



**Università degli Studi di Bologna  
Scuola di Ingegneria**

# **Corso di Reti di Calcolatori T**

**Esercitazione 6 (svolta)  
Java RMI**

**Antonio Corradi, Luca Foschini**

**Michele Solimando, Giuseppe Martuscelli, Marco Torello**

**Anno accademico 2020/2021**

# SPECIFICA: IL CLIENT

---

Si progetti un'applicazione Client/Server per la **gestione delle registrazioni ad un congresso**. L'organizzazione del congresso fornisce agli speaker delle varie sessioni un'interfaccia tramite la quale iscriversi ad una sessione, e la possibilità di visionare i programmi delle varie giornate del congresso, con gli interventi delle varie sessioni

Il **Client** può richiedere operazioni per:

- **registrare uno speaker** ad una sessione
- **ottenere il programma** del congresso

Il Client è implementato come un **processo ciclico** che continua a fare richieste sincrone fino ad esaurire tutte le esigenze utente, cioè fino alla fine del file di input dell'utente

# SPECIFICA: IL SERVER

---

Il **Server** mantiene sul suo nodo di residenza i programmi delle 3 giornate del congresso, ciascuno dei quali è memorizzato in una struttura dati in cui ad ogni riga corrisponde una sessione (in tutto 12 per ogni giornata)

Per ciascuna sessione vengono memorizzati i nomi degli speaker che si sono registrati (al massimo 5)

Sessione	Intervento 1	Intervento 2	...	...	Intervento 5
S1	Nome Speaker1	Nome Speaker2			
S2					
...					
S12					

# SPECIFICA: DETTAGLI ULTERIORI

---

Il **Client** inoltra le richieste al **Server** in modo appropriato, e per ogni possibile operazione prevedono anche una **gestione di eventuali condizioni anomale** (come per esempio la richiesta di registrazione ad una giornata/sessione inesistente oppure per la quale sono già stati coperti tutti gli spazi d'intervento). Si effettuino i controlli dove è più opportuno farli

Alcuni esempi di interazione per la richiesta di registrazione:

```
>Giornata? 25
>Giornata non valida
>Giornata? 2
>Sessione? S46
>Sessione non valida
>Giornata? 2
>Sessione? S1
>Nome speaker? Pippo
>Registrazione effettuata correttamente
ecc.
```

Alcuni esempi di interazione per la visione del programma:

```
>Giornata? 1
>Programma della prima giornata del congresso
Sessione S1:
primo intervento: NomeSpeaker1
secondo intervento: non registrato ...
```

# PROGETTO E SUE PARTI

---

Il progetto RMI si compone di:

- Una **interfaccia remota** (***ServerCongresso***, contenuta nel file *ServerCongresso.java*) in cui vengono definiti i metodi invocabili in remoto dal client (registrazione, programma)
- Una **classe di appoggio** (***Programma*** contenuta nel file *Programma.java*), che implementa la struttura dati contenente gli interventi delle varie sessioni; si noti che i programmi delle varie giornate andranno poi gestiti con una opportuna struttura dati che li raccolga tutti
- Una **classe per la parte server** (***ServerCongressoImpl*** contenuta nel file *ServerCongressoImpl.java*), che implementa i metodi del server invocabili in remoto
- Una **classe per la parte client** (***ClientCongresso*** contenuta nel file *ClientCongresso.java*), che realizza l'interazione con l'utente e effettua le opportune chiamate remote

# DEPLOYMENT

---

Il progetto RMI si compone delle due parti **Cliente** e **Servitore**, che sono sotto il controllo utente e da attivare, e anche della parte di supporto resa necessaria ad RMI per il sistema di nomi, il registry da attivare sul nodo del servitore

Il **Server** presenta l'interfaccia di invocazione:  
***ServerCongressoImpl***

Il **Client** viene attivato con:  
***ClientCongresso NomeHost***

Il Client (istanza della classe relativa) deve recuperare dal registry, in esecuzione sull'host specificato, il riferimento all'oggetto remoto, con interfaccia ServerCongresso, di cui deve invocare i metodi

**L'ordine di attivazione è prima il registry, poi il server, infine la parte cliente**

# INTERFACCIA SERVERCONGRESSO

---

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface ServerCongresso extends Remote  
{  
  
    int registrazione (int giorno, String sessione,  
                      String speaker)  
        throws RemoteException;  
  
    Programma programma (int giorno)  
        throws RemoteException;  
  
}
```

# CLASSE PROGRAMMA DI APPOGGIO

---

```
public class Programma implements Serializable
{ // classe per modellare ogni giornata del convegno
public String speaker[][] = new String[12][5];
public Programma() { // init con stringa nulla
    for(int i=0; i<5; i++) for(int e=0; e<12; e++) speaker[e][i]="";
}
public synchronized int registra (int sessione, String nome)
{for (int k=0;k<5; k++)
    { if ( speaker[sessione][k].equals("") )
        { speaker[sessione][k] = nome; return 0; }
    }
    return 1;
}
public void stampa ()
{ System.out.println("Sessione\tIntervento1\tIntervento2...");
    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);
        } }
} } // Programma
```



# CLIENT 1/3

---

```
class ClientCongresso
{
public static void main(String[] args) // processo cliente
{
    final int REGISTRYPORT = 1099;
    String registryHost = null;
    String serviceName = "ServerCongresso";
    BufferedReader stdIn = new BufferedReader
        (new InputStreamReader(System.in));
    try // Controllo dei parametri della riga di comando
    {
        if (args.length != 1)
        {
            System.out.println("Sintassi:...");
            System.exit(1);
        }
        registryHost = args[0];
        // Connessione al servizio RMI remoto
        String completeName = "/" + registryHost + ":" +
            REGISTRYPORT + "/" + serviceName;
        ServerCongresso serverRMI =
            (ServerCongresso) Naming.lookup (completeName);
        System.out.println("\nRichieste a EOF");
        System.out.print("Servizio(R=Registrazione, P=Programma): ");
        String service; boolean ok;
```

# CLIENT 2/3

---

// Ciclo di interazione con l'utente per chiedere operazioni

```
while ( (service=stdin.readLine()) !=null)
{if (service.equals("R"))
{ ok=false; int g; // lettura giornata
  System.out.print("Giornata (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 interno
  ok=false; String sess; // lettura sessione
  System.out.print("Sessione (S1 - S12)? ");
  while (ok!=true){
    sess = stdin.readLine();
    if ( !sess.equals("S1") && ... !sess.equals("S12"))
    { ... continue; } else ok=true;
  }
  System.out.print("Speaker? "); // lettura speaker
  String speak = stdin.readLine();
  // Parametri corretti, invoco il servizio remoto
  if (serverRMI.Registrazione(gg, sess, speak)==0)
    System.out.println("Registrazione di ...");
  else System.out.println("Registrazione non effettuata");
}
```

# CLIENT 3/3

---

```
else if (service.equals("P"))
{
    int g; boolean ok=false;
    System.out.print("Giornata (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
    Programma prog = serverRMI.programma(g);
    System.out.println("Programma giornata "+g+"\n");
    prog.stampa();
} // Operazione P
else System.out.println("Servizio non disponibile");
System.out.print("Servizio (R=Registrazione, ...)");
} // while
} // try
catch (Exception e){ ... }
} // main
} // ClientCongresso
```

# SERVER 1/2

---

```
public class ServerCongressoImpl
    extends UnicastRemoteObject
    implements ServerCongresso
{
    static Programma prog[]; // si istanzia un programma per giornata

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

    // METODO REMOTO: Richiesta di prenotazione
    public int registrazione (int giorno, String sessione,
                              String speaker) throws RemoteException
    {
        int numSess = -1;
        System.out.println("Server RMI: richiesta registrazione:");
        if (sessione.equals("S1")) numSess = 0;
        else if (sessione.equals("S2")) numSess = 1;
        ...
        else if (sessione.equals("S12")) numSess = 11;
        /* Se i dati sono sbagliati significa che sono stati trasmessi male e quindi si solleva una
           eccezione */
        if (numSess == -1) throw new RemoteException();
        if (giorno < 1 || giorno > 3) throw new RemoteException();
        return prog[giorno-1].registra(numSess, speaker);
    }
}
```

# SERVER 2/2

---

```
// METODO REMOTO: Richiesta di programma
public Programma programma (int giorno) throws RemoteException
{ System.out.println("Server RMI: programma giorno
  "+giorno);
  if (giorno < 1 || giorno > 3) throw new RemoteException();
  return prog[giorno-1];
}

public static void main (String[] args) // Codice di avvio del Server
{ prog = new Programma[3]; //creazione dei programmi per le giornate
  for (int i=0; i<3; i++)      prog[i]= new Programma();

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