TP2: RMI Java ii.3502

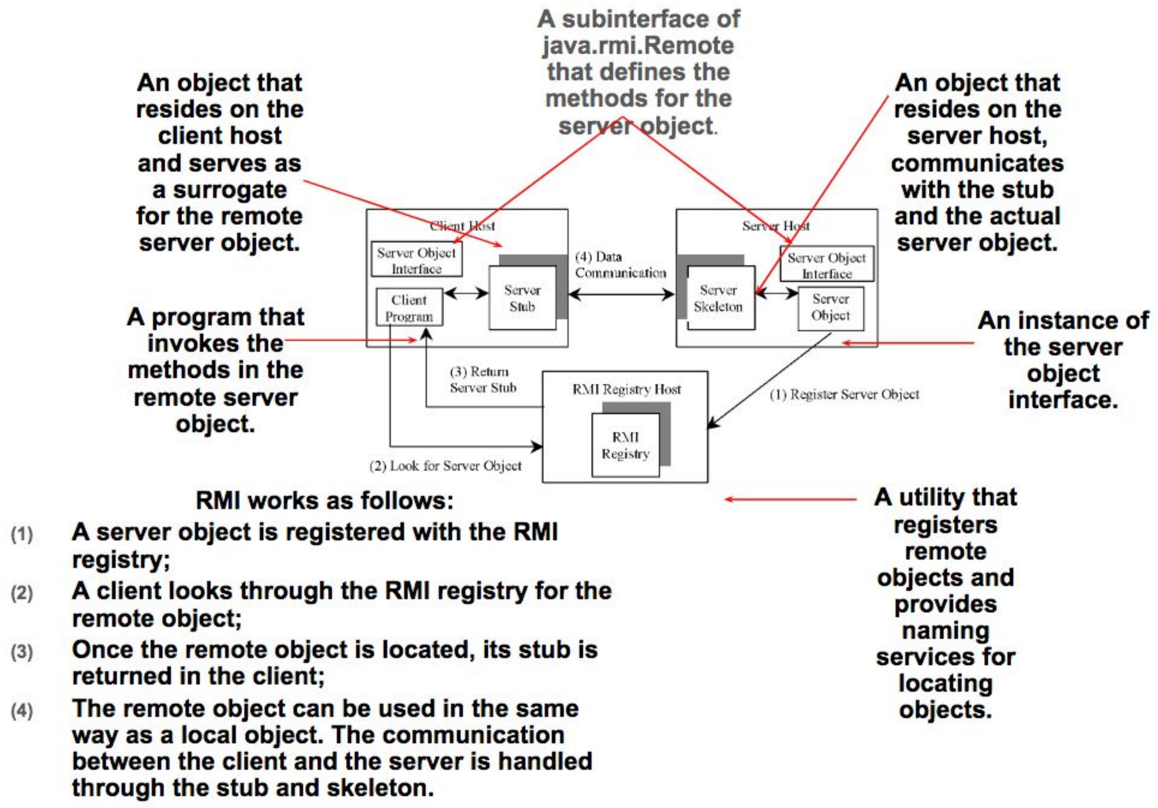
Objective :

The aim of this TP is to learn the process of developing RMI applications ,Develop three-tier applications using RMI and know the differences between RMI and socket-level programming

Part 0 : reminder

RMI is the Java Distributed Object Model for facilitating communication between objects. RMI is a higher level API built on top of the sockets. Socket-level programming allows you to pass data through sockets between computers. RMI not only allows you to spend data among objects on different systems, but also to call methods in a remote object.

An RMI component can act as both a client and a server, depending on the scenario in question.
An RMI system can pass functionality from a server to a client and vice versa. A client / server system typically only transmits data back and forth between the server and the client.


A subinterface of java.rmi.Remote that defines the methods for the server object.
An object that resides on the client host and serves as a surrogate for the remote server object.
An object that resides on the server host, communicates with the stub and the actual server object.
A program that invokes the methods in the remote server object.
An instance of the server object interface.
A utility that registers remote objects and provides naming services for locating objects.
Client Host
Server Object Interface
Client Program
Server Stub
(4) Data Communication
Server Host
Server Skeleton
Server Object Interface
Server Object
(3) Return Server Stub
RMI Registry Host
RMI Registry
(1) Register Server Object
(2) Look for Server Object

RMI works as follows:

(1) A server object is registered with the RMI registry;

(2) A client looks through the RMI registry for the remote object;

(3) Once the remote object is located, its stub is returned in the client;

(4) The remote object can be used in the same way as a local object. The communication between the client and the server is handled through the stub and skeleton.

isep TP2: RMI Java ii.3502

Part 1: Hello World RMI vs hello world Socket

• For RMI, you need to be running *three* processes

- The Client
- The Server
- The Object Registry, rmiregistry, which is like a DNS service for objects

•You also need TCP/IP active

•The class that defines the server object should extend UnicastRemoteObject

- This makes a connection with exactly one other computer
- If you must extend some other class, you can use exportObject() instead
- Sun does *not* provide a MulticastRemoteObject class

•The server class needs to register its server object:

- String url = "rmi://" + ***host*** + ":" + ***port*** + "/" + ***objectName***;

•The default port is 1099

- Naming.rebind(url, ***object***);

•Every remotely available method must throw a RemoteException (because connections can fail)

•Every remotely available method should be synchronized

Hello World server

Interface:

```
import java.rmi.*;

public interface HelloInterface extends Remote {
   public String say() throws RemoteException;
}
```

Class:

```
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements
```

```
HelloInterface {
   private String message; // Strings are serializable

   public Hello (String msg) throws RemoteException {
     message = msg;
   }

   public String say() throws RemoteException {
     return message;
   }
}
```

Registering the Hello World server:

```
class HelloServer {
   public static void main (String[] argv) {
     try {
     Naming.rebind("rmi://localhost/HelloServer",
                                  new Hello("Hello, world!"));
     System.out.println("Hello Server is ready.");
     }
     catch (Exception e) {
     System.out.println("Hello Server failed: " + e);
     }
   }
}
```

The Hello World client program:

```
class HelloClient {
   public static void main (String[] args) {
     HelloInterface hello;
     String name = "rmi://localhost/HelloServer";
```

```
        try {
                hello = (HelloInterface)Naming.lookup(name);
                System.out.println(hello.say());
        }
        catch (Exception e) {
                System.out.println("HelloClient exception: " + e);
        }
        }
}
```

Download the File RMI.zip.
Open your IDE, create two projects (RMIServerSide and RMIClientSide) and copy the java files.

In different terminals try to:

- Run the server program:
  - java HelloServeur
- Run the client program:
  - java HelloClient

If all goes well, you should get the “Hello, World!” message

Part 2: Student Score RMI

We want to create an application that allows us to calculate the scores of the students using RMI. This application has two interfaces: StudentInterface.java and PromotionInterface.java
The interface StudentInterface.java allows the client to access to the student information: name, age,Id, scores….Each student passes many exams and each exam is associated with a score and a coefficient (a double between 0 and 1). This interface proposes three methods:

1. add_exam: adds an exam to a student. An exam has a name, a score and a coefficient.
2. print_exams: sends a string (list of exams) to the client.
3. calculate_ average: calculates the average score.

The interface PromotionInterface.java allows to:

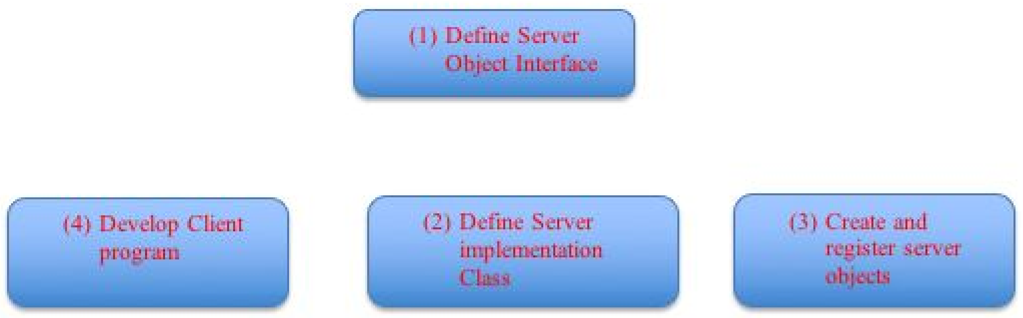
- Create a new student (method add_student)
- Find a student (method get_student). As this method return a reference to an object, the client can ask to calculate the average score of the student.

- Calculate the average score of all the promotion (method promotion_score)

Design this application using RMI, identify all the needed java classes and the data model(without using a database). You should use both RemoteObject and serializable objects.

Hint :

You have to create this program as following:


(1) Define Server Object Interface
(4) Develop Client program
(2) Define Server implementation Class
(3) Create and register server objects

1. Define a server object interface that serves as the contract between the server and its clients, as shown in the following outline:

```
Public interface ServerInterface extends Remote { public void
service1(...) throws RemoteException; // Other methods }
```

A server object interface must extend the java.rmi.Remote interface.

2. Define a class that implements the server object interface, as shown in the following outline:

```
public class ServerInterfaceImpl extends UnicastRemoteObject implements
ServerInterface { public void service1(...) throws RemoteException { //
Implement it } // Implement other methods }
```

The server implementation class must extend the java.rmi.server.UnicastRemoteObject class. The UnicastRemoteObject class provides support for point-to-point active object references using TCP streams

3. Create a server object from the server implementation class and register it with an RMI

```
ServerInterface server = new ServerInterfaceImpl(...);
Registry registry = LocateRegistry.getRegistry();
registry.rebind("RemoteObjectName", obj);
```

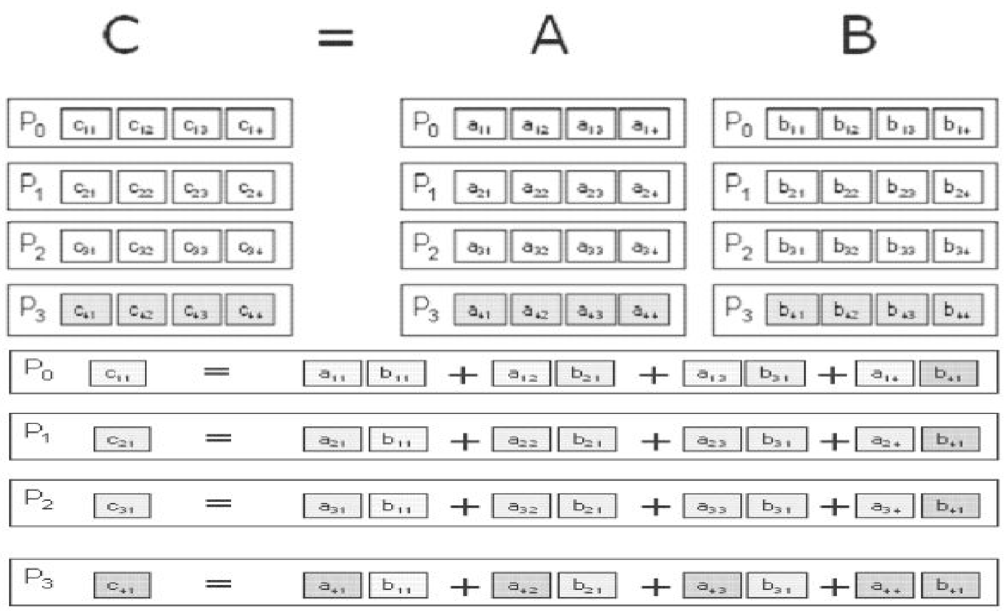
4. Develop a client that locates a remote object and invokes its methods, as shown in the following outline:

```
Registry registry = LocateRegistry.getRegistry(host); ServerInterface
server = (ServerInterfaceImpl) registry.lookup("RemoteObjectName");
server.service1(...);
```

Part 3: Matrix multiplication

Matrix multiplication MM is an important operation of linear algebra. It is widely used in many systems and sometimes it is expensive for some applications when size is large (eg image processing for self-driving cars).

In this part you need to implement a distributed Application using JAVA RMI to reduce the cost of the calculation of this operation.



TP2: RMI Java ii.3502

As shown in the hint of the part 2 you are to implement the following step using RMI:

Step 1: Interface (Matrix definition and operation)
Step 2: Implementation MatrixMultiplication.java
Step 3: Server MatrixServer.java
Step 4: Client ClientMultiplication.java

Hint:

Use this link as helper for this part : https://www.cs.uic.edu/~troy/fall06/cs441/rmi/index.html
It's user a simple two number calculator using JAVA RMI

Download the file “RMI calculator.zip” from moodle to get inspired for this part