Università degli Studi di Bologna
Facoltà di Ingegneria

Principles, Models, and Applications for Distributed Systems M

Lab 6 (worked out)

Java RMI

Luca Foschini

Specification: Client

Design and develop a Client/Server application to manage **conference registrations**. The conference organizing committee provides session speakers with an interface to subscribe to a session and to obtain updated overviews of sessions and presentations over the conference.

The **Client** can invoke functions to:

- **register a new speaker** to a session;
- **get the conference program**.

The Client is realized as a **cyclic process** that interacts with user until she inputs the end of file (i.e. EOF).

Specification: Server

The **Server** stores locally a program for each of the 3 conference days. Each daily program is saved in a matrix o Strings: each row represents a session (at most 12) and each session contains up to 5 presentations; each element contains the speaker name.

Session	Presentation 1	Presentation 2	…	…	Presentation 5
S1	Speaker1 Name	Speaker2 Name			
S2					
…					
S12					

Specification: details

Client sends requests to **Server**; error conditions (for example, when the Client requests to register to a day/session that does not exist or is already full) have to be managed at the most suitable side.

Some examples of registration request:

- `Day? 25`
- `Day not valid`
- `Day? 2`
- `Session? S46`
- `Session not valid`
- `Day? 2`
- `Session? S1`
- `Speaker name? Goofy`
- `Registration successful`

`ecc.`

Example of program overview:

- `Day? 1`
- `Program of the 1 conference day`

```
Session S1:
Session 1: Richard Feynman
Session 2: (not registered) ...
```

Project architecture

The RMI project consists of:

- **A remote interface** (***ConferenceServer*****,** defined in the file *ConferenceServer.java*) that defines the methods that can be remotely invocated by a client (registration, program);
- **A helper class** (***Program***, implemented in the file *Program.java*), that implements the data structure that stores all presentations for all sessions of a given day; for each day, there is a different program;
- **A server side class** (***ConferenceServerImpl***, implemented in the file *ConferenceServerImpl.java*), that implements all the methods available to the client;
- **A client side class** (***ConferenceClient***, implemented in the file *ConferenceClient.java*), that realizes user interaction and sends remote invocations.

Deployment

The RMI project consists of a Client and a Server – that are under user control and have to be started – and by the support needed by RMI as name system – ***the rmiregistry has to be started on the server node***.

The **Server** implements the invocation interface:

ConferenceServerImpl

The **Client** is started using the command line:

ConferenceClient HostName

The Client obtains the reference to the remote object from the rmiregistry running on the server host and uses it to invoke the methods (to register a presentation and to get the conference program) over the server.

Note: the activation order is i) rmiregistry, ii) Server, and iii) Client.

ConferenceServer Interface

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


public interface ConferenceServer extends Remote {

  // return true if successful
  boolean register(int day, String session,
                   String speaker) throws RemoteException;

  Program getProgram (int day) throws RemoteException;

}
```

Program helper class

```
public class Program implements Serializable
{  // class to store presentations of a conference day
public String speaker[][] = new String[12][5];

public Program(){  // init con stringa nulla
 for(int i=0; i<5; i++) for(int e=0; e<12; e++) speaker[e][i]="";
}
public synchronized boolean register (int session, String
     name){
  // return true if successful, else false
  for (int k=0;k<5; k++)
  { if ( speaker[session][k].equals("") )
    { speaker[session][k] = name; return true; }
  }
  return false;
}

public void print (){
  System.out.println("Session\tPresentation1\tPresentation2…");
  for (int k=0; k<12; k++)
  { String line = new String("S"+(k+1));
    for (int j=0;j<5;j++)
    { line = line + "\t\t"+speaker[k][j]; System.out.println(line); }
  }
}
}
```

Client 1/3

```
class ConferenceClient
{
  public static void main(String[] args)
  { final int REGISTRYPORT = 1099;
    String registryHost = null;
    String serviceName = "ConferenceServer";
    BufferedReader stdIn =
      new BufferedReader(new InputStreamReader(System.in));
   // Check command line arguments
    try
    {if (args.length != 1)
      { System.out.println("Syntax: … "); System.exit(1); }
     registryHost = args[0];

    // Connection to remote RMI service
     String completeName = "//" + registryHost +
        ":" + REGISTRYPORT + "/" + serviceName;
     ConferenceServer serverRMI =
        (ConferenceServer) Naming.lookup (completeName);
     System.out.println("\nRequests until EOF");
     System.out.print("Service (R=Registration, P=Get Program): ");
     String service; boolean ok;
```

Client 2/3

```
// user interaction
 while((service=stdIn.readLine())!=null)
  {if (service.equals("R"))
  { ok=false; int g;                          // select day
     System.out.print("Day (1-3)? ");
     while (ok!=true){
       g = Integer.parseInt(stdIn.readLine());
       if (g < 1 || g > 3)
       { System.out.println("Day not valid");
         System.out.print("Day(1-3)? "); continue;
       } else ok=true;
     }  // inner while
     ok=false; String sess;                   // select session
     System.out.print("Session (S1 - S12)? ");
     while (ok!=true){
       sess = stdIn.readLine();
       if ( !sess.equals("S1") && … !sess.equals("S12"))
       { ... continue; } else ok=true;
     }
     System.out.print("Speaker? ");     // input speaker
     String speak = stdIn.readLine();
  // Valid request parameter, do remote invocation
   if (serverRMI. register (gg, sess, speak))
       System.out.println("Registered …");
     else System.out.println("Registration failed");
  }
```

Client 3/3

```
else if (service.equals("P"))
{    int g; boolean ok=false;
     System.out.print("Day (1-3)? ");
     while (ok!=true){
       g = Integer.parseInt(stdIn.readLine());
       if (g < 1 || g > 3){
         System.out.println("Giornata non valida");
         System.out.print("Giornata (1-3)? ");
         continue;
       }else ok=true;
     }  // while
     Program prog = serverRMI.getProgram(g);
     System.out.println("Program for day "+g+"\n");
     prog.print();
   } // Operation P
   else System.out.println("Service unavailable");
   System.out.print("Service (R=Registration, …");
  }  // while
  }  //try
  catch (Exception e){ ... }

}   // main
} // ConferenceClient
```

Server 1/2

```
public class ConferenceServerImpl
     extends UnicastRemoteObject
     implements ServerCongresso
{ static Program prog[]; // create a program for each conference day

  // Constructor
 public ServerCongressoImpl()throws RemoteException {super(); }

  // REMOTE METHOD: registration request
 public bool register (int day, String session,
                String speaker) throws RemoteException
  { int numSess = -1;
  System.out.println("RMI Server: registration request:");
  if (sessione.equals("S1")) numSess = 0;
  else if (sessione.equals("S2")) numSess = 1;
  …
  else if (sessione.equals("S12")) numSess = 11;
  /* If no valid session is found, raise an exception */
  if (numSess == -1) throw new RemoteException();
  if (day < 1 || day > 3) throw new RemoteException();
  return prog[giorno-1].register(numSess,speaker);
  }
```

Server 2/2

```
  // REMOTE METHOD: complete program request
 public Program getProgram (int day)throws RemoteException
  { System.out.println("RMI server: program for day "+giorno);
    if (giorno < 1 || giorno > 3) throw new RemoteException();
    return prog[giorno-1];
  }

public static void main (String[] args) // main to start RMI server
  { prog = new Program[3]; // create programs
  for (int i=0; i<3; i++)  prog[i]= new Program();

  final int REGISTRYPORT = 1099;
  String registryHost = "localhost";
  String serviceName = "ConferenceServer";
  try
  {           // Register RMI server
   String completeName = "//" + registryHost +
                    ":" + REGISTRYPORT + "/" + serviceName;
   ServerCongressoImpl serverRMI = new ServerCongressoImpl();
   Naming.rebind (completeName, serverRMI);
  }   // try
  catch (Exception e){ ... }
  }   /* main */ }
```