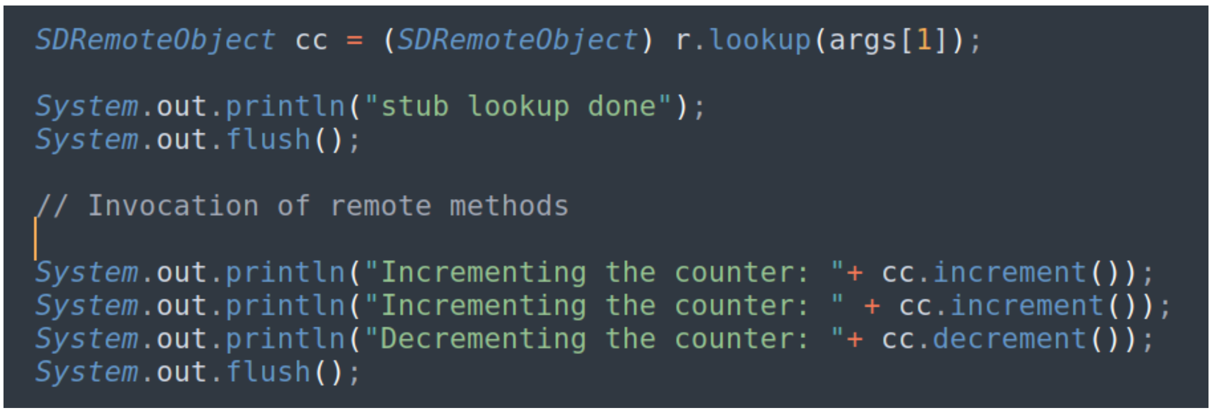
3. RMI C/S

Alberto Miguel Diez

a. Write an RMI client program that allows us to test the server. Deploy the client and the server in your own Linux infrastructure.

To carry out this exercise I have based on the code provided in http://paloalto.unileon.es/ds/lab/rmipract.zip

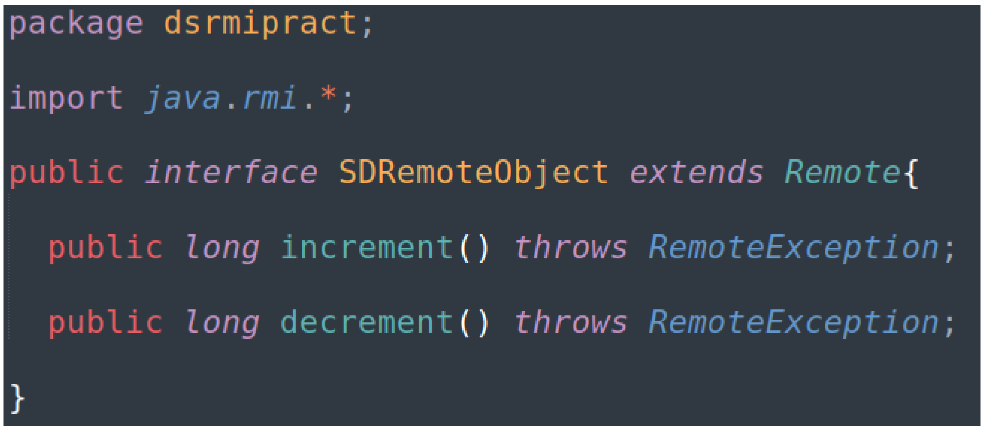
I change the Client code in such a way that the remote object invokes the increment() and decrement() methods. To test the operation I have performed the function increase and decrease several times.



```java
SDRemoteObject cc = (SDRemoteObject) r.lookup(args[1]);

System.out.println("stub lookup done");
System.out.flush();

// Invocation of remote methods

System.out.println("Incrementing the counter: "+ cc.increment());
System.out.println("Incrementing the counter: " + cc.increment());
System.out.println("Decrementing the counter: "+ cc.decrement());
System.out.flush();
```

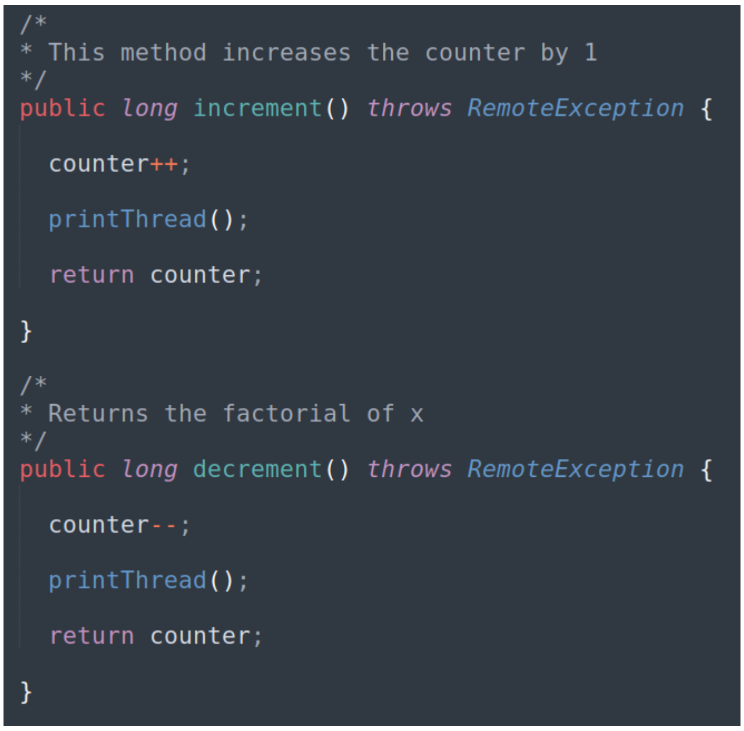
Server:

I have created an interface with two methods increment () and decrement () that return a long number corresponding with the counter variable.



```java
package dsrmipract;

import java.rmi.*;

public interface SDRemoteObject extends Remote{

  public long increment() throws RemoteException;

  public long decrement() throws RemoteException;

}
```

The SDRemoteObjectImpl class implements this interface, below I show a screenshot that corresponds to the implementation of the interface methods. Keep in mind that the counter variable is a private attribute of the class.



```
/*
* This method increases the counter by 1
*/
public long increment() throws RemoteException {

  counter++;

  printThread();

  return counter;

}

/*
* Returns the factorial of x
*/
public long decrement() throws RemoteException {

  counter--;

  printThread();

  return counter;

}
```

Next, I will deploy the server and the client on my computer. For this I will use a build.xml and the ant tool. Also, I had to install apache2 on my computer and start it using the following command:

sudo service apache2 start

Then, I compile the client and the server code with the following commands:

sudo ant compile_client

sudo ant compile_server

The last command copies the interface into the web server. Then, I runned the rmiregistry:

sudo ant rmiregistry

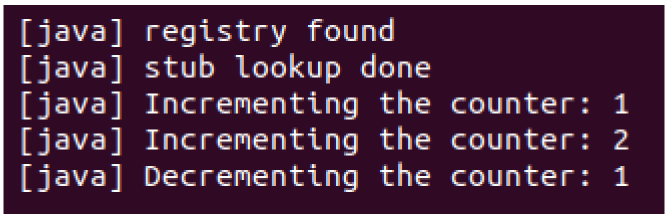
Finally the server can be run (for this part you have to give the name to the object and the IP addresses, everything is configured in the xml).

sudo ant run_server

To run the clien: *sudo ant run_client*

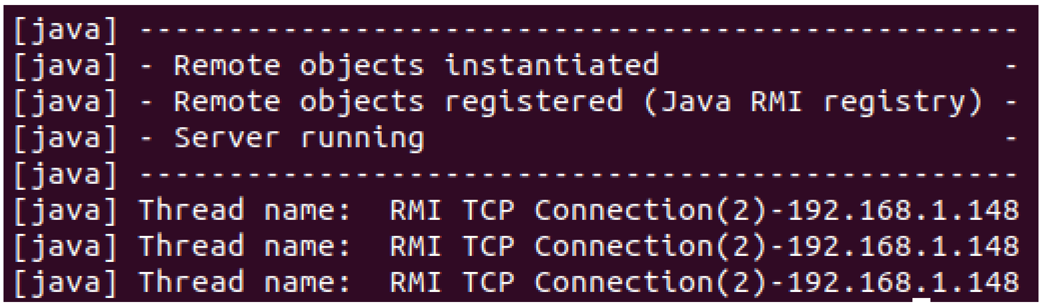
b. Provide an extensive discussion of your software design and the tests that demonstrate that it functions correctly

To demonstrate the correct operation of the program I carried out an execution. In the following capture of the client's output, it can be seen that the program works correctly. The counter is initialized at 0, therefore the first increase sets it to a value of 1, the second to 2 and finally when decreasing the value it returns to 1.



```
[java] registry found
[java] stub lookup done
[java] Incrementing the counter: 1
[java] Incrementing the counter: 2
[java] Decrementing the counter: 1
```

The next capture is the server output.



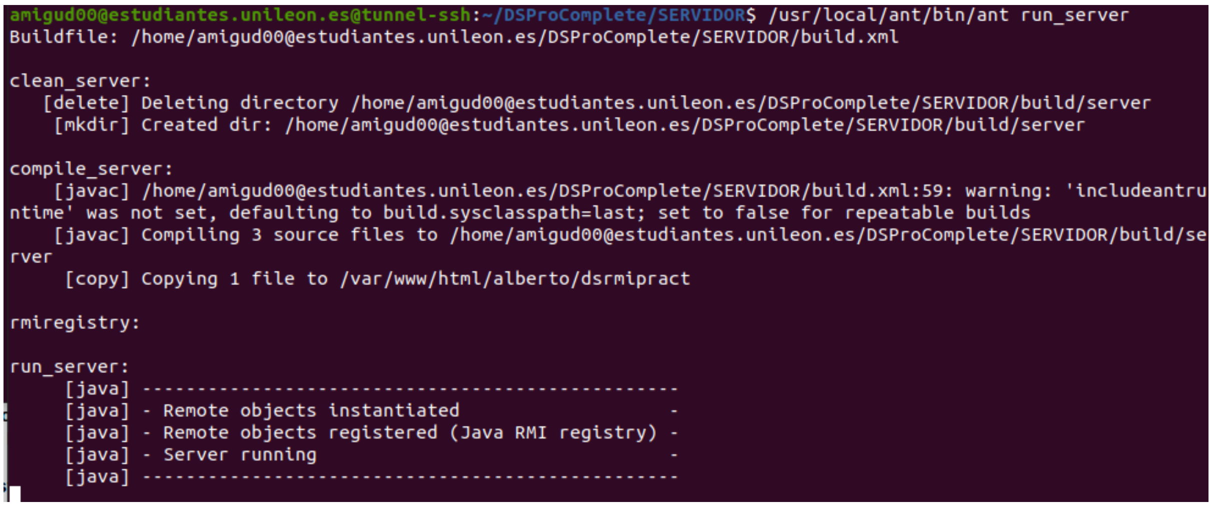
```
[java] ---------------------------------------------
[java] - Remote objects instantiated               -
[java] - Remote objects registered (Java RMI registry) -
[java] - Server running                            -
[java] ---------------------------------------------
[java] Thread name:  RMI TCP Connection(2)-192.168.1.148
[java] Thread name:  RMI TCP Connection(2)-192.168.1.148
[java] Thread name:  RMI TCP Connection(2)-192.168.1.148
```

c. Highlight the core difficulties involved in this tiny distributed project .
The biggest difficulty was configuring the build.xml for the execution of the program.

d. When you finish this RMI C/S small project, extend it by making your server available in paloalto.unileon.es and by checking it via Internet:

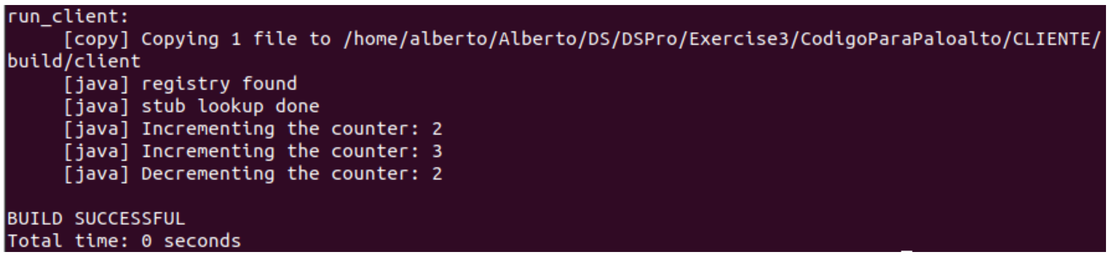
> **a. I will open a specific TCP port at paloalto.unileon.es for your own RMI server; send me a request to chema.foces@unileon.es and I'll provide you a port number for your personal use. Remote access to paloalto.unileon.es with ssh is at port 50500. The port I'll assign to you is for your Java RMI server only.**
> **b. paloalto.unileon.es's public rmiregistry listens at port 1099, however, Internet access to it is transparently provided to your C/S system via NAPT at the lab's router. The external port for accessing rmiregistry is port number 60001. Now, compose a more detailed explanation about how this instance of rmiregistry is accessed from anywhere in Internet.**

Next, I will demonstrate the execution of the program:
In the paloalto server it has to run the server with the /usr/local/ant/bin/ant run_server command.



```
amigud00@estudiantes.unileon.es@tunnel-ssh:~/DSProComplete/SERVIDOR$ /usr/local/ant/bin/ant run_server
Buildfile: /home/amigud00@estudiantes.unileon.es/DSProComplete/SERVIDOR/build.xml

clean_server:
   [delete] Deleting directory /home/amigud00@estudiantes.unileon.es/DSProComplete/SERVIDOR/build/server
    [mkdir] Created dir: /home/amigud00@estudiantes.unileon.es/DSProComplete/SERVIDOR/build/server

compile_server:
    [javac] /home/amigud00@estudiantes.unileon.es/DSProComplete/SERVIDOR/build.xml:59: warning: 'includeantru
ntime' was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds
    [javac] Compiling 3 source files to /home/amigud00@estudiantes.unileon.es/DSProComplete/SERVIDOR/build/se
rver
     [copy] Copying 1 file to /var/www/html/alberto/dsrmipract

rmiregistry:

run_server:
     [java] ---------------------------------------------
     [java] - Remote objects instantiated               -
     [java] - Remote objects registered (Java RMI registry) -
     [java] - Server running                            -
     [java] ---------------------------------------------
```

Next, the client that has a different build will be run. To run the client, we're going to use the ant run_client command.



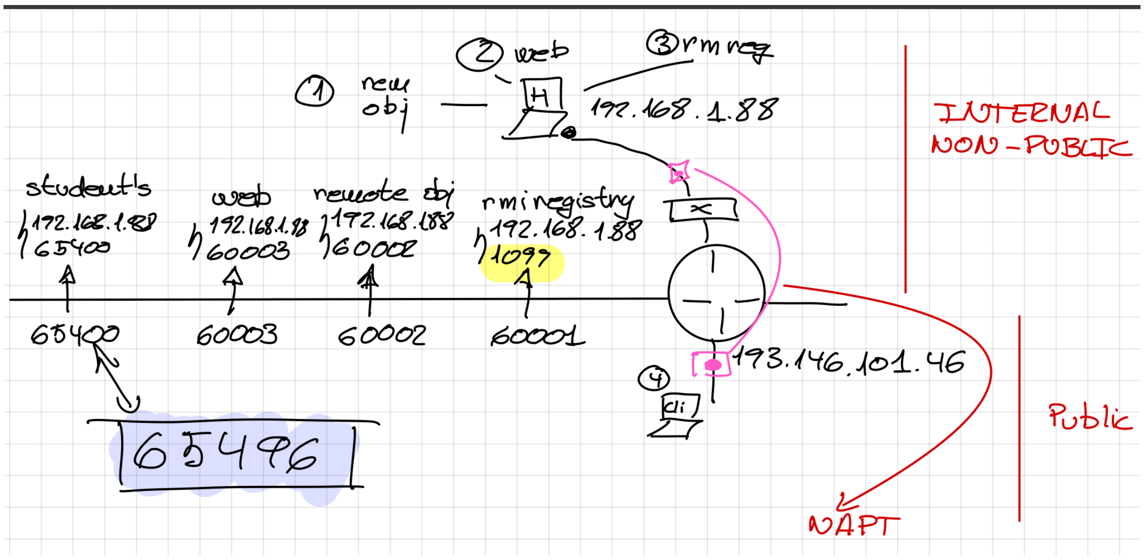
```
run_client:
     [copy] Copying 1 file to /home/alberto/Alberto/DS/DSPro/Exercise3/CodigoParaPaloalto/CLIENTE/
build/client
     [java] registry found
     [java] stub lookup done
     [java] Incrementing the counter: 2
     [java] Incrementing the counter: 3
     [java] Decrementing the counter: 2

BUILD SUCCESSFUL
Total time: 0 seconds
```

It can be verified in the previous capture that the client has correctly found the registry and the stub of the object, and therefore has been able to perform the methods associated with that object.

One of the difficulties of this exercise is dealing with the NAPT protocol implemented by the paloalto router. This makes the ports accessed by the client (public ports) within the internal network have a different value, as well as the IP addresses. The client will look for the rmiregistry at IP 193.146.101.46 (which corresponds to the IP of paloalto.unileon.es) and the computer where the server is located is 192.168.1.88. The rmiregistry will be running on the public port 193.146.101.46:60001 that corresponds to the private network with the computer 192.168.1.88 and port 1099. On the other hand, the remote object will be in a personal port of mine, which in my case is the port 65496. Finally, the web where the remote object class resides is on port 60003, this is also a challenge since the well-known port for web servers is 80. In my case the remote object class is in the URL paloalto.unileon.es:60003/alberto/dsrmipract.

Another problem that I have found has been to configure the build.xml for the client and the server where I had to adjust the ports that I mentioned above. Modifying the code to fit the paloalto service has been easier.

Below I show a small diagram of the structure of the network that has influenced the development of this exercise.



① new obj
② web
③ rmreg
H
192.168.1.88
INTERNAL NON-PUBLIC
student's 192.168.1.88 65400
web 192.168.1.88 60003
remote obj 192.168.1.88 60002
rmiregistry 192.168.1.88 1099
65400
60003
60002
60001
65496
193.146.101.46
④ cli
Public
NAPT