Service-Centric Cloud Technologies - Practical Study #6
Distributed Computing with RMI

In this RMI-based application, you will create an Account object that represents a bank account and then use RMI to export it as a remote object, which is accessed by remote clients (e.g. ATMs, on-line banking, etc.) to carryout transactions.

Unpack the folder containing the exercise files "**rmiBank.zip**". Create a new project "**rmiBank**" *with Existing Sources*. The project contains the following source files:

i. ***Account.java*** is the interface for this application, declaring the methods implemented and published by the server. The Account interface includes methods to get the account name and balance and to make deposits and withdrawals.

ii. ***AccountImpl.java*** is the implementation of the interface. It contains the business-logic implementation file for this application. It implements all the methods declared in the *Account* interface and has a constructor that takes the name of the account holder as a parameter.

iii. ***AccountServer.java*** is the implementation of a backbone Server, which creates an instance of the remote service. The Server program makes the remote object available to all the clients by registering the object with the RMI registry. It creates an *AccountImpl* object and then binds it to a name in the local registry using the *java.rmi.Naming* interface.

iv. ***AccountClient.java*** is the client program that uses the remote service. It looks up the remote Account object using the java.rmi.Naming interface and then calls the deposit method on the Account object:

```
AccountImpl acct = new AccountImpl("Mr_Blobby");
Naming.rebind("BlobbyAccount", acct);
```

1. Compile the project files

Use the NetBeans IDE as usual. This will create .class files for the respective java files. By default, the .class files (AccountServer.class, AccountClient.class, etc.) will be located in: **"…[your path]…\rmiBank\build\classes"**.

2. Starting the rmiregistry.

Before starting the registry, the java CLASSPATH has to include your project's classes folder. You can set it by executing the following instruction from the command line:

```
H:\...> set CLASSPATH=/…[your path]… rmiBank\build\classes/
```

Start the **RMI Registry** to enable the registration of remote objects. You need to set the path to the bin folder holding the **"rmiregistry.exe"** file. For example, the Java path can be set as follows (the path might be different depending on the installation folder):

```
H:\...> set path="C:\Program Files\Java\jdk1.7.0_07\bin";%path%
```

Now, from the same command line above start the registry:

```
H:\...> start rmiregistry
```

4. Run the RMI server.

Run the AccountServer file in the NetBeans IDE

The server will start with the message:

Registered account as BlobbyAccount

5. Run the Client.

Run the AccountServer file in the NetBeans ID*E*

Following is the trace of the program when the client program is executed twice.

run-single:

Deposited 12,000 into account owned by Mr_Blobby

Balance now totals: 24000.0

BUILD SUCCESSFUL (total time: 0 seconds)

6. Modify the Application as follows:

1. Add a method to the *Account.java* interface that calculates the interest on the current balance at a value of 5%;
2. Make the necessary changes to the client and server sides to implement and demonstrate the above modifications.

7. For the ambitious:

How would you change the design of the enterprise application to support multiple bank clients? Remember that the starting point should always be the remote interface, i.e. "Account.java".

- Read the Advanced RMI notes in the practical section on NOW
- Discuss with your colleagues and tutor.