Assignment 5
Web Architectures 2018-2019

Nicolò Pomini

October 18, 2018

1 Introduction

The goal of this assignment is building a java application that leans on remote method invocation (RMI).

In particular, a class called *Document* has a list of strings, and two methods: one for adding a new string to the collection, and the other one to print the whole collection of strings. The role of the server is to *validate* an instance of *Document* adding a timestamp in the end of it, while the client has to create an instance of *Document* and invoke the remote method offered by the server to validate it.

2 Explanation

The problem was solved in several steps.

First of all, the *Document* class was created. As written in the problem statement, it has a collection of strings and two method: one to add a new string and one to print the entire collection.

Secondly, the remote method interface was created. This interface is used both by the server and the client: the former has to *validate* the *Document* object it receives, by adding a timestamp in the end of it; the latter has to request the validation of its *Document* object calling this remote method.

Thirdly, the server was implemented. In addition to what described above, the server has also to start the RMI registry and to bind itself in the latter.

Finally, the client was written. It creates a *Document* object, it writes something on it, and after all invokes the remote method to validate its object.

3 Implementation

The project is implemented in three different packages: one for common objects, one for the server and one for the client. Every package is named like `it.unitn.disi.webarch18.ass5.<something>`.

As regard the *Document* class, it is implemented in the package `common` into the file `Document.java`. The collection of string is implemented using an `ArrayList` of strings; the method `addString` simply adds a string to this collection, and the `toString` method puts the whole collection in a single string object. This ojbect also implements `java.io.Serializable`, in order to be streamed between client and server.

Inside the same package the remote method interface is also placed, in the file called `DocumentValidation.java`. This interface extends the class `java.rmi.Remote`, and for this reason each of its method (only one in this case) may throw a `java.rmi.RemoteException`. The signature of the remote method is the following:

```
public Document validate(Document document)
                    throws RemoteException;
```

which takes a *Document* as parameters and return the same object with the validation string added on it.

The server class is placed into the `server` package, in the file `Server.java`. This class extends the `java.rmi.server.UnicastRemoteObject` class, in order to be a RMI server, and of course it implements the interface described above, to provide the document validation method. In particular, the latter is so structured:

```
 @Override
public Document validate(Document document) throws RemoteException {
    System.out.println("Validating a new document");
    Date now = new Date();
    DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    document.addString("Validated on " + dateFormat.format(now));
    System.out.println("Document validated");
    return document;
}
```

which simply creates the current timestamp and adds it in the end of the document. Furthermore, the server starts the RMI registry, and registers itself into this registry. The code is the following:

```
 try {
    // Create regestry
    LocateRegistry.createRegistry(1099);
    //register the server
    Server server = new Server();
    Naming.bind("rmi:///Server", server);
    System.out.println("The server is ready");
} catch (RemoteException | AlreadyBoundException | MalformedURLException ex) {
    System.err.println("Startup server error:\n" + ex.getMessage());
}
```

The client makes to main actions: firstly, it prepares a *Document* putting into it some strings; secondly, it calles the remote method to validate the object created previously. In particular, once the document is created, a random number (between 2 and 10) of random strings are added to the document, and the latter is printed to the standard output. After that, the remote method is invoked, looking for the server into the RMI registry and calling the `validate` method. Finally, the object is printed again, to show that the validation is added successfully. This portion of code deals with the remote object invokation:

```
 try {
    //validate the object
    DocumentValidation remoteServer = (DocumentValidation) Naming.lookup
                                      ("rmi://localhost/Server");
    document = remoteServer.validate(document);
    // Print the new "viewed" document
    System.out.println("Document after validation");
    System.out.println(document.toString());
} catch (NotBoundException | MalformedURLException | RemoteException ex) {
    System.err.println("Unable to sign the document, the following error happened:\n"
                         + ex.getMessage());
}
```

4 Deployment

To deploy the project, several steps are needed.

First of all, all the source files have to be compiled, with the following commands:

```
cd src
javac it/unitn/disi/webarch18/ass5/common/*.java
javac it/unitn/disi/webarch18/ass5/server/*.java
javac it/unitn/disi/webarch18/ass5/client/*.java
```

After that, the server has to be launched, with the command:

```
java it.unitn.disi.webarch18.ass5.server.Server
```

and once the following message is shown to the standard output

```
The server is ready
```

the server is ready.

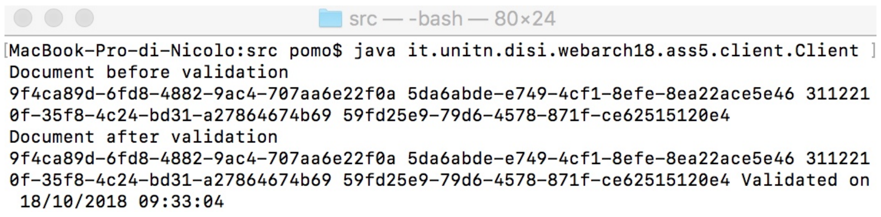
Finally, the client is launched with the following command:

```
java it.unitn.disi.webarch18.ass5.client.Client
```

The execution does not require any user action, because the document generated by the client is filled by some random string created automatically. The output of the client can be seen in Figure 1, where the original and the validated versions of the document can be seen, while the output of the client is shown in Figure 2.

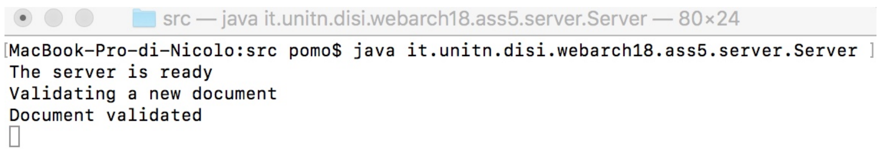
5 Comments and notes

The only problem with this sort of solutions is that they have a high latency: the server needs several seconds to be ready, and also the client takes some seconds (around 4 seconds for every test I tried) to complete its execution.



```
MacBook-Pro-di-Nicolo:src pomo$ java it.unitn.disi.webarch18.ass5.client.Client
Document before validation
9f4ca89d-6fd8-4882-9ac4-707aa6e22f0a 5da6abde-e749-4cf1-8efe-8ea22ace5e46 311221
0f-35f8-4c24-bd31-a27864674b69 59fd25e9-79d6-4578-871f-ce62515120e4
Document after validation
9f4ca89d-6fd8-4882-9ac4-707aa6e22f0a 5da6abde-e749-4cf1-8efe-8ea22ace5e46 311221
0f-35f8-4c24-bd31-a27864674b69 59fd25e9-79d6-4578-871f-ce62515120e4 Validated on
 18/10/2018 09:33:04
```

Figure 1: The output of the client process. In particular, both the original and the validated version of the document are shown: the latter version is equal to the first one, with the timestamp of the validation added in the end of the document.


src — java it.unitn.disi.webarch18.ass5.server.Server — 80×24

```
MacBook-Pro-di-Nicolo:src pomo$ java it.unitn.disi.webarch18.ass5.server.Server
The server is ready
Validating a new document
Document validated
```

Figure 2: The output of the server process.