EE6052 Web-based Application Design
CE4208/EE4023/ED5022 Distributed Systems

Exercise Sheet 6

Always log on as user "puser" with password "PUSER".

Objectives:
Learn how to:

- develop and run an RMI server application
- develop and run an RMI client application
- develop and run distributed applications based on Java Remote Method Invocation (RMI).

Preparation:
Modify the `java.policy` file. Make sure to edit the correct policy file: In the menu "Tools→Java Platforms" you can find the path to the used Java implementation. From there, go to the directory "`jre\lib\security`" in which you should find the correct `java.policy` file. Simplest (and most insecure solution) is to add the following at the top of your file

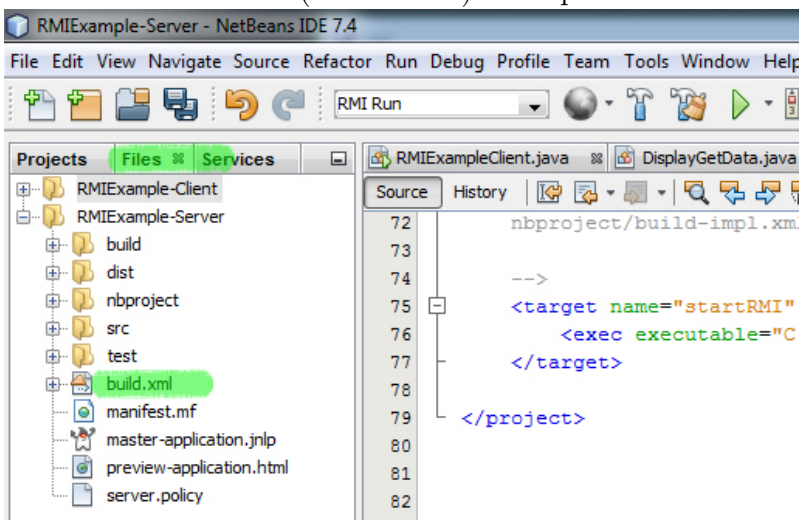
```
grant {
   permission java.security.AllPermission;
};
```

Please remove this addition from the policy file when you have finished this lab!!!

After the Lab
Remove the added permissions from the policy file.
Please also remove your project folder from the PC - otherwise, students in subsequent labs will find your solution ...

Ex6.1: *Your first RMI Application*
The RMI Server:

- Start NetBeans and create a new project called "MyServer" (choose type Java→Java Application).
- Switch to file view (tab "Files") and open the file "build.xml":



  At the end of that file (before the end tag `</project>`) enter the following:

```
<target name="startRMI" depends="init">
   <exec executable="${java.home}/bin/rmiregistry"
      dir="${build.classes.dir}"></exec>
</target>
```

  This will add a target "startRMI" that can be used to start the Java RMI registry with proper class path.

- Switch back to projet view and Create a java package in your project and add an interface (don't forget to derive your interface from the interface `java.rmi.Remote` and add javadoc comments to it) with the following two methods:

```
public void printMessageOnServer(String message);
public String getDataFromServer(String name);
```

- Add a class that implements your interface. The method `printMessageOnServer(...)` outputs the passed message to the console. The method `getDataFromServer(...)` converts the passed name into all upper case characters and returns the resulting string.
- Provide your class with a `main()` method that performs:

  (a) Ensures a Security Manager is running.

(b) Creates an object of your class and converts it into a remote stub.
(c) Registers the created stub with the local registry.
(d) I also recommend to print a message to the console, to confirm that the server is running properly.
(e) Don't forget to catch the RemoteException

The RMI Client:

- Add another project called "MyClient" to NetBeans.
- Add the .jar file from your server as a library to your client project:
  (a) Right click on the project and select "Properties".
  (b) Select "Libraries" on the left hand side
  (c) Click on "Add JAR/Folder" and locate the .jar file from your server project. The jar file can be found in the "dist" folder of your server project. If you plan to run your project later on a different computer, I recommend to select "Reference as Relative Path".
- Create a package and add a class to it. Add a `main()` method that
  (a) Ensures a Security Manager is running.
  (b) Retrieves a remote object reference to your server.
  (c) Prompts the user (I recommend using the class JOptionPane for input/output) for a message.
  (d) Invokes the method `printMessageOnServer(...)` on the server (and passes the message from the user to it).
  (e) Prompts the user for his/her name.
  (f) Invokes the method `getDataFromServer(...)` on the server and displays the returned string.

Running your application:
To run you application, you need to do the following (in the given order):

(a) Switch to class view. In the server projct, right-click on `build.xml` and select "Run Target→Other Targets→startRMI".
(b) Run the server - wait until your confirmation message appears.
(c) Run the client.
(d) Please advised that you need to terminate the rmiregistry and the server manually - otherwise, these will continue to run!!! To terminate these, go the corresponding output tab and click on the red square to the right of it.

Troubleshooting:

Program terminates with an AccessControlException: access denied Make sure your edited the (correct) java policy file as per instructions in the preparation step.

ConnectException: Connection refused Make sure to run the rmiregistry prior to executing the server.

ServerException: RemoteException occurred in the serve thread Make sure to run the correct rmiregistry with proper class path set. Easiest way is to follow the instruction above via the build.xml file.

Class not found exception when runing the client Make sure to:

- have the jar file from the server added as a library
- have imported the interface or package from the server project
- not have an interface in the client project with the same name as the server interface

Ex6.2: *A slightly more useful RMI Application*
You are to write an IT Acronym Server. Clients connect to this server and send an acronym such as ("RMI" or "CAS") and the server responds with the full form of the acronym. For example if the clients sends "CAS" as request the server will respond with "CAS : compare and swap", or if the clients sends "RMI" the server will respond with "RMI: remote method invocation". Use at least the following acronyms:

Acronym	**Full Form**
B2B	Business-to-Business
EJB	Enterprise Java Bean
JDK	Java Development Kit
NFS	Network File System
RMI	Remote Method Invocation

Ex6.3: *A RMI White-Board Application*
A white-board is an application that allows multiple clients to work on the same shared resource. Clients can join on the white-board. After joining they can post a string to the white-board which will be repeated to all connected clients. Design and implement a simple white-board application based on Java RMI - clients need to be able to join/leave white-boards and be able to post messages to white-boards.

Hint: Both client and server need to have remote interfaces. The server's need is obvious. However, as the server needs to distribute any posted string to all clients, clients also need a remote interface that can be invoked by the server.