

Assignment #2: Simple Client/Server Programming with CORBA and RMI

DUE: 10:30 am, Wednesday, April 21.

This assignment consists of 2 main parts: (i) writing simple client-server programs using CORBA (and Java), and (ii) writing simple client-server programs using Java RMI. For each of these parts, you will need to complete roughly the same three sub-tasks: (a) implement and run a simple “Hello World” program, (b) implement a Java program that acts as a client and contacts a grading server that we have written and deployed, and (c) implement and deploy a Java program that acts as a server and accepts invocations from a grading client that we have written.

You will not have access to the implementations of our grading client or grading servers, but you will (of course) be given the required interfaces.

There are many system, environment, and language issues you have to get just right. So leave yourself enough time to do this assignment! Of course, I would recommend starting right away...

This assignment is relatively simple, but there are many parts. Read the following sections carefully to be sure you submit all the required information.

Part I: CORBA Client-Server

Task A: “Hello World”

Good Citizenship.

In the course of completing this assignment, you will start many java processes. If you start these processes in the background (eg the directions at Sun’s site have you use `&`), make sure you clean up afterwards by killing these background processes (use `ps` to see if there are any and use `kill` to remove them).

“Hello World”: Naming Service

Follow the instructions given at Sun’s web site for implementing the “hello world” program in CORBA:

<http://java.sun.com/j2se/1.4.2/docs/guide/idl/GShome.html>

Implement two versions of this program: one that uses the Inheritance Model and the other that uses the Delegation (Tie) Model. Both implementations should use the Naming Service for obtaining references to remote objects.

What to turn in

For Task A, turn in the following:

1. A UML diagram of your “hello world” application, built using the inheritance model. Your diagram should include the server, servant, client, and classes generated by idlj. Clearly distinguish between generated classes and those that you wrote by hand.
2. A UML diagram as above, but describing the “hello world” application built using the delegation (tie) model.

Task B: Building and Running a Client

Implement a client program that invokes a method on a CORBA object satisfying the following IDL interface:

```
module GraderApp {
    interface Grader {
        boolean ping (in string id);
    };
};
```

The ping method will always return the value “true”. Your client should call the “ping()” method *with a string argument that is you and your partner’s complete names*. For example:

```
result = target.ping("Paul Sivilotti and Greg Buehrer");
```

This allows the Grader implementation to record you as having completed this part of the assignment! If you do not use your names as the argument, you will not get credit for this part of the assignment.

The Grader implementation object will be running for several days before the due date (and time) of the assignment. When you call “ping”, it will record the date and time at which this call was made. If you get a result returned from ping (a boolean), your client was successful. There is no need to bother the grader with email.

Binding to the Grader Implementation Object. The Grader implementation object is named “GraderServant794R”. The port and location of the naming service will be posted to the course newsgroup when the server is available to accept requests.

Task C: Building and Running a Server

Implement and deploy a CORBA server that implements the following IDL interface:

```
module StudentApp {
```

```
interface Student {
    string getGrade ();
};
};
```

Your implementation of the `getGrade` method need not do anything particular (except terminate without an exception). The string that it returns should be the grade that you expect for this part of the assignment (something like “A” is appropriate).

You should register your implementation with the Naming Service under the name `Student_xxxxx`, with the x’s replaced by your login id in lower case. For example, `Student_paolo` or `Student_buehrer`. *Use the same naming service as was used in the previous part.* Again, the hostname and port number will be posted to the newsgroup when it is available.

Completing submission of this part. We will test your completion of this part of the assignment by binding to your server and invoking the `getGrade()` method. In order for us to know that your server is ready to be tested, you *must send email to the TA indicating the name of your service and that it is ready to be graded.*

There are 3 windows of opportunity for you to be graded: 8:30 - 9:30 am the day the lab is due, 8:30 - 9:30 am the next day (with a 25% penalty), and 8:30 - 9:30 am the next day (with a 50% penalty). You *MUST* send your email before 8:30 am the morning you wish to be graded. Have your object up and running during the grading window (8:30 - 9:30). Leave it up until you hear back from the TA.

At the TA’s discretion, there may be an early window available as well. Watch the newsgroup for details.

Part II: Java RMI Client-Server

Task A: Getting Your Feet Wet: “Hello World”

Follow the step-by-step tutorial instructions given at Sun’s web site for implementing the “hello world” program in Java RMI:

<http://java.sun.com/j2se/1.4.2/docs/guide/rmi/getstart.doc.html>

(See the class web page for a link to this part of the Sun Java documentation.)

What to turn in

Part (i) is not handed in or graded. However, you are *strongly* urged to complete this part as it will be tremendous help in understanding how to write, compile, and execute Java RMI programs. The step-by-step instructions for completing part (i) should make it relatively easy for you to do so. Extending your solution to complete parts (ii) and (iii) should then not be difficult either.

Task B: Building and Running a Client

Implement a Java RMI client object that invokes a method on an object satisfying the following interface (which will also be posted to the class newsgroup).

```
package listen;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Grader extends Remote {
    boolean ping (String id) throws RemoteException;
}
```

The ping method will always return the value “true”. Your client should call the “ping()” method *with a string argument that is you and your partner’s complete names*.

```
result = target.ping("Paul Sivilotti and Greg Buehrer");
```

This allows the Grader implementation to record you as having completed this part of the assignment! If you do not use your name as the argument, you will not get credit for this part of the assignment.

The Grader implementation object will be running for several days before the due date (and time) of the assignment. When you call “ping”, it will record the date and time at which this call was made. If you get a result returned from ping (a boolean), your client was successful. There is no need to bother the grader with email.

Binding to the Grader Implementation Object. The Grader implementation object is named “GraderServant894x”. The port and location of the naming service will be posted to the course newsgroup when the server is available to accept requests. (Note, your client can just take the hostname and port number as command-line arguments, much like the sample Hello World client.)

Task C: Building and Running a Server

Implement and deploy a Java RMI server object that implements the following interface (which will also be posted to the class newsgroup):

```
package tester;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface StudentServer extends Remote {
    String getGrade () throws RemoteException;
}
```

Your implementation of the `getGrade` method need not do anything particular (except terminate without an exception). The string that it returns should be the grade that you expect for this part of the assignment (something like “A” is appropriate).

You should register your implementation with the Naming Service under the name `Student_XXXXX`, with the x’s replaced by your login id in lower case. For example, `Student_paolo` or `Student_desais`. You should use an `rmiregistry` that you yourself start (not the one used in the previous part). The hostname and port number of your `rmiregistry` must be sent to the grader when requesting that your server be tested.

Completing submission of this part. We will test your completion of this part of the assignment by binding to your server and invoking the `getGrade()` method. In order for us to know that your server is ready to be tested, you *must send email to the TA indicating the name of your service, that it is ready to be graded, and the hostname and port number of the `rmiregistry` where it is bound.*

There are 3 windows of opportunity for you to be graded: 8:30 - 9:30 am the day the lab is due, 8:30 - 9:30 am the next day (with a 25% penalty), and 8:30 - 9:30 am the next day (with a 50% penalty). You *MUST* send your email before 8:30 am the morning you wish to be graded. Have your object up and running during the grading window (8:30 - 9:30). Leave it up until you hear back from the TA.

At the TA’s discretion, there may be an early window available as well. Watch the newsgroup for details.