



Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

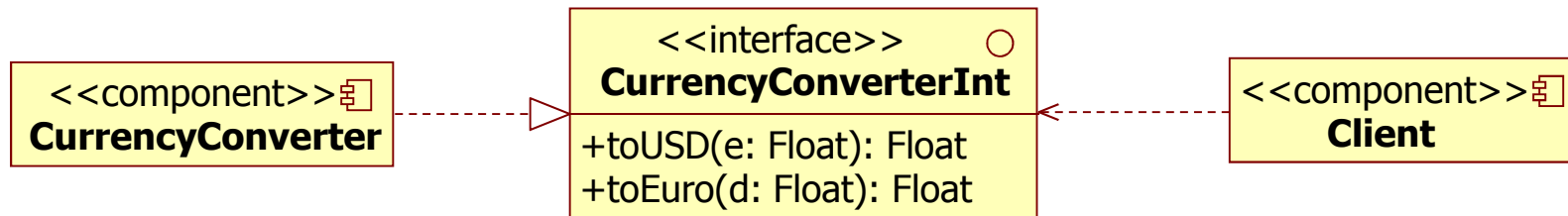
Programmazione Concorrente e Distribuita Esempi di programmazione con RMI

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it



Currency Converter

- Consente di convertire un valore da EURO a DOLLARI e viceversa.



- Cominciamo a vedere una implementazione locale (il CurrencyConverter e il suo client sono oggetti dello stesso processo).



CurrencyConverterInterface

```
public interface CurrencyConverterInterface {  
    float toEur(float usd);  
    float toUsd(float eur);  
}
```



CurrencyConverter

```
public class CurrencyConverter
    implements CurrencyConverterInterface {
    public CurrencyConverter() { }
    public float toEur(float usd) {
        return usd*0.895415473F;
    }
    public float toUsd(float eur) {
        return eur*1.114365F;
    }
}
```



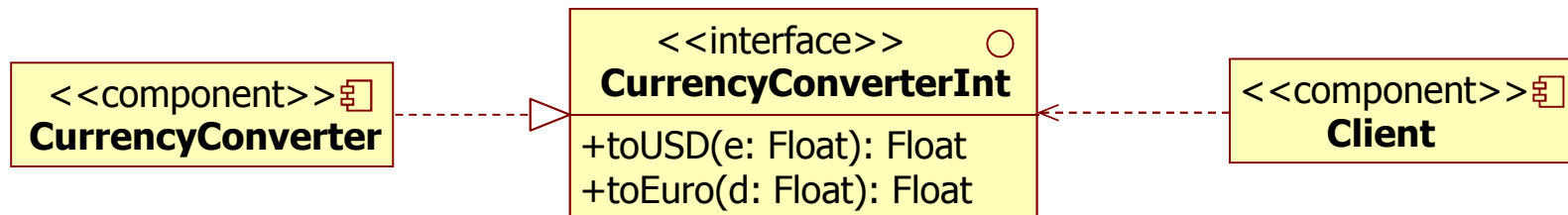
CurrencyConverterClient

```
public class CurrencyConverterClient {
    public static void main(String[] args) {
        try {
            CurrencyConverter stub = new CurrencyConverter();
            for(int usd = 1; usd < 10; usd++){
                System.out.println(usd + " USD = " +
                                   stub.toEur(usd) + " EUR");
            }
            for(int eur = 1; eur < 10; eur++){
                System.out.println(eur + " EUR = " +
                                   stub.toUsd(eur) + " USD");
            }
        } catch (Exception e) {
            System.err.println("Client exc.: " + e.toString());
            e.printStackTrace();
        }
    }
}
```



Currency Converter distribuito

- I client chiedono le conversioni al server
- Il lavoro di conversione viene fatto dal server
- I risultati delle conversioni vengono restituiti ai client
- Concettualmente non cambia nulla rispetto alla situazione locale.
- Ma l'oggetto server e gli oggetti client possono stare su macchine diverse.





Interfaccia remota **CurrencyConverter**

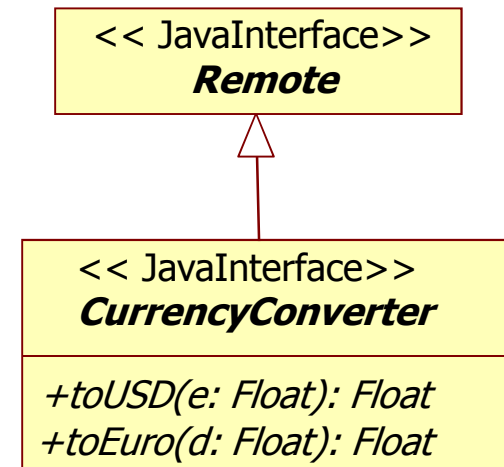
- L'interfaccia che definisce le funzionalità del Server remoto (cioè accessibile da altri oggetti non locali tramite il sistema RMI) devono ereditare da **`java.rmi.Remote`**
- I metodi definiti in tali interfacce devono dichiarare l'eccezione **`java.rmi.RemoteException`**



Interfaccia remota **CurrencyConverter**

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

```
public interface CurrencyConverter extends Remote {  
    float toEur(float usd) throws RemoteException;  
    float toUsd(float eur) throws RemoteException;  
}
```





CurrencyConverterImpl

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CurrencyConverterImpl implements CurrencyConverter{

    public CurrencyConverterImpl() { }
    public float toEur(float usd) throws RemoteException {
        return usd*0.895415473F;
    }
    public float toUsd(float eur) throws RemoteException {
        return eur*1.114365F;
    }
}
```

Non estende
UnicastRemoteObject



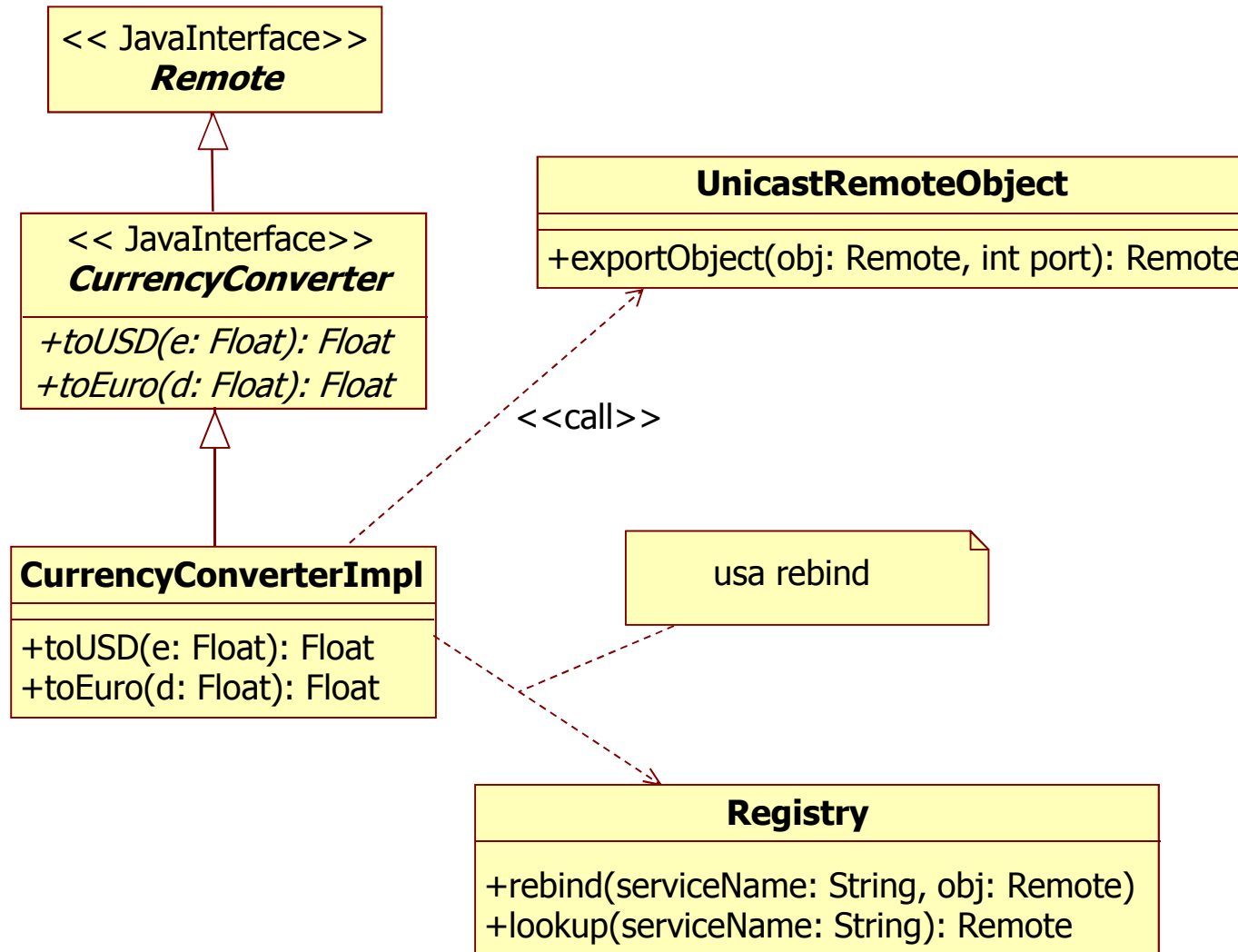
CurrencyConverterImpl

```
public static void main(String args[]) {  
    try {  
        CurrencyConverterImpl obj = new CurrencyConverterImpl();  
        CurrencyConverter stub = (CurrencyConverter)  
            UnicastRemoteObject.exportObject(obj, 3939);  
        Registry registro = LocateRegistry.createRegistry(1099);  
        registry.rebind("CurrencyConverter", stub);  
        System.err.println("Server ready");  
    } catch (Exception e) {  
        System.err.println("Server exception: " + e.toString());  
        e.printStackTrace();  
    }  
}
```

Crea un oggetto normale,
poi lo converte a
UnicastRemoteObject



CurrencyConverterImpl



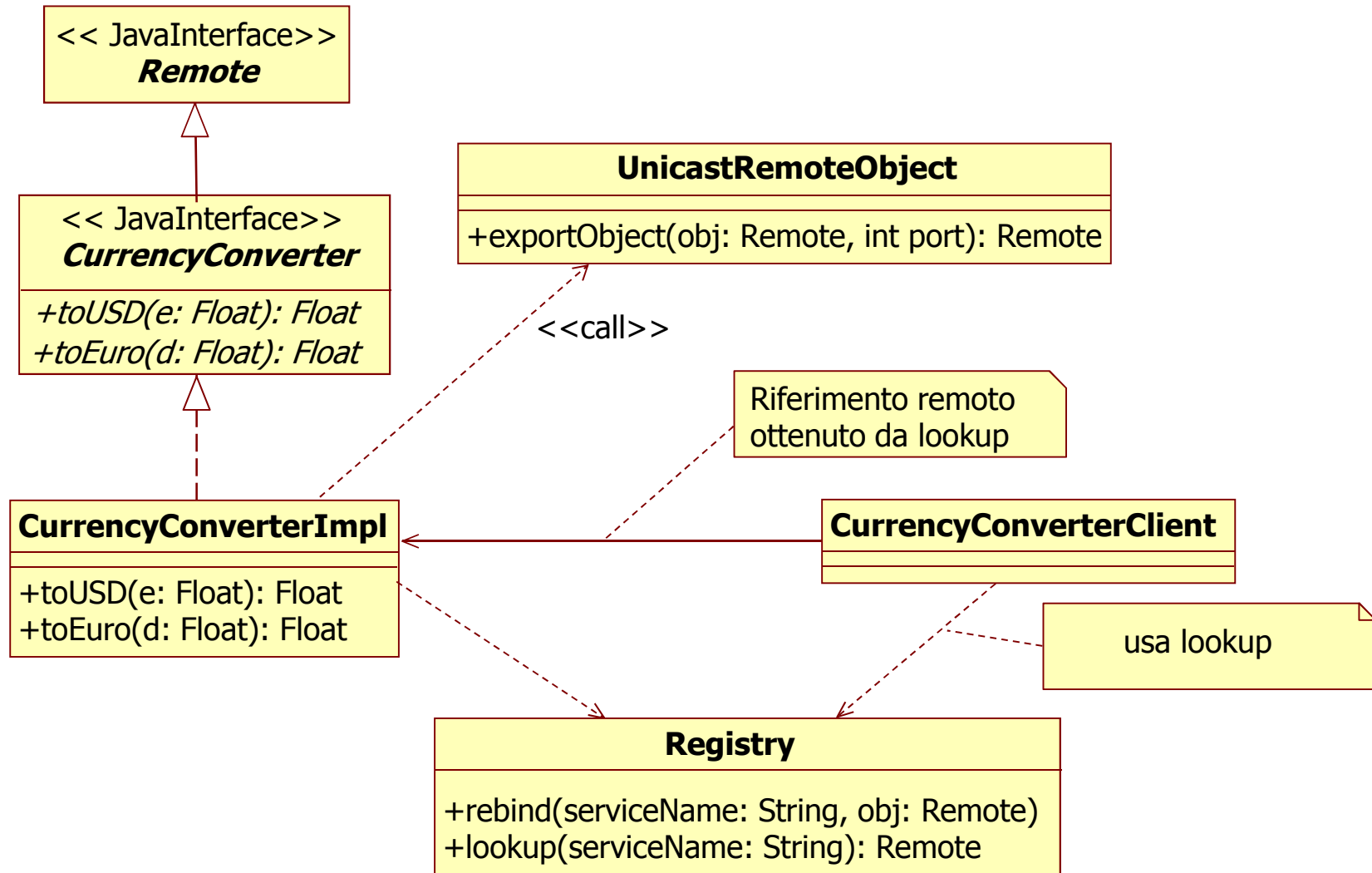


CurrencyConverterClient

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class CurrencyConverterClient {
    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            CurrencyConverter stub = (CurrencyConverter)
                registry.lookup("CurrencyConverter");
            for(int usd = 1; usd < 10; usd++){
                System.out.println(usd+" USD = "+stub.toEur(usd)+" EUR");
            }
            for(int eur = 1; eur < 10; eur++){
                System.out.println(eur+" EUR = "+stub.toUsd(eur)+" USD");
            }
        } catch (Exception e) {
            System.err.println("Client exc.: " + e.toString());
        }
    }
}
```

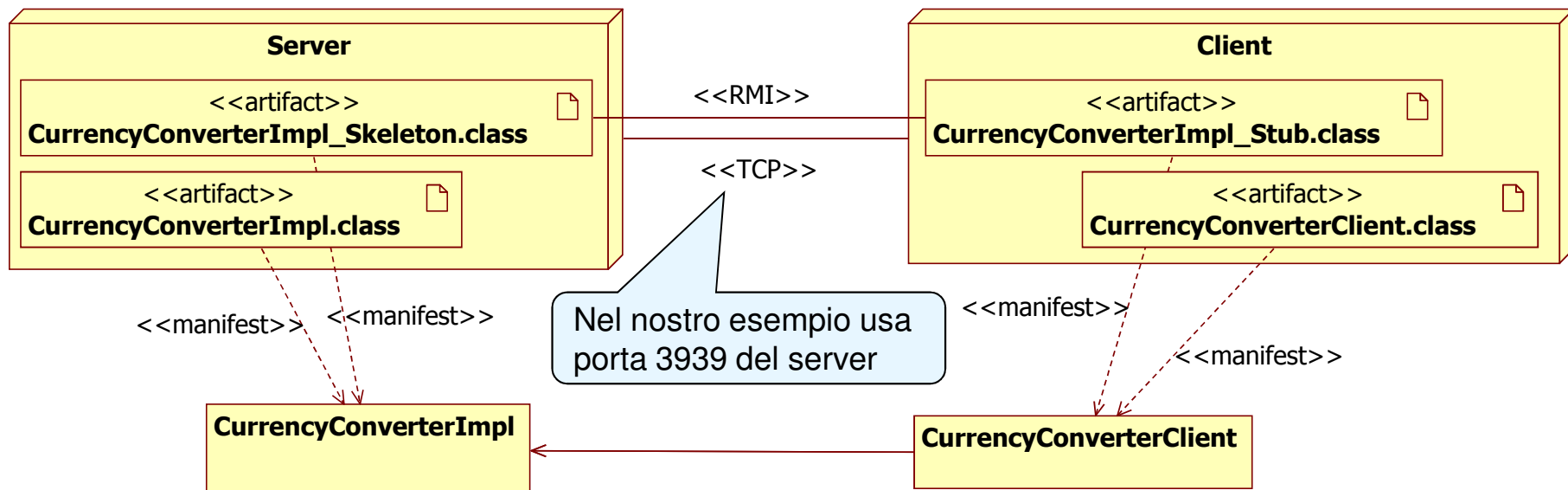


CurrencyConverterImpl



Deployment e situazione a run-time

NB: registry omissso



- NB: nelle versioni recenti di Java, stub e skeleton non sono espliciti, ma stanno dentro il client e il server rispettivamente.
 - Logicamente non cambia nulla.



Estensione: passiamo un oggetto come argomento

- Il client fornisce come parametro un oggetto di classe “conversione”, contenente:
 - ▶ La somma data,
 - ▶ Valuta in cui è espressa la somma data
 - ▶ Valuta in cui convertire la somma data
- E riceve un oggetto contenente le stesse informazioni più la somma ottenuta come risultato della conversione.

Per semplicità usiamo la stessa classe per le richieste e per le risposte.



Class Conversion

```
import java.io.Serializable;

public class Conversion implements Serializable {
    private static final long serialVersionUID = 1L;
    float givenAmount;
    String givenCurrency;
    float targetAmount;
    String targetCurrency;
    Conversion(float inAmount, String inCurrency,
               String outCurrency) {
        givenAmount=inAmount;
        givenCurrency=inCurrency;
        targetCurrency=outCurrency;
        targetAmount=-1;
    }
}
```




Class Conversion

```
public float getGivenAmount() {
    return givenAmount;
}
public String getGivenCurrency() {
    return givenCurrency;
}
public float getTargetAmount() {
    return targetAmount;
}
public String getTargetCurrency() {
    return targetCurrency;
}
public void setTargetAmount(float targetAmount) {
    this.targetAmount = targetAmount;
}
public String toString() {
    return ""+givenAmount+" "+givenCurrency+" = "+targetAmount+"
        "+targetCurrency;
}
}
```



CurrencyConverterInterface

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

```
public interface CurrencyConverterInterface extends Remote {  
    Conversion compute(Conversion conv) throws RemoteException;  
}
```



Class CurrencyConverter

```
import java.rmi.RemoteException;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;

public class CurrencyConverter extends UnicastRemoteObject
implements CurrencyConverterInterface {
    private static final long serialVersionUID = 1L;
    public CurrencyConverter() throws RemoteException {
        super();
    }
    private float toEur(float usd) {
        return usd*0.895415473F;
    }
    private float toUsd(float eur) {
        return eur*1.114365F;
    }
}
```



Class CurrencyConverter

```
public Conversion compute(Conversion conv) {
    float convertedAmount=0;
    if(conv.givenCurrency.equals("USD")) {
        convertedAmount=toEur(conv.getGivenAmount());
    }
    if(conv.givenCurrency.equals("EUR")) {
        convertedAmount=toUsd(conv.getGivenAmount());
    }
    conv.setTargetAmount(convertedAmount);
    return conv;
}

public static void main(String args[]) {
    try {
        CurrencyConverter obj = new CurrencyConverter();
        Registry registro = LocateRegistry.createRegistry(1099);
        registro.rebind("CurrencyConverter", obj);
        System.err.println("Server ready");
    } catch (Exception e) {
        System.err.println("Server exception: " + e.toString());
    } } }
```



Class CurrencyConverterClient

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.*;
import java.util.Random;

public class CurrencyConverterClient {
    public static void main(String[] args) {
        try {
            new CurrencyConverterClient().exec();
        } catch (RemoteException | NotBoundException e) {
            e.printStackTrace();
        }
    }
}
```



Class CurrencyConverterClient

```
void exec() throws RemoteException, NotBoundException {
    Conversion conv;
    Random rnd=new Random();  int times=2+rnd.nextInt(10);
    String fromCurrency, toCurrency;
    Registry registro = LocateRegistry.getRegistry(1099);
    CurrencyConverterInterface stub =
        (CurrencyConverterInterface)
        registro.lookup("CurrencyConverter");
    for(int i=0; i<times; i++) {
        try {
            if(rnd.nextBoolean()) {
                fromCurrency="USD"; toCurrency="EUR";
            } else {
                fromCurrency="EUR"; toCurrency="USD";
            }
            conv = new Conversion(rnd.nextFloat()*100,
                                fromCurrency, toCurrency);
            System.out.println(stub.compute(conv));
        } catch (Exception e) {
            System.err.println("Client exc.: " + e.toString());
        }
    }
}
```

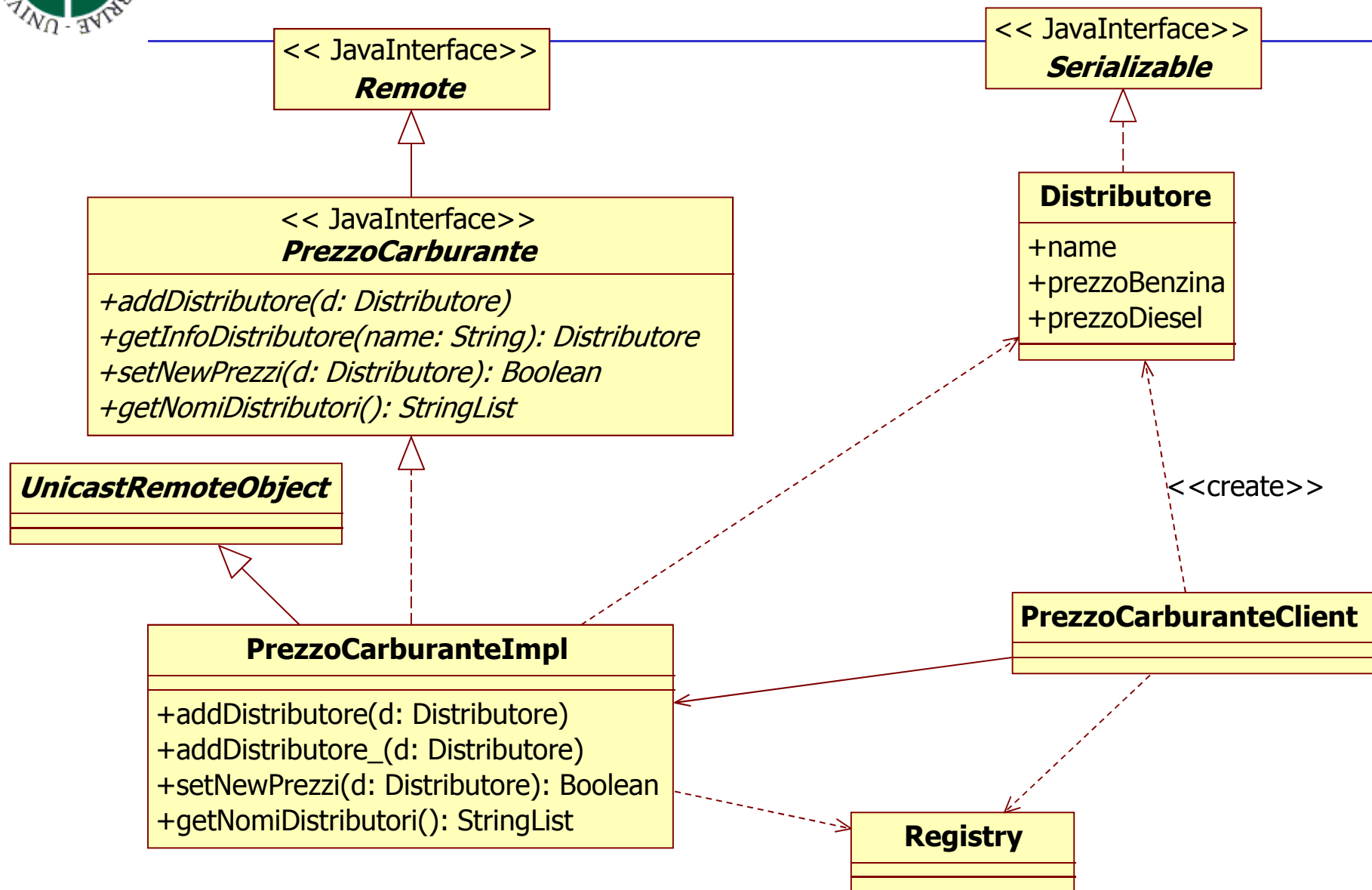


Esempio: Prezzi Benzina

- Consente di ottenere i prezzi del carburante da un Distributore
- Il lavoro di memorizzazione dei prezzi viene fatto dal server
- I prezzi del carburante sono letti dai client.



Prezzi carburante





Interfaccia PrezzoCarburante

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

public interface PrezzoCarburante extends Remote {
    Distributore getInfoDistributore(String name)
                                   throws RemoteException;
    boolean setNewPrezzi(Distributore distributore)
                                   throws RemoteException;
    void addDistributore(Distributore distributore)
                                   throws RemoteException;
    List<String> getNomiDistributori() throws RemoteException;
}
```



Distributore

```
public class Distributore implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private double prezzoBenzina;
    private double prezzoDiesel;
    public Distributore(String name, double b, double d) {
        this.name = name;
        this.prezzoBenzina = b;
        this.prezzoDiesel = d;
    }
    public String getName() {
        return name;
    }
    public double getPrezzoBenzina() {
        return prezzoBenzina;
    }
    public void setPrezzoBenzina(double prezzoBenzina) {
        this.prezzoBenzina = prezzoBenzina;
    }
}
```



Distributore

```
public double getPrezzoDiesel() {  
    return prezzoDiesel;  
}  
public void setPrezzoDiesel(double prezzoDiesel) {  
    this.prezzoDiesel = prezzoDiesel;  
}  
public String toString(){  
    return "Distributore: " + name  
        + "\n\tbenzina: " + prezzoBenzina  
        + "\n\tdiesel: " + prezzoDiesel;  
}  
}
```



PrezzoCarburanteImpl

```
import java.util.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class PrezzoCarburanteImpl implements PrezzoCarburante {
    Hashtable<String, Distributore> distributori =
        new Hashtable<String, Distributore>();
    public Distributore getInfoDistributore(String name)
        throws RemoteException {
        return distributori.get(name);
    }
}
```



PrezzoCarburanteImpl

```
public boolean setNewPrezzi(Distributore distr)
                                throws RemoteException {
    Distributore d = distributori.get(distr.getName());
    if(d==null){
        System.err.println("setNewPrezzi: il distributore "+
                           distr.getName()+ " non esiste");
        return false;
    }
    d.setPrezzoBenzina(distr.getPrezzoBenzina());
    d.setPrezzoDiesel(distr.getPrezzoDiesel());
    return true;
}

public List<String> getNomiDistributori()
                                throws RemoteException {
    Enumeration<String> nomiDistr = distributori.keys();
    List<String> list = Collections.list(nomiDistr);
    return list;
}
```



PrezzoCarburanteImpl

```
public void addDistributore(Distributore distr)
                                throws RemoteException {
    if (distributori.containsKey(distr.getName())) {
        setNewPrezzi(distr);
        System.out.println("aggiornati prezzi per distributore "+
                            distr.getName());
    } else {
        distributori.put(distr.getName(), distr);
        System.out.println("aggiunto nuovo distributore " +
                            distr.getName());
    }
}
```



PrezzoCarburanteImpl

```
public static void main(String args[]) {
    try {
        PrezzoCarburanteImpl obj = new PrezzoCarburanteImpl();
        PrezzoCarburante stub = (PrezzoCarburante)
            UnicastRemoteObject.exportObject(obj, 3939);
        Registry registry = LocateRegistry.createRegistry(1099);
        registry.rebind("Prezzi", stub);
        System.err.println("Server ready");
    } catch (Exception e) {
        System.err.println("Server exception: " + e.toString());
        e.printStackTrace();
    }
}
```



PrezzoCarburanteClient

```
import java.util.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
public class PrezzoCarburanteClient {
    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            PrezzoCarburante stub = (PrezzoCarburante)
                                    registry.lookup("Prezzi");
            int numDistr = (int) (Math.random()*100);
            for(int tappa=0; tappa<numDistr ; tappa++){
                // ...
            }
        } catch (Exception e) {
            System.err.println("Client exc: " + e.toString());
        }
    }
}
```




PrezzoCarburanteClient

```
for(int tappa=0; tappa<numDistr ; tappa++){
    String distrName = "d"+(int) (Math.random()*10);
    Distributore resp = stub.getInfoDistributore(distrName);
    if(resp==null){
        System.out.println(distrName + " sconosciuto: aggiungo!");
        stub.addDistributore(new Distributore(distrName,
                                                Math.random()+1, Math.random()+1));
    } else { // aggiorni i prezzi
        System.out.println("prezzi vecchi: " + resp);
        stub.setNewPrezzi(new Distributore(distrName,
                                            Math.random() + 1, Math.random()+1));
        Distributore distribNew =
            stub.getInfoDistributore(distrName);
        System.out.println("prezzi nuovi: " + distribNew);
    }
    List<String> list = stub.getNomiDistributori();
    System.out.println("Num distrib. salvati: " + list.size());
    // viaggio ancora per poi rifermarmi ad un nuovo distributore
    Thread.sleep((long) (Math.random()*1000));
}
```



Come fermare ordinatamente il server

- Se fermiamo il server semplicemente uccidendo il processo, può darsi che «sotto» qualcosa resti in uno stato inconsistente.
- Aggiungiamo al server un comando di «shutdown» che ne provoca lo spegnimento ordinato.



Interfaccia PrezzoCarburante

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

public interface PrezzoCarburante extends Remote {
    Distributore getInfoDistributore(String name)
                                throws RemoteException;
    boolean setNewPrezzi(Distributore distributore)
                                throws RemoteException;
    void addDistributore(Distributore distributore)
                                throws RemoteException;
    List<String> getNomiDistributori() throws RemoteException;
    void shutdown() throws RemoteException;
}
```



PrezzoCarburanteClient

```
import java.util.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
public class PrezzoCarburanteClient {
    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            PrezzoCarburante stub = (PrezzoCarburante)
                                   registry.lookup("Prezzi");

            int numDistr = (int) (Math.random()*100);
            for(int tappa=0; tappa<numDistr ; tappa++){
                // ...
            }
            stub.shutdown();
        } catch (Exception e) {
            System.err.println("Client exc: " + e.toString());
        }
    }
}
```



PrezzoCarburanteImpl

```
public void shutdown() throws RemoteException {
    Registry registry = LocateRegistry.getRegistry();
    try {
        registry.unbind("Prezzi");
        UnicastRemoteObject.unexportObject(this, false);
    } catch (NotBoundException e) {}
    new Thread() {
        public void run() {
            System.out.print("Shutting down...");
            try {
                sleep(2000);
            } catch (InterruptedException e) {}
            System.exit(0);
        }
    }.start();
}
```

In questo modo si evita di spegnere il server prima che l'interazione col client sia completata. In questo modo il metodo shutdown viene eseguito completamente poi il server termina (con la exit).



CALLBACK



Comunicazioni iniziate dal server

- Nelle applicazioni viste finora il server aveva sempre un ruolo passivo
- Cioè si limita a dare risposte al client, mediante un'interazione sincrona.
- Ci sono casi in cui
 - ▶ Il server deve prendere l'iniziativa
 - ▶ L'interazione è asincrona, cioè il client non si blocca in attesa della risposta. Quest'ultima viene inviata dal server in un momento non predicibile.
 - ▶ In entrambi i casi il client deve essere continuamente in grado di ricevere messaggi dal server.

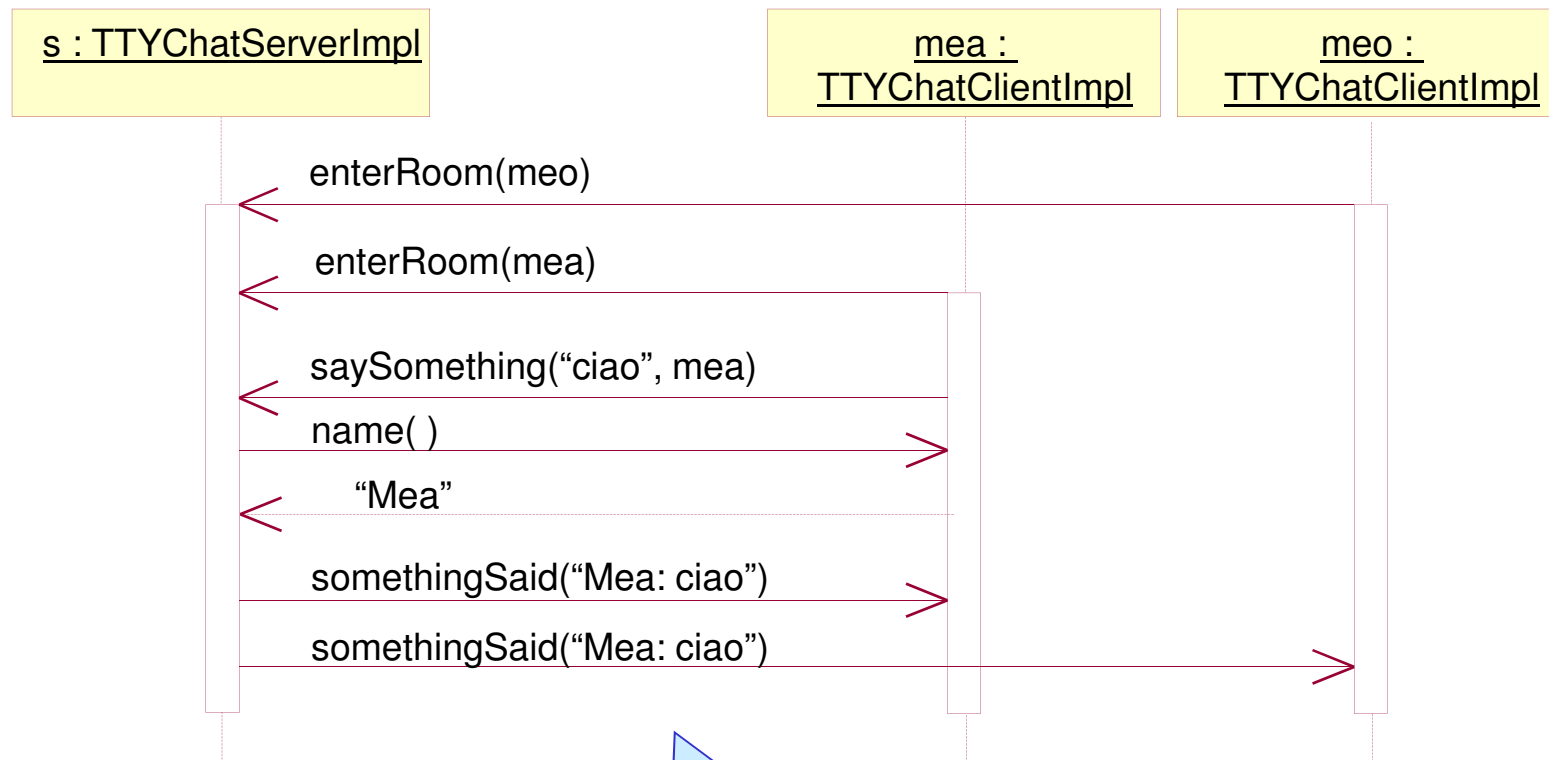


Applicazione: TTYchat

- Vogliamo sviluppare un sistema di chat
- I partecipanti (client) entrano in una stanza virtuale (gestita dal server).
- Ogni volta che un partecipante dice qualcosa (comunicando un messaggio al server), tutti i partecipanti che si trovano nella stanza ricevono lo stesso messaggio.
- Un client può uscire dalla stanza. Di conseguenza non riceverà più i messaggi scambiati nella stanza.



TTYchat



Il server chiama
metodi dei client!



La caratteristica distintiva di TTYchat

- In questa applicazione, il partecipante A manda un messaggio al server
 - ▶ Cosa possibile perché il client possiede un riferimento remoto all'oggetto server (reperito attraverso il registry)
- A questo punto, il server deve mandare un messaggio al partecipante B
 - ▶ Per far questo, **il server deve avere un riferimento remoto al partecipante B**



Callback Server-to-Client

- Nel paradigma distribuito object-oriented i ruoli di client e server non sono rigidamente fissati e sono validi solo nel contesto di una singola interazione (richiesta di servizio)
- In particolare è possibile realizzare una comunicazione a due vie tra due oggetti remoti
 - ▶ Si utilizza ancora la denominazione convenzionale di client e server per classificare gli oggetti sulla base del comportamento “prevalente”
- Se un client è a sua volta un oggetto remoto sarà possibile inviare il suo riferimento al server in modo che quest’ultimo chiami i metodi remoti del client
 - ▶ Il termine “**callback**” identifica proprio un’operazione remota invocata da un oggetto che prevalentemente si comporta da server (fornitore di servizi) verso un oggetto che prevalentemente ha un comportamento da client (fruitore di servizi) e che in precedenza ha passato al server il proprio riferimento



Interfaccia remota del server **TTYchat**

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface TTYchat extends Remote {  
    void enterRoom(TTYchatClient client)  
        throws RemoteException;  
    void saySomething(String something, TTYchatClient speaker)  
        throws RemoteException;  
}
```

Gli argomenti **TTYchatClient client** e **TTYchatClient speaker** saranno riferimenti remoti al client che fa la chiamata di metodo.



TTYchatImpl: implementazione del server

```
import java.util.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.lang.Thread;

public class TTYchatImpl extends UnicastRemoteObject
    implements TTYchat {
    private static final long serialVersionUID = 1L;
    List<TTYchatClient> occupants;
    public TTYchatImpl() throws RemoteException {
        occupants = new ArrayList<TTYchatClient>();
    }
    public synchronized void enterRoom(TTYchatClient c)
        throws RemoteException {
        occupants.add(c);
    }
}
```

referimento
remoto al client



TTYchatImpl: implementazione del server

```
public synchronized void saySomething(String s,  
                                     TTYchatClient cc) throws RemoteException {  
    String message = cc.name()+" : "+s;  
    System.out.println(Thread.currentThread() +  
                       " : Server: got " +message);  
    int numClients=occupants.size(), i=0;  
    while(i<numClients) {  
        TTYchatClient c=occupants.get(i);  
        try {  
            c.somethingSaid(message);  
            i++;  
        } catch (Exception x) {  
            System.out.println("Someone left");  
            occupants.remove(c);  
            numClients--;  
        }  
    }  
}
```

riferimento
remoto al client

Qui il server chiama un
metodo remoto del client



TTYchatImpl: implementazione del server

```
static public void main(String args[]) {  
    try {  
        TTYchatImpl obj = new TTYchatImpl();  
        Registry registro = LocateRegistry.createRegistry(1099);  
        registro.rebind("TTYCHAT", obj);  
        System.out.println("TTYChat Server bound in registry");  
    } catch (Exception e) {  
        System.out.println("TTYChatServer err: " +  
                           e.getMessage());  
    }  
}
```



TTYchatClient: interfaccia remota del client

- È necessario che anche il client abbia un'interfaccia remota perché sia possibile il callback

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface TTYchatClient extends Remote {  
    void somethingSaid(String something)  
                                   throws RemoteException;  
    String name() throws RemoteException;  
}
```




TTYchatClientImpl: implementazione del client

```
import java.io.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class TTYchatClientImpl extends UnicastRemoteObject
                                implements TTYchatClient {
    private static final long serialVersionUID = 1L;
    String my_name;
    public TTYchatClientImpl(String n)
                                throws RemoteException {
        my_name = n;
    }
}
```

Anche il client è
un'oggetto remoto!

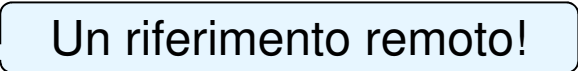
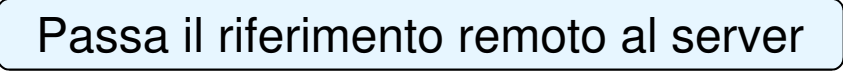


TTYchatClientImpl: implementazione del client

```
public String name() throws RemoteException {  
    return my_name;  
}  
public void somethingSaid(String who_what)  
                                throws RemoteException {  
    System.out.println(who_what);  
}
```











TTYchatClientImpl: implementazione del client

```
public static void main(String args[]) {  
    try {  
        BufferedReader input = new BufferedReader(  
            new InputStreamReader(System.in));  
        System.out.println("What is your name?");  
        TTYchatClientImpl me =    
            new TTYchatClientImpl(input.readLine());  
        Registry reg = LocateRegistry.getRegistry();  
        TTYchat server = (TTYchat) reg.lookup("TTYCHAT");  
        server.enterRoom(me);    
        System.out.println("You can now chat in the room");  
        while (true) {  
            server.saySomething(input.readLine(), me);  
        }  
    } catch (Exception e) {  
        System.out.println("Client err:"+e.getMessage());  
    }  
}
```



Dislocazione

▼		Client
		TTYchat.java
		TTYchatClient.java
		TTYchatClientImpl.java
▼		Server
		TTYchat.java
		TTYchatClient.java
		TTYchatImpl.java



Esecuzione

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Server$ java TTYchatImpl
TTYChat Server bound in registry
Thread[RMI TCP Connection(2)-127.0.0.1,5,RMI Runtime]:ServerGiulietta
' sei tu, Romeo?
Thread[RMI TCP Connection(4)-127.0.0.1,5,RMI Runtime]:ServerPerche' sei tu, Romeo?
Giulietta: Perche' sei tu, Romeo?
Romeo: Boh

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Client$ java TTYchatClientImpl
What is your name?
Romeo
You can now chat in the room
Giulietta: Perche' sei tu, Romeo?
Boh
Romeo: Boh
```



Modifiche

- Separiamo il server dall'implementazione dell'interfaccia
- Introduciamo **exitRoom** e la terminazione ordinata dei client



TTYchat: interfaccia remota del server

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

public interface TTYchat extends Remote {
    void enterRoom(TTYchatClient client)
                                throws RemoteException;
    void exitRoom(TTYchatClient client)
                                throws RemoteException;
    void saySomething(String something, TTYchatClient speaker)
                                throws RemoteException;
}
```



TTYchatImpl: implementazione del server

```
import java.util.*;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.lang.Thread;

public class TTYchatImpl extends UnicastRemoteObject
    implements TTYchat {
    private static final long serialVersionUID = 1L;
    List<TTYchatClient> occupants;
    public TTYchatImpl() throws RemoteException {
        occupants = new ArrayList<TTYchatClient>();
    }
}
```




TTYchatImpl: implementazione del server

```
public synchronized void enterRoom(TTYchatClient c
                                   throws RemoteException {
    occupants.add(c);
}
public synchronized void exitRoom(TTYchatClient c)
                                   throws RemoteException {
    occupants.remove(c);
}
```



TTYchatImpl: implementazione del server

```
public synchronized void saySomething(String s,
                                     TTYchatClient cc) throws RemoteException {
    String message = cc.name()+" : "+s;
    System.out.println(Thread.currentThread() +
                       ":Server: got " +message);
    int numClients=occupants.size(), i=0;
    while(i<numClients) {
        TTYchatClient c=occupants.get(i);
        try {
            c.somethingSaid(message);
            i++;
        } catch (Exception x) {
            System.out.println("Someone left");
            occupants.remove(c);
            numClients--;
        }
    }
}
```

NO main!



TTYchatServer: usa TTYChatImpl

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;

public class TTYchatServer {
    static public void main(String args[]) {
        try {
            TTYchatImpl obj = new TTYchatImpl();
            Registry registro =
                LocateRegistry.createRegistry(1099);
            registro.rebind("TTYCHAT", obj);
            System.out.println("TTYChat Server bound in
registry");
        } catch (Exception e) {
            System.out.println("TTYChatServer err: " +
                e.getMessage());
        }
    }
}
```

Il main è qui!
(identico a prima)



TTYchat: implementazione del client

```
// omissis ...
Registry reg = LocateRegistry.getRegistry();
TTYchat server = (TTYchat) reg.lookup("TTYCHAT");
server.enterRoom(me);
System.out.println("You can now chat in the room");
while (true) {
    String s = input.readLine();
    if(s.equals("<quit>")) {
        server.exitRoom(me);
        break;
    } else {
        server.saySomething(s, me);
    }
}
UnicastRemoteObject.unexportObject(me, false);
// ...
```