



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

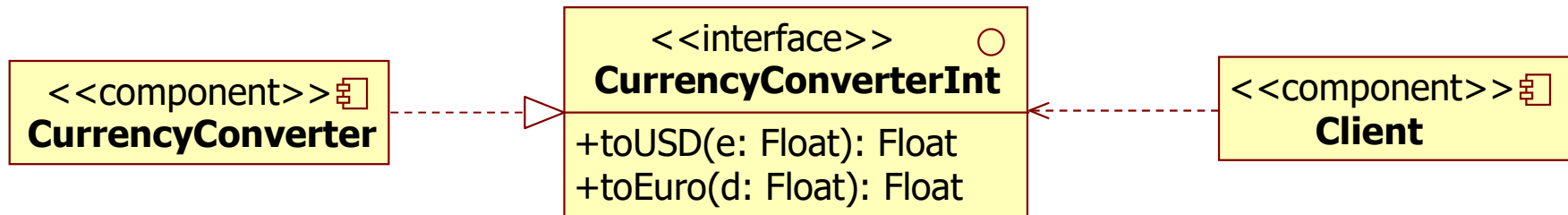
Programmazione Concorrente e Distribuita

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*0.95F;
    }
    public float toUsd(float eur) {
        return eur*1.06F;
    }
}
```

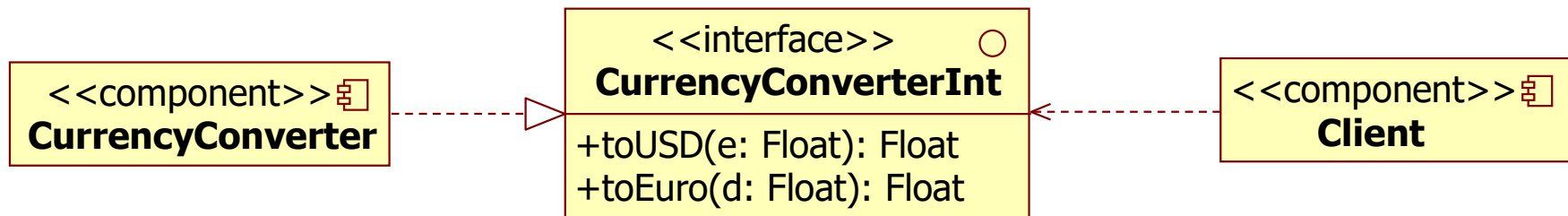


CurrencyConverterClient

```
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**

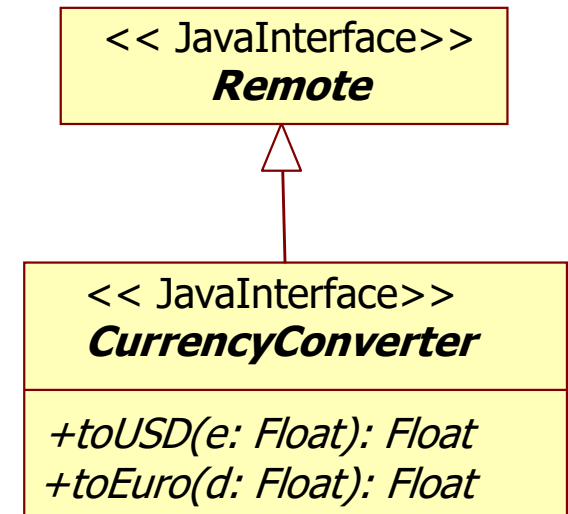
- L'interfaccia che definisce le funzionalità del Server remoto (cioè accessibile da altri oggetti non locali tramite il sistema RMI) devono ereditare da **`java.rmi.Remote`**
- I metodi definiti in tali interfacce devono dichiarare l'eccezione **`java.rmi.RemoteException`**



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*0.95F;  
    }  
    public float toUsd(float eur) throws RemoteException {  
        return eur*1.06F;  
    }  
}
```

Non estende
UnicastRemoteObject

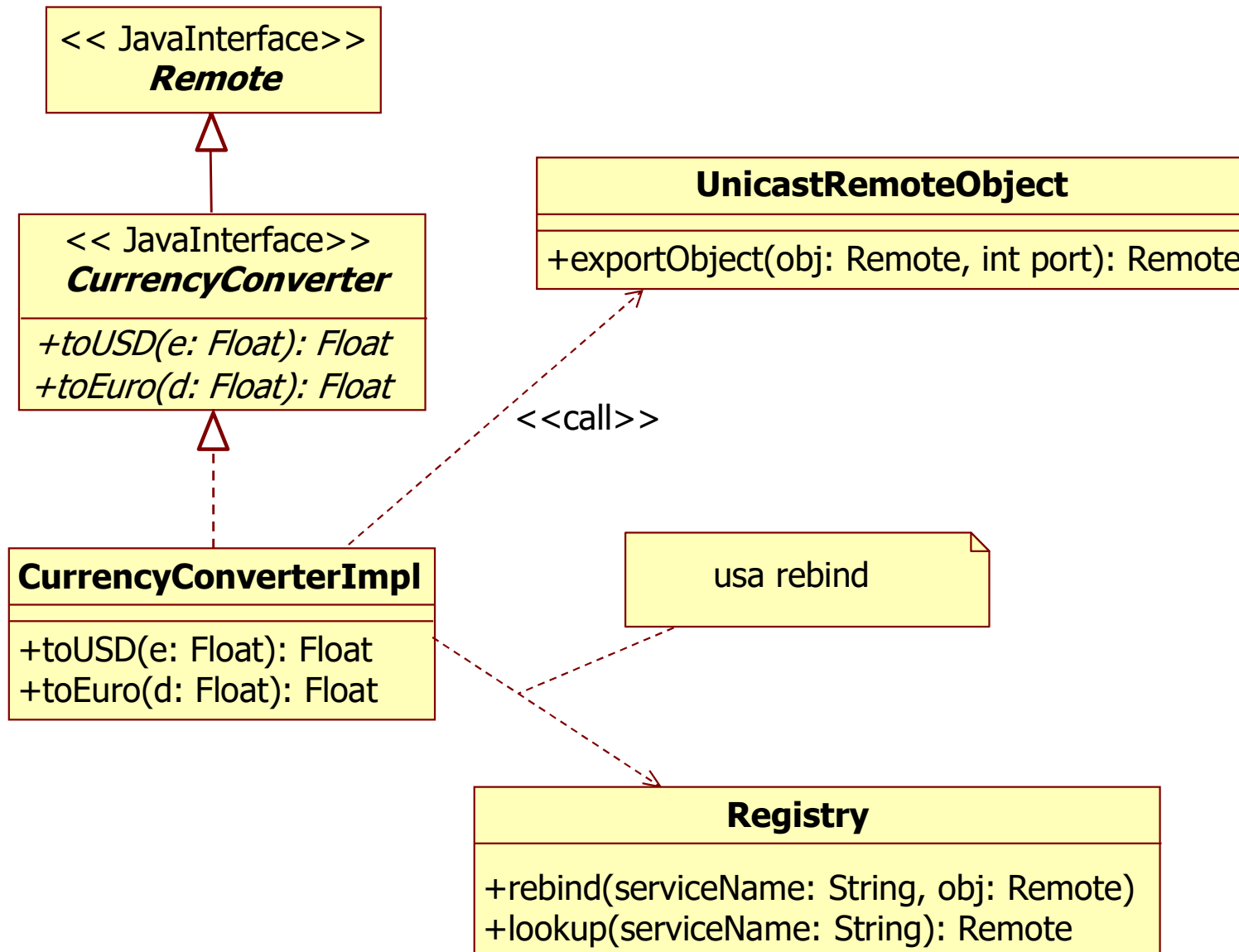


CurrencyConverterImpl

Crea un oggetto normale,
poi lo converte a
UnicastRemoteObject

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

CurrencyConverterImpl



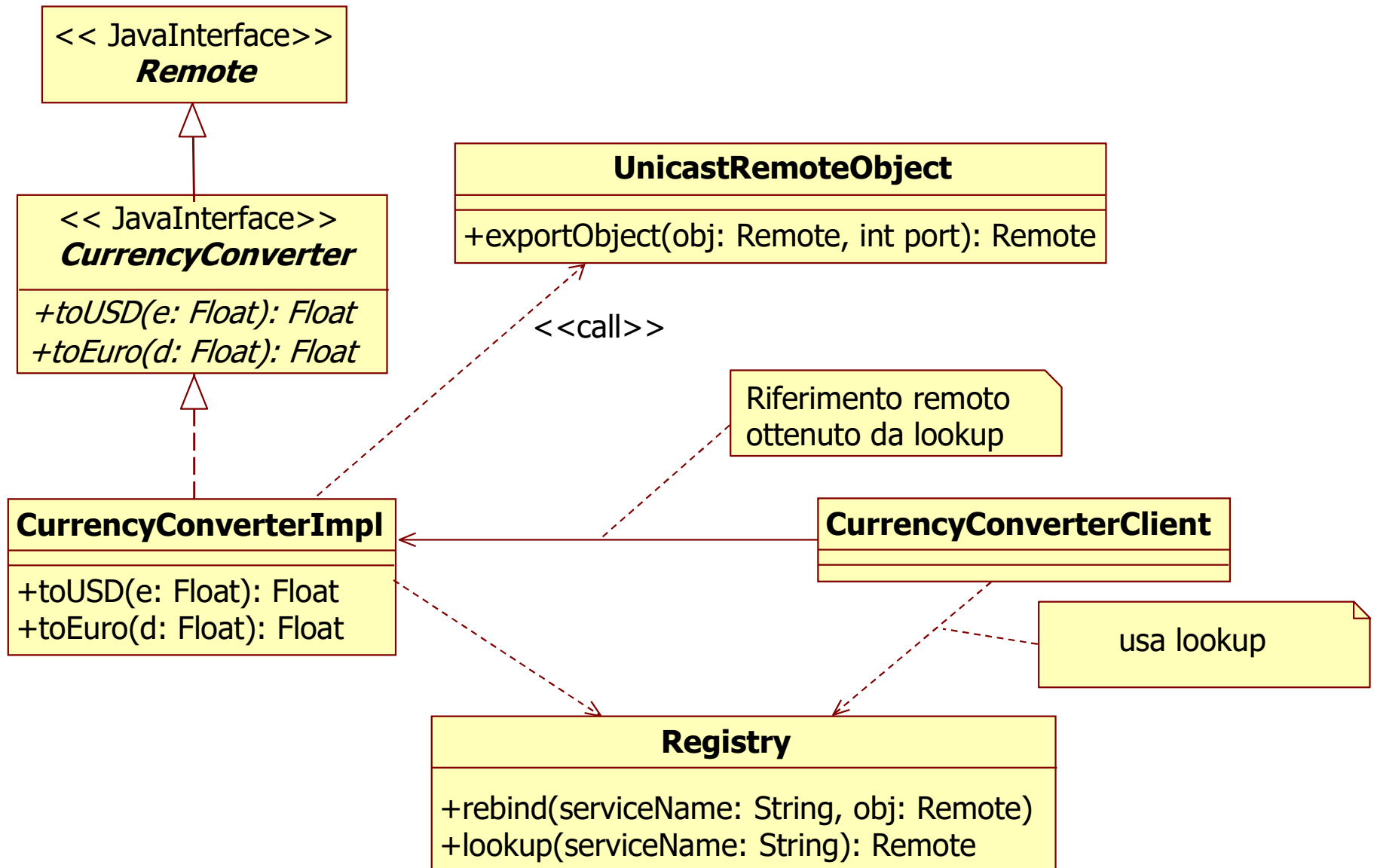


CurrencyConverterClient

```
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());
        }
    }
}
```

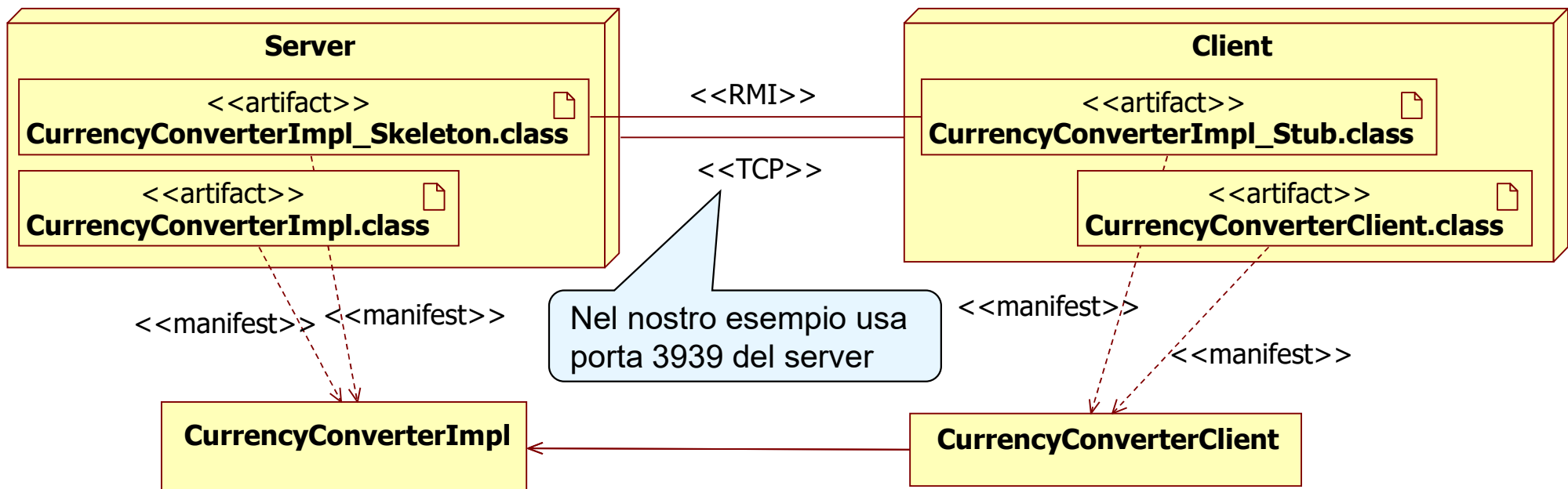


CurrencyConverterImpl



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*0.95F;
    }
    private float toUsd(float eur) {
        return eur*1.06F;
    }
}
```



Class CurrencyConverter

```
public Conversion compute(Conversion conv) {  
    if (conv.givenCurrency.equals("USD") &&  
        conv.targetCurrency.equals("EUR")) {  
        conv.setTargetAmount(toEur(conv.getGivenAmount()));  
        return conv;  
    }  
    if (conv.givenCurrency.equals("EUR") &&  
        conv.targetCurrency.equals("USD")) {  
        conv.setTargetAmount(toUsd(conv.getGivenAmount()));  
        return conv;  
    }  
    return null;  
}
```



Class CurrencyConverter

```
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

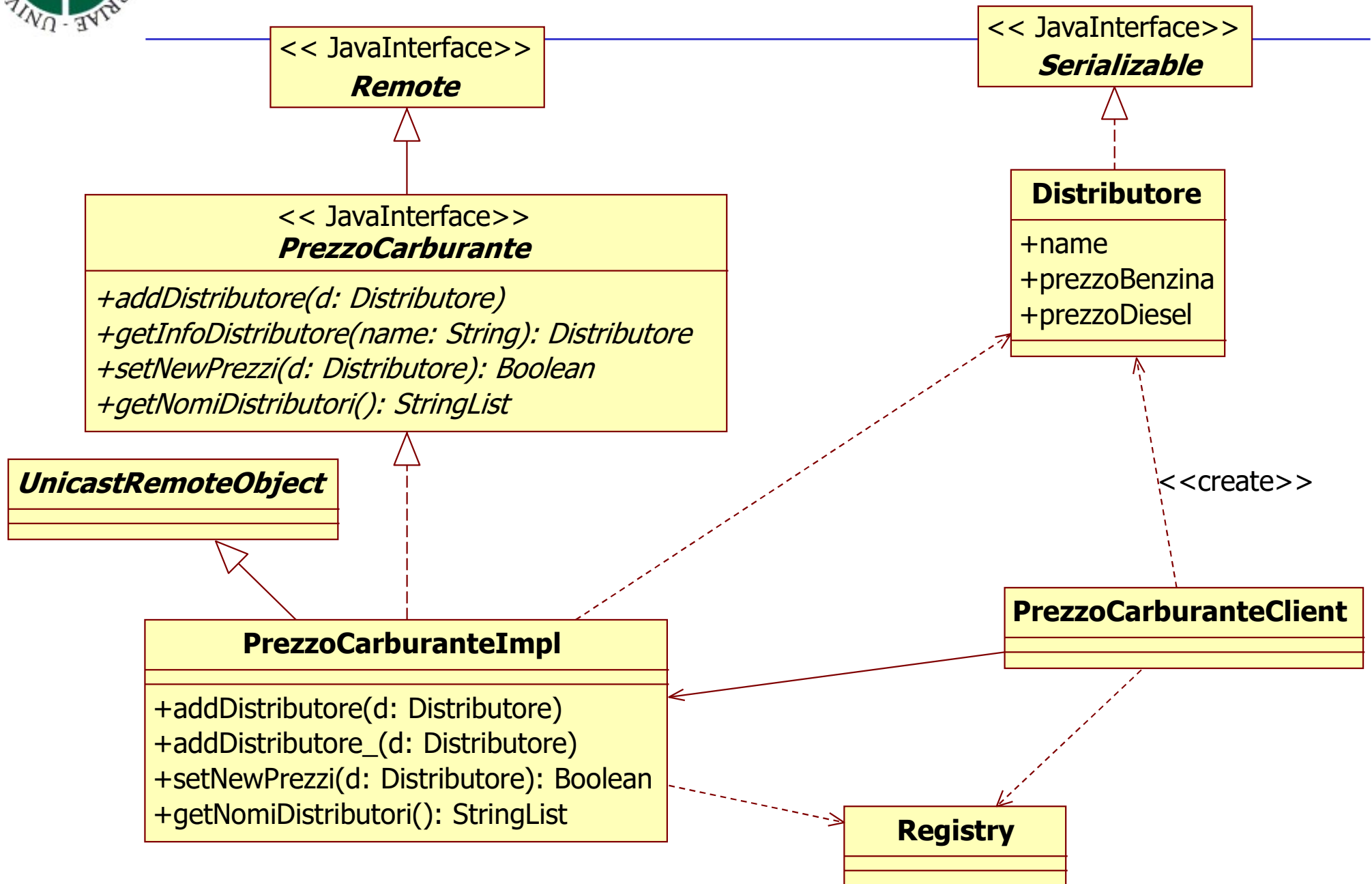
```
void exec() throws RemoteException, NotBoundException {
    Conversion conv;
    Random rnd=new Random();  int times=2+rnd.nextInt(10);
    String fromCurrency, toCurrency;
    Registry registro = LocateRegistry.getRegistry(1099);
    CurrencyConverterInterface stub =
        (CurrencyConverterInterface)
            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 {
    void addDistributore(Distributore distributore)
                                throws RemoteException;
    List<String> getNomiDistributori() throws RemoteException;
    Distributore getInfoDistributore(String name)
                                throws RemoteException;
    boolean setNewPrezzi(Distributore distributore)
                                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 riferarmi 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.



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 {  
    System.out.print("Shutting down...");  
    Registry registry = LocateRegistry.getRegistry();  
    try {  
        registry.unbind("PrezziCarburante");  
        UnicastRemoteObject.unexportObject(this, true);  
    } catch (NotBoundException e) {}  
}
```

L'oggetto cessa di essere un remote object.
Il main termina (immediatamente)



UnicastRemoteObject.unexportObject

```
public static boolean unexportObject(Remote obj,  
                                     boolean force)  
    throws NoSuchObjectException
```

- Removes the remote object, `obj`, from the RMI runtime. If successful, the object can no longer accept incoming RMI calls. If the **force** parameter is true, the object is forcibly unexported even if there are pending calls to the remote object or the remote object still has calls in progress. If the **force** parameter is false, the object is only unexported if there are no pending or in progress calls to the object.
- Returns:
true if operation is successful, false otherwise
- Throws:
NoSuchObjectException - if the remote object is not currently exported



PrezzoCarburanteImpl

```
public void shutdown() throws RemoteException {
    Registry registry = LocateRegistry.getRegistry();
    try {
        registry.unbind("PrezziCarburante");
        UnicastRemoteObject.unexportObject(this, false);
    } catch (NotBoundException e) {
        System.out.println("Shutdown failed");
    }
    new Thread() {
        public void run() {
            try { sleep(2000);
            } catch (InterruptedException e) { }
            System.exit(0);
        }
    }.start();
    System.out.println("Shutdown request completed");
}
```

Unexport fallisce perché il server è impegnato a eseguire il metodo shutdown richiesto dal client.

Il metodo shutdown termina mentre il thread dorme. Quando il thread si sveglia l'interazione col client è finita e quindi fa exit.



Shutdown: terza via

- Vogliamo fare shutdown
 - ▶ Senza interrompere le comunicazioni col client
 - ▶ Senza usare exit
- Come?
 - ▶ Terminiamo l'esecuzione di shutdown()
 - ▶ Poi facciamo unexport.
 - Delegando un thread che agisca con un po' di ritardo.



Shutdown thread

```
import java.rmi.NoSuchObjectException;
import java.rmi.Remote;
import java.rmi.server.UnicastRemoteObject;

public class Unexporter extends Thread {
    Remote toBeUnexported;
    Unexporter(Remote obj) {
        toBeUnexported=obj;
    }
    public void run() {
        try { Thread.sleep(2000); } catch (InterruptedException e) {}
        try {
            if(UnicastRemoteObject.unexportObject(toBeUnexported, false)) {
                System.out.println("Unexporting succeeded.");
            } else {
                System.out.println("Unexporting failed.");
            }
        } catch (NoSuchObjectException e) {
            System.err.println("Unexporting failed by exception.");
        }
    }
}
```

Il ritardo serve ad assicurarci che shutdown sia terminato quando facciamo unexport



Shutdown thread

```
public void shutdown() throws RemoteException {  
    System.out.println("Shutdown request received");  
    new Unexporter(this).start();  
    System.out.println("Shutdown request completed");  
}
```

CALLBACK



Comunicazioni iniziate dal server

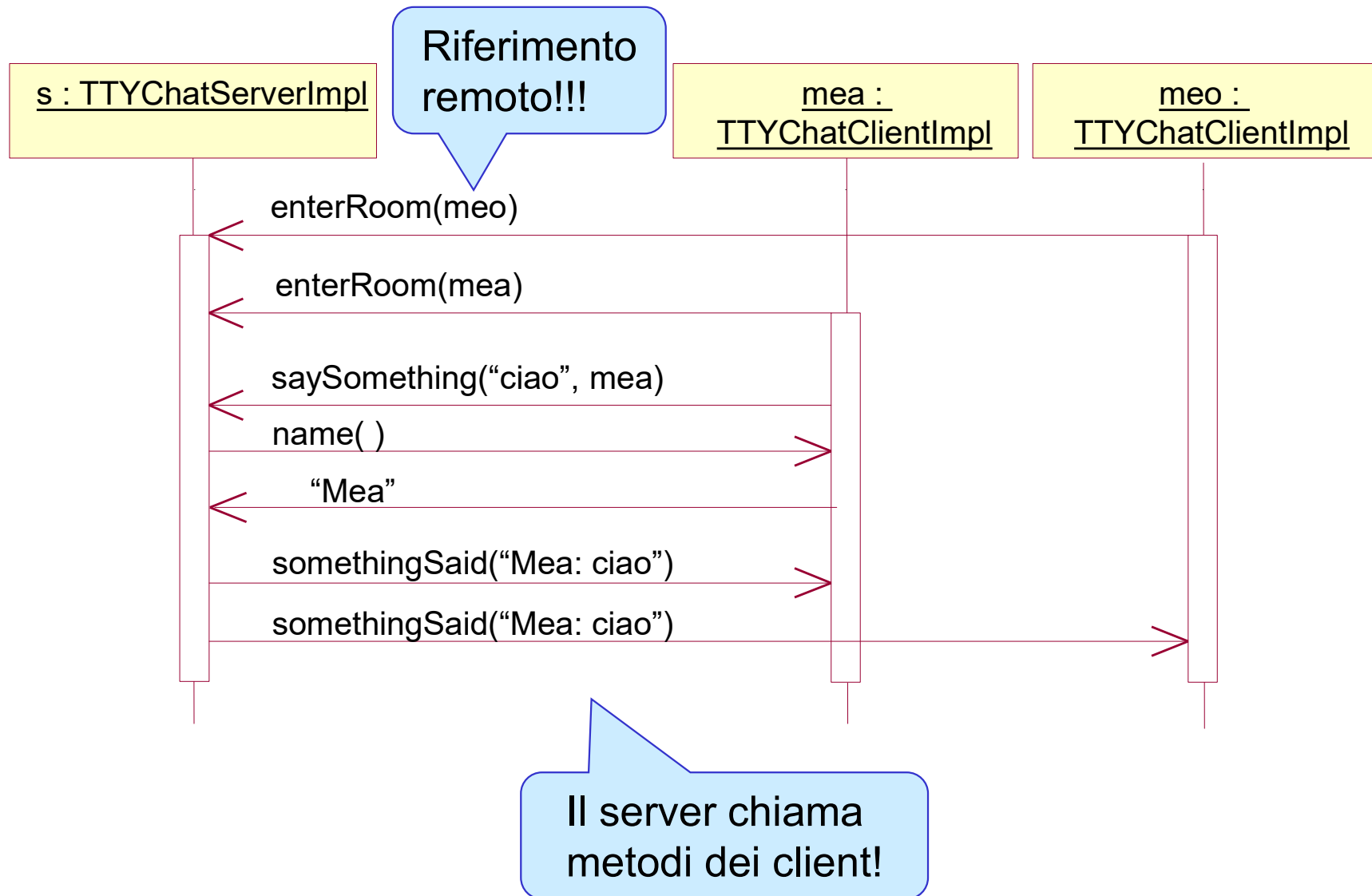
- Nelle applicazioni viste finora il server aveva sempre un ruolo passivo
- Cioè si limita a dare risposte al client, mediante un'interazione sincrona.
- Ci sono casi in cui
 - ▶ Il server deve prendere l'iniziativa
 - ▶ L'interazione è asincrona, cioè il client non si blocca in attesa della risposta. 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, tutti i partecipanti che si trovano nella stanza ricevono lo stesso messaggio.
 - ▶ Il partecipante comunica un messaggio al server
 - ▶ Il server comunica il messaggio a tutti gli altri partecipanti nella stanza
- Un client può uscire dalla stanza. Di conseguenza non riceverà più i messaggi scambiati nella stanza.

TTYchat





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

- 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** sono referimenti 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 {
        super();
        occupants = new ArrayList<TTYchatClient>();
    }
    public synchronized void enterRoom(TTYchatClient c)
        throws RemoteException {
        occupants.add(c);
    }
}
```

riferimento
remoto al client



TTYchatImpl: implementazione del server

```
public synchronized void saySomething(String s,  
TTYchatClient cc) throws RemoteException {
```

Rif. remoto al client

```
String message = cc.name()+" : "+s;  
System.out.println(Thread.currentThread() +  
":Server: got " +message);
```

Rif. remoto al client

Qui il server chiama un metodo remoto del client

```
for(TTYchatClient ttyc: occupants) {  
    try {  
        ttyc.somethingSaid(message);  
    } catch (Exception x) {  
        System.out.println("Someone left");  
        occupants.remove(ttyc);  
    }  
}
```

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 {
        super();
        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



RMI: esempi



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);
    for(TTYchatClient ttyc: occupants) {
        try {
            if(!ttyc.equals(cc)) {
                ttyc.somethingSaid(message);
            }
        } catch (Exception x) {
            System.out.println("Someone left");
            occupants.remove(ttyc);
        }
    }
}
```

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 reg.");
        } 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);
// ...
```

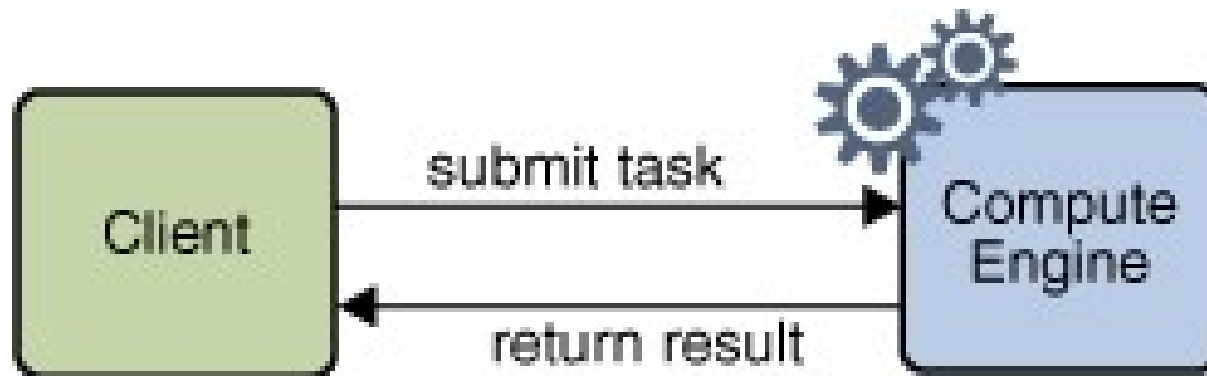


MOTORE DI CALCOLO

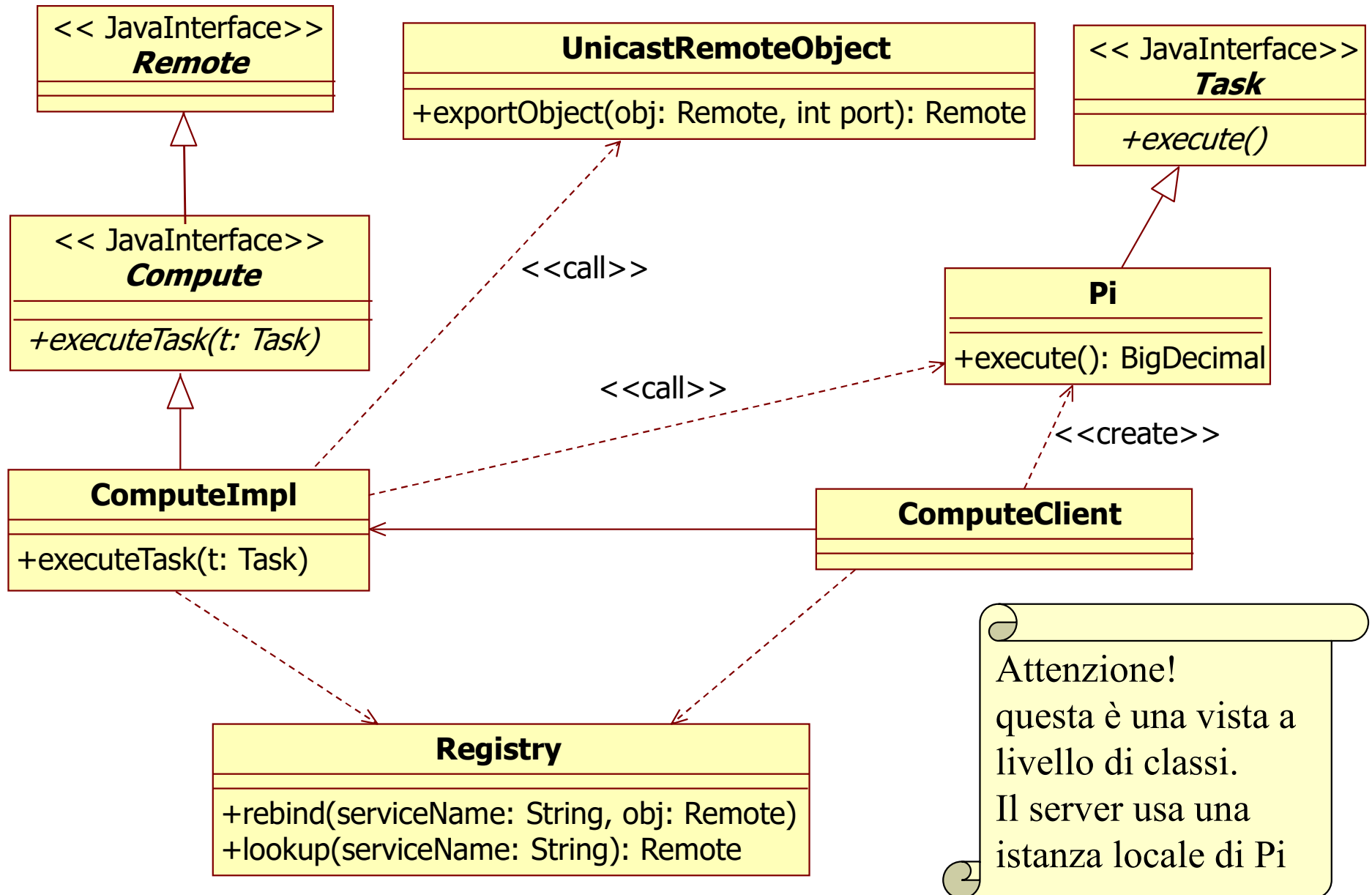
Tagliato perche' richiede trasmissione di classi

Esempio: Motore di Calcolo

- Consente di sottomettere dei task generici al Motore di Calcolo:
 - ▶ il lavoro pesante viene fatto dal server
 - ▶ i risultati dell'esecuzione dei task vengono restituiti al client.



Compute Engine





Interfacce: Compute e Task

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Compute extends Remote {  
    <T> T executeTask(Task<T> t) throws RemoteException;  
}  
  
public interface Task<T> {  
    T execute();  
}
```



ComputeImpl

```
import java.rmi.registry LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
public class ComputeImpl implements Compute {
    public ComputeImpl() {}
    public <T> T executeTask(Task<T> t) { return t.execute(); }
    public static void main(String[] args) {
        try {
            ComputeImpl engine = new ComputeImpl();
            Compute stub = (Compute)
                UnicastRemoteObject.exportObject(engine, 3939);
            Registry registry = LocateRegistry.createRegistry(1099);
            registry.rebind("Compute", stub);
            System.out.println("ComputeEngine bound");
        } catch (Exception e) {
            System.err.println("ComputeEngine exception:");
            e.printStackTrace();
        }
    }
}
```

Per il server
è locale



Task da eseguire

```
import java.io.Serializable;
import java.math.BigDecimal;
public class Pi implements Task<BigDecimal>, Serializable {
    private static final long serialVersionUID = 227L;
    private static final BigDecimal FOUR = BigDecimal.valueOf(4);
    private static final int roundingMode =
        BigDecimal.ROUND_HALF_EVEN; // rounding mode to use
    private final int digits; // decimal digits
    // task to calculate pi to the specified precision.
    public Pi(int digits) {
        this.digits = digits;
    }
    public BigDecimal execute() { // Calculate pi
        return computePi(digits);
    }
}
```




Task da eseguire

```
/**
 * Compute the value of pi to the specified number of
 * digits after the decimal point. The value is
 * computed using Machin's formula:
 *      
$$\pi/4 = 4 \cdot \arctan(1/5) - \arctan(1/239)$$

 * and a power series expansion of  $\arctan(x)$  to
 * sufficient precision.
 */
public static BigDecimal computePi(int digits) {
    int scale = digits + 5;
    BigDecimal arctan1_5 = arctan(5, scale);
    BigDecimal arctan1_239 = arctan(239, scale);
    BigDecimal pi = arctan1_5.multiply(FOUR).subtract(
        arctan1_239.multiply(FOUR));
    return pi.setScale(digits, BigDecimal.ROUND_HALF_UP);
}
```



Task da eseguire

```
public static BigDecimal arctan(int inverseX, int scale) {
    BigDecimal result, numer, term;
    BigDecimal invX = BigDecimal.valueOf(inverseX);
    BigDecimal invX2 = BigDecimal.valueOf(inverseX * inverseX);
    numer = BigDecimal.ONE.divide(invX, scale, roundingMode);
    result = numer;
    int i = 1;
    do {
        numer = numer.divide(invX2, scale, roundingMode);
        int denom = 2 * i + 1;
        term = numer.divide(BigDecimal.valueOf(denom), scale,
                                roundingMode);
        if ((i % 2) != 0)
            result = result.subtract(term);
        else
            result = result.add(term);
        i++;
    } while (term.compareTo(BigDecimal.ZERO) != 0);
    return result;
}}
```



ComputeClient

```
import java.math.BigDecimal;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class ComputeClient {
    public static void main(String args[]) {
        String host = (args.length < 1) ? null : args[0];
        int digits = (args.length < 2) ? 100 :
                                Integer.parseInt(args[1]);
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Compute comp = (Compute) registry.lookup("Compute");
            Pi task = new Pi(digits);
            BigDecimal pi = comp.executeTask(task);
            System.out.println(pi);
        } catch (Exception e) {
            System.err.println("ComputePi exception:");
            e.printStackTrace();
        }
    }
}
```