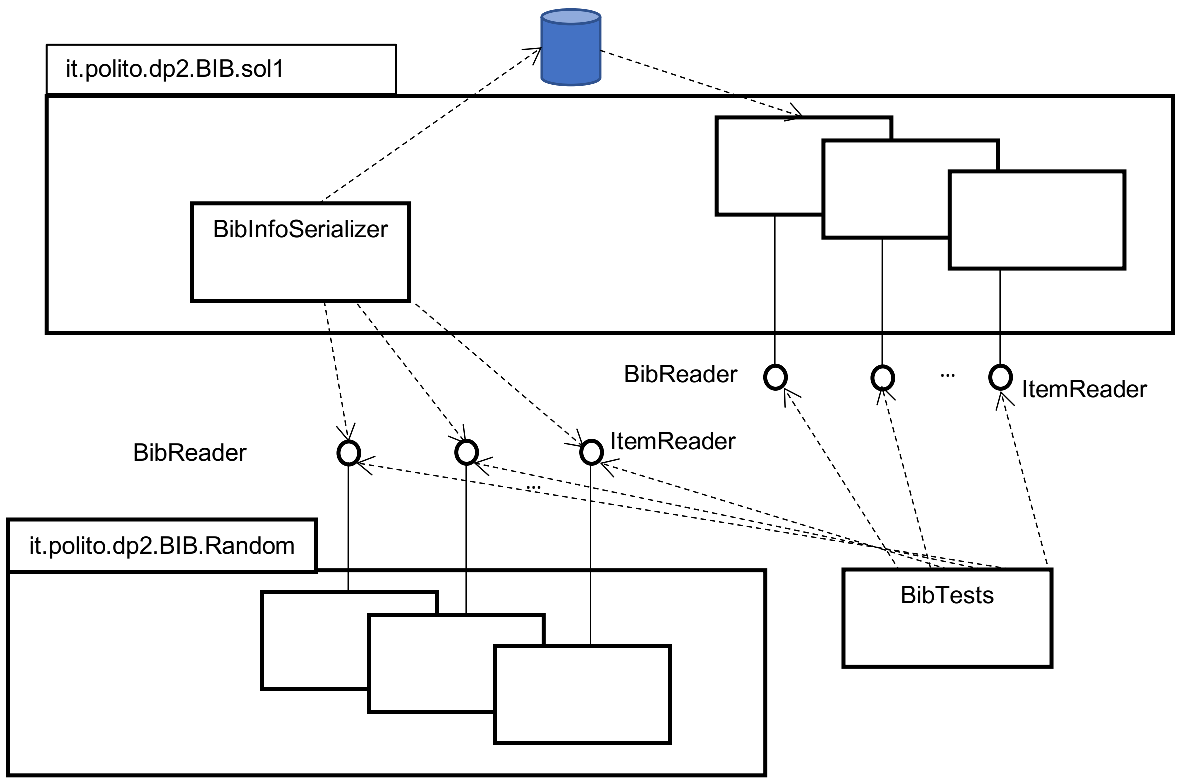
2. Test your application by writing a main that instantiates your library and that calls the interface methods. As input files, you can use the ones generated by your serializer. Try to introduce errors in the input files and check that your library can correctly detect them, by throwing the `BibReaderException`.

3. When you are confident that your library works as expected, you can run the automatic acceptance tester provided with the material of this Lab. The tester runs the solutions of Lab2 and Lab3 together in order to check that

- the `BibInfoSerializer` application generates well-formed and valid *XML* files (against your *schema*).
- the data stored by the `BibInfoSerializer` application in the output *XML* file are loaded by the classes of the library developed in Lab3 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) may be done at evaluation time according to the list of criteria for evaluation published in the course web site (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:



The automatic tester (`BibTests`) uses the data generator provided with Lab2 and it is based on a set of *Junit* tests. The sources of the tests are available in this Lab material, in the package `it.polito.dp2.BIB.ass1.tests`.

In order to be accepted for exemption, your solutions must pass at least the mandatory Junit tests with the following source files of the generator: `biblio.xml` and `biblio2.xml`.

In order to run the tests on your machine, you can issue the following `ant` command:

`$ ant -DsourceFileName=`*fileName* `runFuncTest`

where *fileName* is the name of the file to be used as data source by the data generator. The results of the *Junit* tests can be displayed graphically by double clicking on the `testout.xml` file in Eclipse.

Submission instructions

A single *.zip* file must be submitted, including all the files that have been produced in Labs 1, 2, and 3. 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.