

# **Esercitazioni Ing.Sw**

Alessandro Margara / Gian Enrico Conti  
II sem 2023

Webex:

<https://politecnicomilano.webex.com/meet/gianenrico.conti>

# RMI chat

ing. Gian Enrico Conti

Prova Finale - Ingegneria del Software - AA 2022/23

La logica iniziale:

- una classe server
- una classe client

New project..

Due file .. "ChatClientApp" / "ChatServerApp" X ora vuoti..


```
public class ChatServerApp
{
    public static void main( String[] args )
    {
        System.out.println( "Hello from ChatServerApp!" );
    }
}
```

```
public class ChatClientApp {
    public static void main( String[] args )
    {
        System.out.println( "Hello from ChatClientApp!" );
    }
}
```

La logica iniziale:

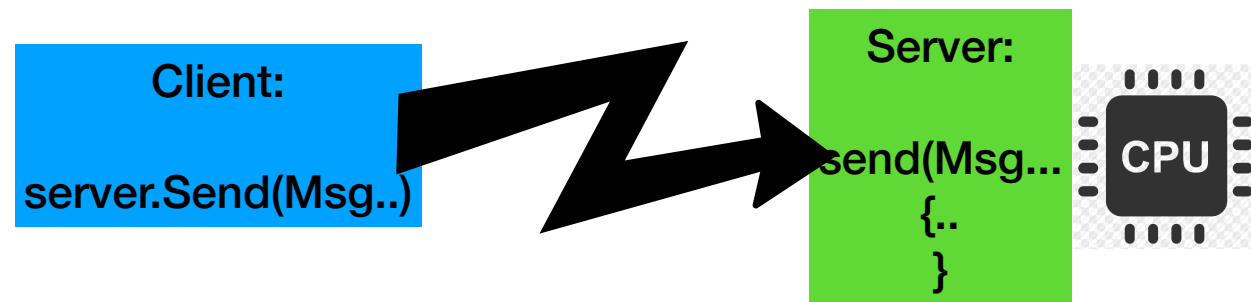
- una classe server
- una classe client
- una interfaccia di COSA in client puo fare (remotamente) sul server
- una interfaccia di COSA il server puo fare (remotamente) sul client

**I.e. NON esiste metodo receive sul server !**



Attenzione!  
Dovete  
pensare a  
rovescio!

- Si connette (login)
- Manda messaggi



..

- Si connette (login)
- Manda messaggi

Interfaccia:

```
public interface ChatServer extends Remote{  
    void login(ChatClient cc) throws RemoteException;  
    void send (String message) throws RemoteException;  
}
```

Non esiste.. ok..

E il SERVER le implementerà!

- Fa solo broadcast dei messaggi ricevuti (sia send)
- (NON riceve!)

Interfaccia:

```
public interface ChatClient extends Remote {  
    void receive (String message) throws RemoteException;  
}
```

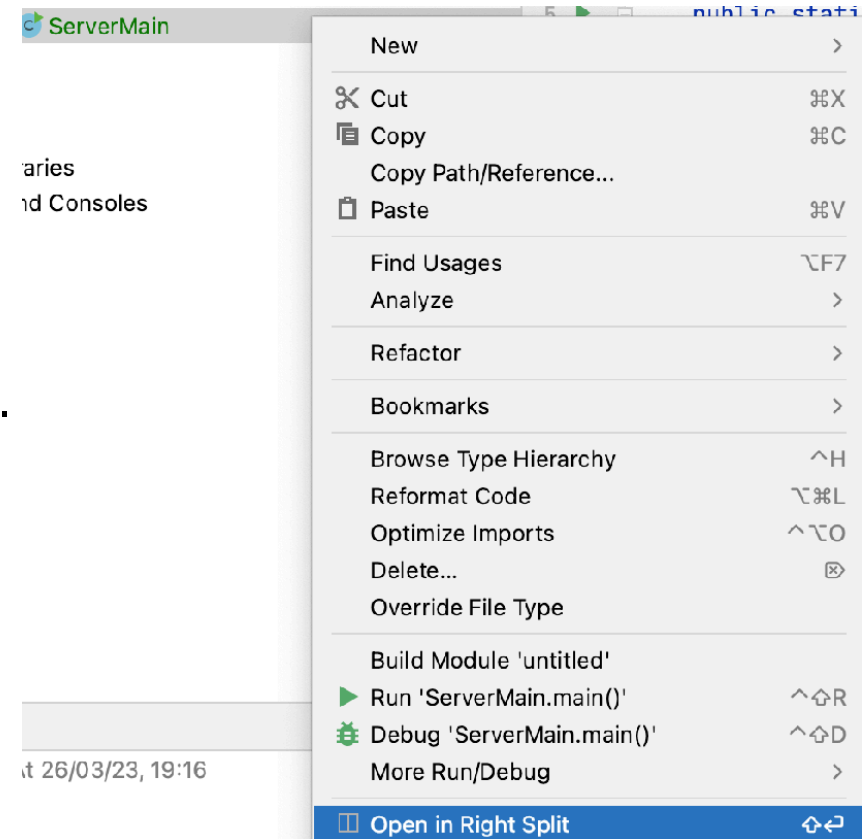
Due file .. "ChatClient" / "ChatServer".  
copiamo.. avremo un po' di msg .. alt↵

(Nota: partite da **client**..)



Un run di prova.. cosi abbiamo le config.. e poi split window x debug:

E posizioneremo i break point piu avanti..



Deve accettare invocazioni remote:

```
extends UnicastRemoteObject
```

Ma implementa la interfaccia:

```
extends UnicastRemoteObject implements ChatServer
```

Quindi:

```
public class ChatServerApp extends UnicastRemoteObject implements ChatServer
```

Appena lo faremo vari complaint..

complaint.. import.. ok..

Interfaccia:



- Serve list dei client che si connettono
- Le interfacce vanno riempite
- Main codice avvio server

- Serve list dei client che si connettono:

```
private final List<ChatClient> chatClients;  
  
public ChatServerApp() throws RemoteException {  
    this.chatClients = new ArrayList<>();  
}
```

- Le interfacce vanno riempite:

```
public void login(ChatClient cc) throws RemoteException {  
    this.chatClients.add(cc);  
}  
  
public void send (String message) throws RemoteException {  
    System.out.println ("server received: " + message);  
    for (ChatClient cc : chatClients) {  
        cc.receive(message);  
    }  
}  
}
```

- Main codice avvio server

```
public static void main (String[] args)
{
    try {
        new ChatServerApp().startServer();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



try...

```
private void startServer() throws RemoteException {

    // Bind the remote object's stub in the registry
    //DO NOT CALL Registry registry = LocateRegistry.getRegistry();
    Registry registry = LocateRegistry.createRegistry(Settings.PORT);
    try {
        registry.bind("ChatService", this);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("Server ready");

}
```



Commenti...

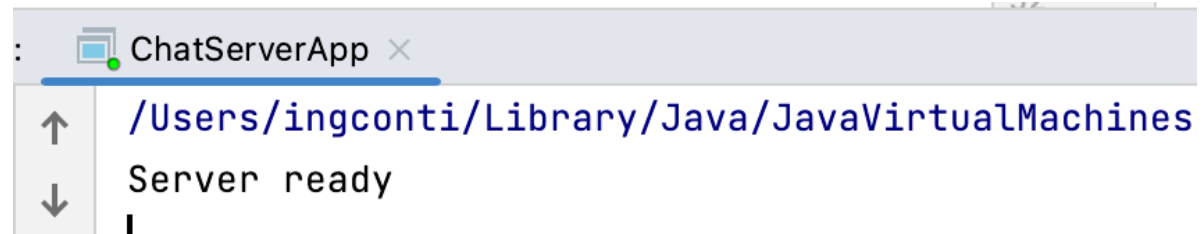
Non trova porta.. **Settings.PORT..**

- Siamo ordinati:

```
public class Settings {  
    static int PORT = 1234;  
    static String SERVER_NAME = "127.0.0.1";  
}
```

- Run del server..





```
ChatServerApp x
/Users/ingconti/Library/Java/JavaVirtualMachines
Server ready
|
```

- NON termina
- È GIA IN Loop...
- RMI ha pool di thread.. runnable... socket.. all for free.

- Anche cl. f. Remote, e implementa la SUA interfaccia:

```
public class ChatClientApp extends UnicastRemoteObject implements ChatClient {
```

Soliti complaints e interfacce..

```
public void receive(String message) throws RemoteException {
```

```
}
```

```
..
```

- Riempiamo:

```
public void receive (String message) throws RemoteException {  
    System.out.println(message);  
}
```

E' preferibile avere un rif. all' oggetto server:

```
private ChatServer cs;
```

.. main..

- "Core code"

```
private void startClient() throws Exception {  
    // Getting the registry  
    Registry registry;  
  
    registry = LocateRegistry.getRegistry(Settings.SERVER_NAME,  
        Settings.PORT);  
  
    // Looking up the registry for the remote object  
    this.cs = (ChatServer) registry.lookup("ChatService");  
    this.cs.login(this);  
  
    inputLoop();  
}
```

Leggiamo  
KBD

Ok.. input loop e main..

"ChatService" should be the SAME in:

CL:  
registry.bind("ChatService", this);

SRV:  
this.cs = (ChatServer) registry.lookup("ChatService");

input loop e main..

```
public static void main( String[] args )
{
    try {
        new ChatClientApp().startClient();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



try...

```
void inputLoop() throws IOException {
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    String message;

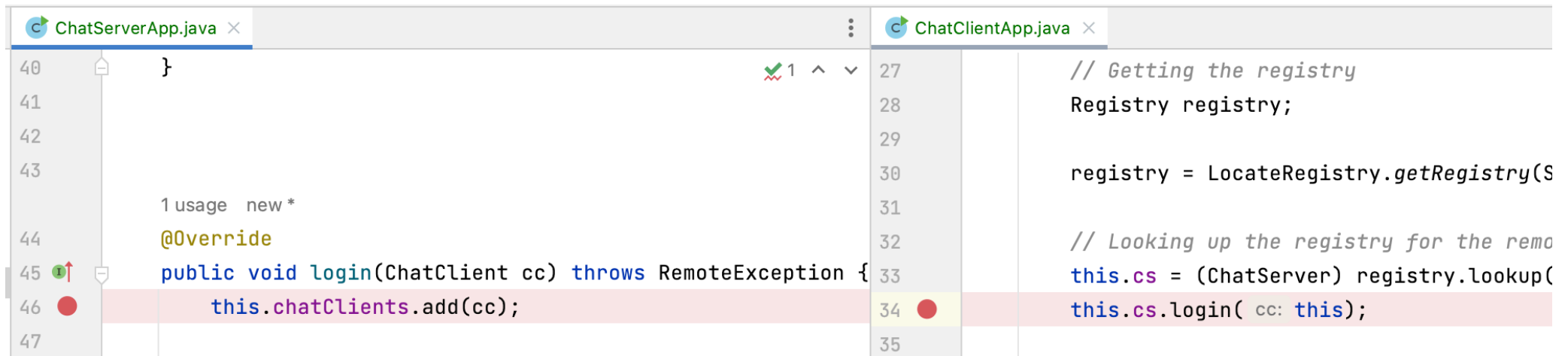
    while ( (message = br.readLine ()) != null) {
        cs.send(message);
    }
}
```

Build del client..

Le interfacce di RMI vogliono costruttore...



Mettiamo breakpoint e partiamo in debug (*prima server*)



Run del client.. si ferma al ns breakpoint..

Step over..

Debug passa al server!

..

Free run del server..

console del client e free run.. scriviamo chars..

Lato client:

```
/Users/ingconti/Libr  
Connected to the tar  
hello  
hello
```

Lato server:

```
ChatServerApp x ChatClientApp  
/Users/ingconti/Library/Java/  
Server ready  
server received: hello
```

Ok.. ma si vede il ns stesso messaggio..(to be done)..  
More Clients... shell.. not here!



Poiché in Java esiste la alberatura dei package,  
(Nel ns caso .... org.example)

Per usare la cmd line, portarsi SULLA cartella ....main/java:

Quindi pes.

```
cd /Users/ingconti/IdeaProjects/RMI_CHAT_EX3/src/main/java
```

E qui:

```
javac org/example/ChatServerApp.java
```

```
javac org/example/ChatClientApp.java
```

Run (prima server):

```
java org/example/ChatServerApp
```

```
...
```

```
java org/example/ChatClientApp
```

## Note:

- A differenza del server TCP, il server rimane sempre in attesa, anche senza i loop che abbiamo fatto
  - RMI è pensato per mantenere viva la connessione lato server per garantire l'invocazione remoto dei client
  - provate a lasciare aperta una SHELL con il server e a connettere/disconnettere più volte il client...
-

- Ragionate in modo da eseguire comandi sul server
- sforzatevi di ragionare a rovescio, è corretto!
- L'implementazione RMI moltissimo lavoro per voi
- **attenzione:**
  - RMI è sincrono!
  - Avete comunque problemi di sincronizzazione nel modello, i metodi sono eseguiti sul proprio thread

Se ho TCP... (see slide "RMI vs TCP")