Lezione n.9
LPR- INFORMATICA APPLICATA
REMOTE METHOD INVOCATION
CALLBACKS

04/05/2009

Laura Ricci

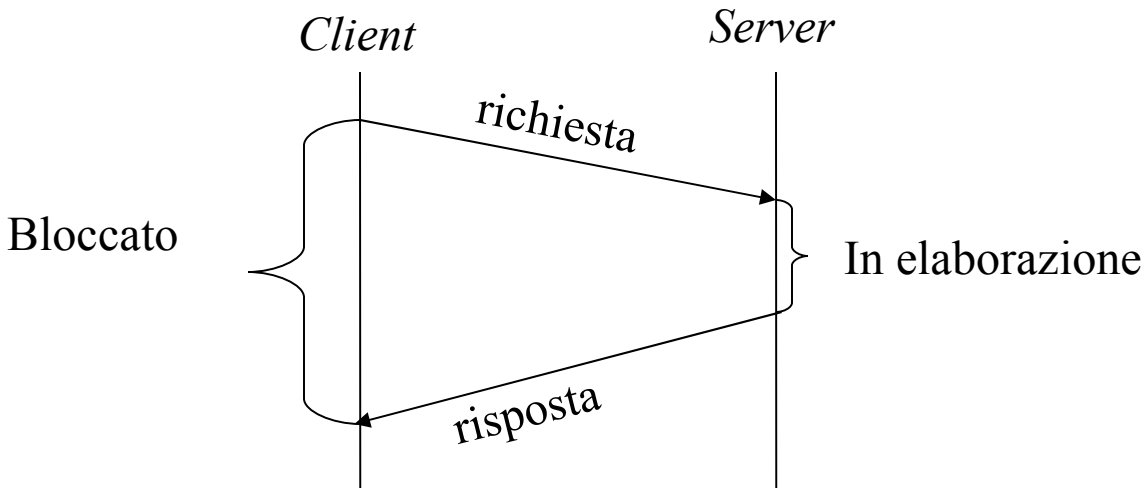
RIASSUNTO DELLA LEZIONE

- paradigma di interazione domanda/risposta
- remote procedure call
- RMI (Remote Method Invocation): API JAVA
- Esercizio
- Il meccansimo delle Callbacks

PARADIGMA DI INTERAZIONE A DOMANDA/RISPOSTA

Paradigma di interazione basato su richiesta/risposta

- il client invia ad un server un messaggio di richiesta
- il server risponde con un messaggio di risposta
- il client rimane bloccato (sospende la propria esecuzione) finchè non riceve la risposta dal server

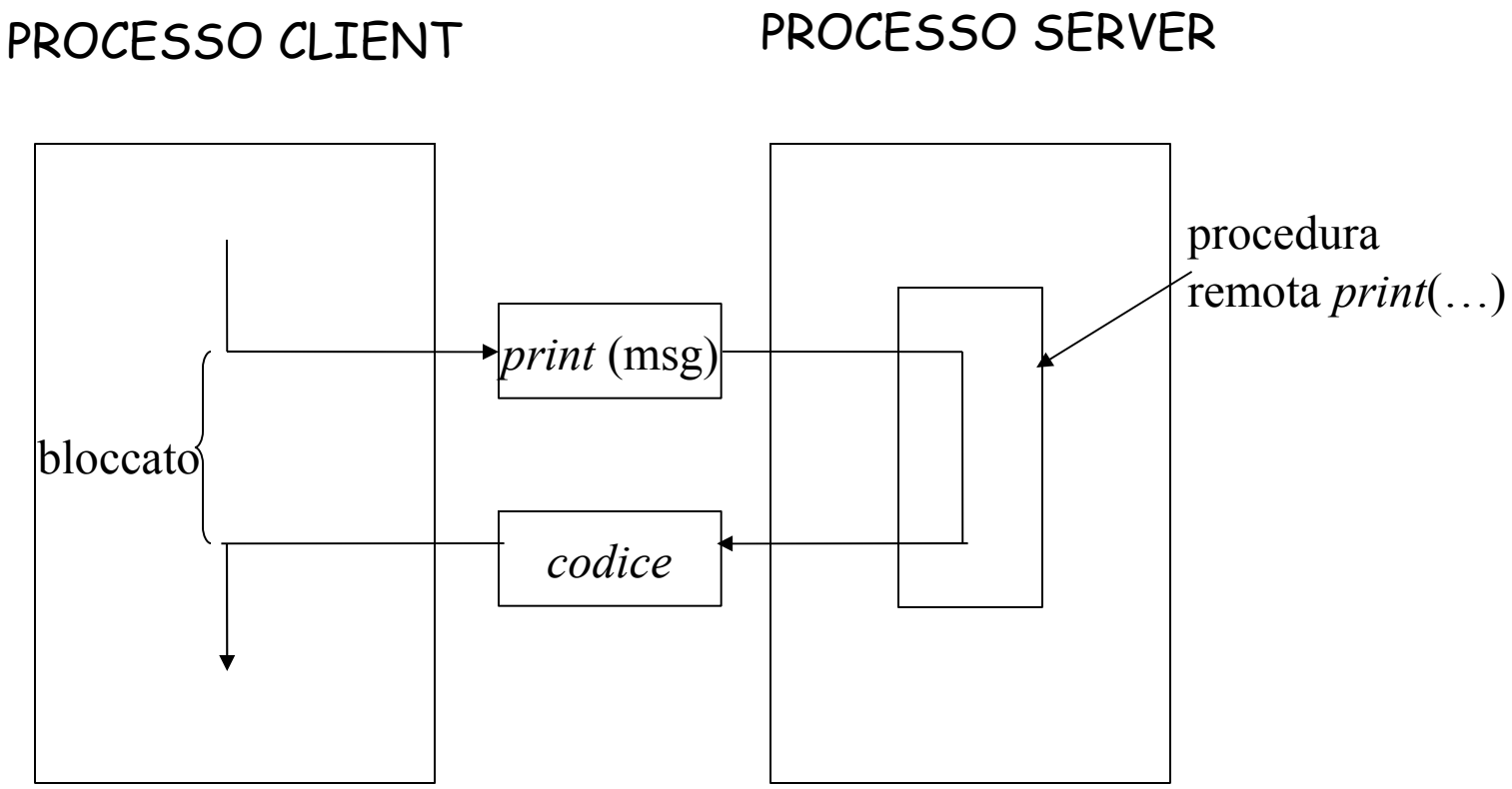


REMOTE PROCEDURE CALL

Esempio interazione domanda/risposta:

- un client richiede ad un server la stampa di un messaggio. Il server restituisce al client un codice che indica l'esito della operazione. Il client attende l'esito dell'operazione
- richiesta del client al server = invocazione di una procedura definita sul server
- Il client invoca una procedura remota RPC (Remote Procedure Call)
- I meccanismi utilizzati dal client sono gli stessi utilizzati per una normale invocazione di procedura, ma …
  - l'invocazione di procedura avviene sull' host su cui è in esecuzione il client
  - la procedura viene eseguita sull' host su cui è in esecuzione il server
  - i parametri della procedura vengono inviati automaticamente sulla rete dal supporto all'RPC

REMOTE PROCEDURE CALL



Esempio: richiesta stampa di messaggio e restituzione esito operazione

REMOTE METHOD INVOCATION

implementazioni RPC

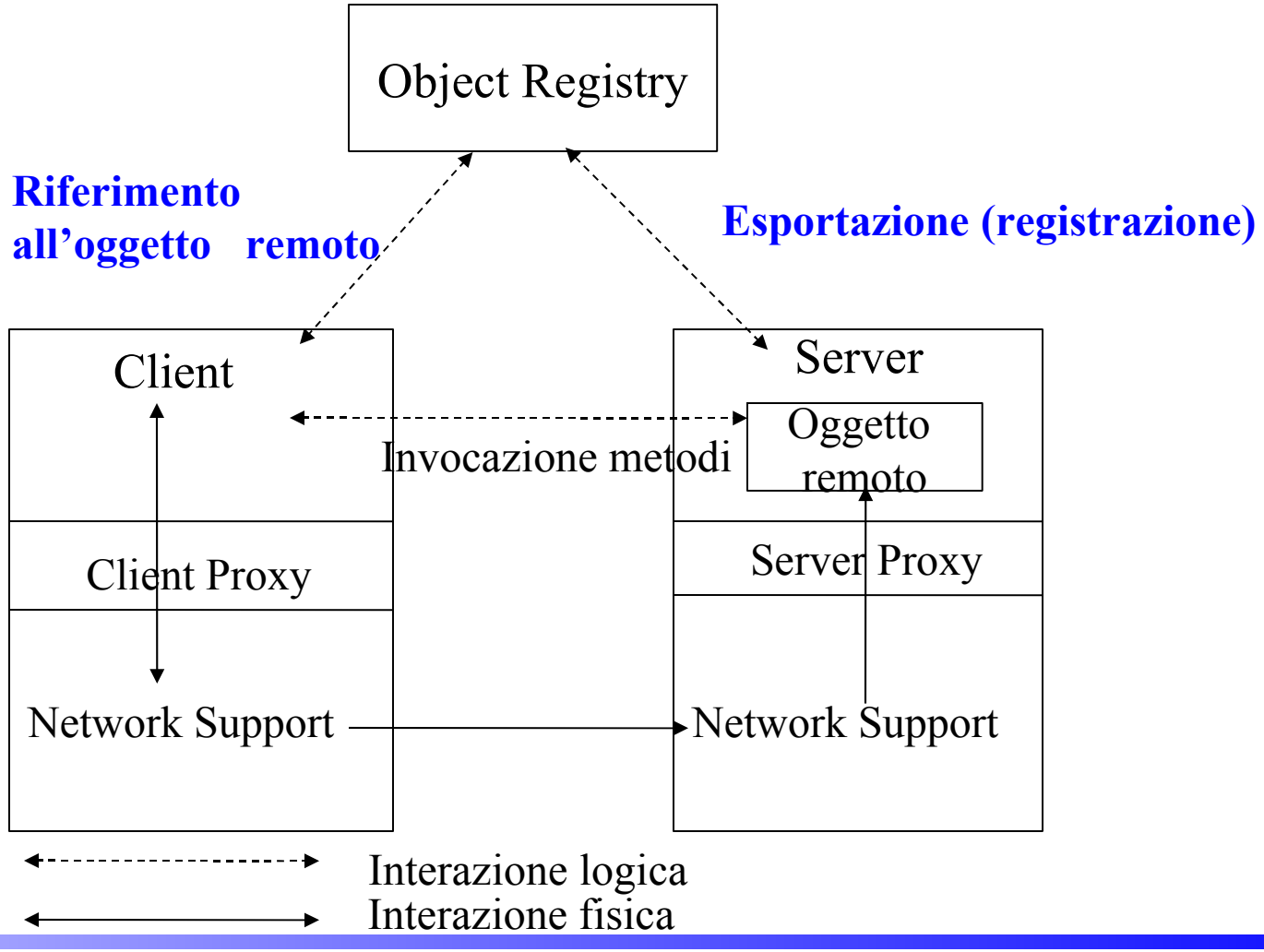
- Open Network Computing Remote Procedure Call (Sun)
- Open Group Distributed Computing Environment (DCE)
- …

Evoluzione di RPC = paradigma di interazione basato su oggetti distribuiti

remote method invocation (RMI): evoluzione del meccanismo diinvocazione di procedura remota al caso di oggetti remoti

JAVA RMI: JAVA API per la programmazione distribuita ad oggetti

OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE



OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE

Il server che definisce l'oggetto remoto:

- definisce un oggetto distribuito = un oggetto i cui metodi possono essere invocati da parte di processi in esecuzione su hosts remoti
- esporta (pubblica) l'oggetto: crea un mapping

  nome simbolico oggetto/ riferimento all'oggetto

  e lo pubblica mediante un servizio di tipo registry
  (simile ad un DNS per oggetti distribuiti)

OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE

Quando il client vuole accedere all'oggetto remoto

- ricerca un riferimento all'oggetto remoto mediante i servizi offerti dal registry
- invoca i metodi definiti dall'oggeto remoto (remore method invocation).
- invocazione dei metodi di un oggetto remoto
  - a livello logico: identica all' invocazione di un metodo locale
  - a livello di supporto: è gestita da un client proxy che provvede a trasformare i parametri della chiamata remota in dati da spedire sulla rete.

  Il network support provvede quindi all'invio vero e proprio dei dati sulla rete

OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE

Quando il server che gestisce l'oggetto remoto riceve un'invocazione per quell'oggetto

- il network support passa i dati ricevuti al server proxy
- il server proxy trasforma i dati ricevuti dal network support
  - in una invocazione ad un metodo locale
  - occorre trasformare i dati ricevuti dal network support nei parametri del metodo invocato

RMI: API JAVA

I metodi definiti dall'oggetto remoto ed i rispettivi parametri (le segnature dei metodi) devono essere noti:

- al client, che richiede un insieme di servizi mediante l'invocazione di tali metodi
- al server, che deve fornire un'implementazione di tali metodi

Il client non è interessato all' implementazione di tali metodi

In JAVA:

- definizione di un' interfaccia che contiene le segnature di un insieme di metodi, ma non il loro codice

- definizione di una classe che implementi l'interfaccia: contiene il codice dei metodi elencati nella interfaccia

INTERFACCE JAVA: RIPASSO

Una interfaccia JAVA può contenere solamente:

- metodi astratti e costanti: niente costruttori, niente variabili, solo l'intestazione dei metodi.
- i metodi sono tutti astratti, anche se manca la parola chiave abstract: infatti al posto del corpo c'è solo un punto e virgola;
- si può dichiarare che una classe implementa (implements) una data interfaccia: deve allora fornire un'implementazione per tutti i suoi metodi.
- una classe può implementare più di una interfaccia: la relazione implements non deve rispettare la regola dell'ereditarietà singola

INTERFACCE: RIPASSO

```
interface Figure {
/* gli oggetti delle classi che realizzano questa
   interfaccia sono caratterizzati da  un tipo  e da un'area
*/
    int PROVA = 5;
    double area(  );
    String tipo(   );
}
```

INTERFACCE: RIASSUNTO

```
class RadiceFigure {
   protected double dim1;
   protected double dim2;
    RadiceFigure(double a, double b)
           { dim1 = a; dim2 = b; }   }
class Rectangle extends RadiceFigure implements Figure{
      Rectangle(double a, double b) { super(a, b); }
  // definisce area() di Figure
  public double area( ) {   return dim1 * dim2;   }
   // definisce tipo() di Figure
  public String tipo( ) { return "Rectangle";} }
```

INTERFACCE: RIASSUNTO

```
class Triangle extends RadiceFigure implements Figure {
        Triangle(double a, double b) {super(a, b);}
      // definisce area( ) di Figure
      public double area( ) {  return dim1 * dim2 / 2;  }
      // definisce tipo( ) di Figure
       public String tipo( ) {return "Triangle "; }
   }
```

INTERFACCE: RIPASSO

```
class prova {
public static void main (String args[ ]) {
    // Figure f = new Figure(10, 10); // questo è illegale
    Figure figref; //  OK non creo l' oggetto
    figref = new Rectangle(9, 5);
    System.out.println(figref.tipo( ) + figref.area( ));
    figref = new Triangle(10, 8);
    System.out.println(figref.tipo( ) + figref.area( ));
    // figref.dim1 = 0 ; //anche questo e` illegale
  }
```

JAVA: REMOTE INTERFACE

Esempio 2: un Host è connesso ad una stazione metereologica che rileva temperatura, umidità,….mediante diversi strumenti di rilevazione. Sull'host è in esecuzione un server che fornisce queste informazioni agli utenti interessati.

```
import java.rmi.*;
public interface weather extends Remote;

public double getTemperature ( ) throws RemoteException;
public double getHumidity    ( ) throws RemoteException;
public double getWindSpeed   ( ) throws RemoteException;
```

JAVA RMI

I metodi esportati di un oggetto remoto devono essere definiti mediante un'interfaccia remota

- estende l'interfaccia Remote
- i metodi definiti possono sollevare eccezioni remote. Una eccezione remota indica un generico fallimento nella comunicazione remota dei parametri e dei risultati al/da il metodo remoto

Esempio 1: Definiamo un oggetto remoto che fornisce un servizio di echo, cioè ricevuto un valore come parametro, restituisce al chiamante lo stesso valore

- Passo 1. Definire una interfaccia che includa le segnature dei metodi che possono essere invocati da remoto
- Passo 2. Definire una classe che implementi l'interfaccia. Questa classe include l'implementazione di tutti i metodi che possono essere invocati da remoto

JAVA RMI

Passo 1. Definizione dell'interfaccia

```
import java.rmi.*;
public interface EchoInterface extends Remote {
    String getEcho (String Echo) throws RemoteException;}
```

Remote è una interfaccia che non definisce alcun metodo. Il solo scopo è quello di identificare gli oggetti che possono essere utilizzati in remoto

Passo 2. Implementazione dell'interfaccia

```
public class Server implements EchoInterface {
public Server(   ) {   }
    public String getEcho (String echo) {return echo ;} }
```

- definizione di una classe che implementa l'interfaccia remota
- la classe può definire ulteriori metodi pubblici, ma solamente quelli definiti nella interfaccia remota possono essere invocati da un altro host

JAVA RMI

Passo 3. Definire ed esportare un'istanza dell'oggetto remoto

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
public class ServerActivate {
public static void main(String args[ ]) {
try {Server obj = new Server(    );
     EchoInterface stub = (EchoInterface)
                    UnicastRemoteObject.exportObject(obj, 0);
      Registry registry = LocateRegistry.getRegistry (  );
      registry.bind  ("Echo", stub);
      System.err.println("Server ready");
    } catch (Exception e) {
    System.err.println("Server exception: "+e.toString());}}}
```

CREAZIONE E PUBBLICAZIONE DELL'OGGETTO REMOTO

Il server

- crea un'istanza dell'oggetto (**new**)
- invoca il metodo statico `UnicastRemoteObject.exportObject(obj, 0)` che
  - esporta l'oggetto remoto obj creato in modo che le invocazioni ai suoi metodi possano essere ricevute sulla porta specificata. La porta 0 specifica l'uso di una porta anonima
  - restituisce lo stub dell'oggetto remoto.
  - stub = contiene uno scheletro per ogni metodo definito nell'interfaccia (con le solite segnature), ma trasforma l'invocazione di un metodo in una richiesta ad un host ed ad una porta remoto
- dopo aver eseguito il metodo, un server RMI aspetta invocazioni di metodi remoti su un serverscket legato alla porta specificata
- lo stub generato (da passare al client) contiene indirizzo IP e porta su cui è attivo il server RMI

CREAZIONE DELLO STUB

- Il server deve generare lo stub e renderlo disponibile al client
- NOTA BENE: se si utilizza una versione di JAVA antecedente alla 5, lo stub deve essere
  - generato mediante rmic (rmi compiler) e passato esplicitamente al client
- Nelle ultime versioni di JAVA è invece possibile utilizzare la exportObject per generare lo stub
- Per invocare i metodi dell'oggetto remoto, il client deve avere a disposizione lo stub dell'oggetto
- JAVA mette a disposizione del programmatore un semplice name server (registry) che consente
  - Al server di registrare lo stub con un nome simbolico
  - Al client di reperire lo stub tramite il suo nome simbolico

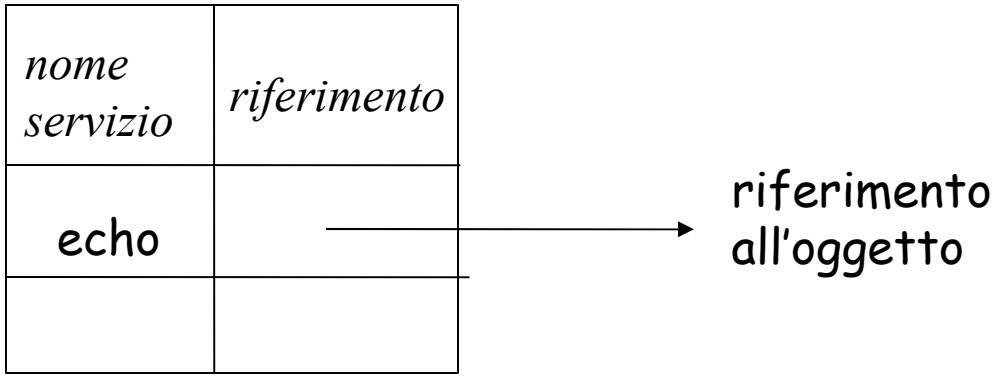
JAVA : ESPORTAZIONE DELLO STUB

Il Server

- per rendere disponibile lo stub creato agli eventuali clients, inserisce un riferimento allo stub creato nel registry locale (che deve essere attivo su local host sulla porta di default 1099)

```
Registry registry = LocateRegistry.getRegistry( );
registry.bind  ("Echo", stub);
```

- Registry = simile ad un DNS per oggetti remoti, contiene legami tra il nome simbolico dell'oggetto remoto ed il riferimento all'oggetto



JAVA: IL REGISTRY

- la classe LocateRegistry contiene metodi per la gestione dei registry
- La getRegistry( ) restituisce un riferimento ad un registry allocato sull'host locale e sulla porta di default 1099
- Si può anche specificare il nome di un host e/o una porta per individuare il servizio di registry su uno specifico host e/o porta
- nel caso più semplice si utilizza un registry locale, attivato sullo stesso host su cui è in esecuzione il server
- se non ci sono parametri oppure se il nome dell'host è uguale a null, allora l'host di riferimento è quello locale

RMI:IL SERVER

Dopo che il server ha registrato lo stub relativo all'oggetto remoto esportato,

- il main termina
- esiste comunque un thread attivo in attesa di invocazione di metodi remoti sul serversocket associato, per cui il programma non termina
- il thread rimane attivo fintanto che
  - Il binding creato per quell'oggetto rimane valido e
  - Esistono riferimenti all'oggetto remoto in clients remoti

JAVA : IL REGISTRY

Supponiamo che registry sia l'istanza di un registro individuato mediante getregistry( )

- registry.bind (...) crea un collegamento tra un nome simbolico (qualsiasi) ed un riferimento all'oggetto. Se esiste già un collegamento per lo stesso oggetto all'interno dello stesso registry, viene sollevata una eccezione

- registry.rebind (...) crea un collegamento tra un nome simbolico (qualsiasi) ed un riferimento all'oggetto. Se esiste già un collegamento per lo stesso oggetto all'interno dello stesso registry, tale collegamento viene sovrascritto

- **è** possibile inserire più istanze dello stesso oggetto remoto nel registry, con nomi simbolici diversi

JAVA: ATTIVAZIONE DEL SERVIZIO

Per rendere disponibile i metodi dell'oggetto remoto, è necessario attivare due tipi di servizi

- il registry che fornisce il servizio di registrazione di oggetti remoti
- Il server implementato fornisce accesso ai metodi remoti

Per attivare il registry in background:

$ rmiregistry & (in LINUX)

$ start rmiregistry (in WINDOWS)

- viene attivato un registry associato per default alla porta 1099
- se la porta è già utilizzata, viene sollevata un'eccezione. Si può anche scegliere esplicitamente una porta

$ rmiregistry 2048 &

RMI REGISTRY

- Il registry viene eseguito per default sulla porta 1099
- Se si vuole eseguire il registry su una porta diversa, occorre specificare il numero di porta da linea di comando, al momento dell'attivazione

  start rmiregistry 2100

- la stessa porta va indicata sia nel client che nel server al momento del reperimento del riferimento al registro, mediante LocateRegistry.getRegistry

  Registry registry = LocateRegistry.getRegistry(2100);

- NOTA BENE: il registry ha bisogno dell'interfaccia e dei .class, per cui attenti a come sono impostati i path!

IL CLIENT RMI

Il client:

- ricerca uno stub per l'oggetto remoto
- invoca i metodi dell'oggetto remoto come fossero metodi locali (l'unica differenza è che occorre intercettare RemoteException)

Per ricercare il riferimento allo stub, il client

- deve accedere al registry attivato sul server.

- il riferimento restituito dal registry è un riferimento ad un oggetto di tipo Object: è necessario effettuare il casting dell'oggetto restituito al tipo definito nell'interfaccia remota

IL CLIENT RMI

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;
 public class Client {
      private Client(  ) {   }
      public static void main(String[  ] args) throws
                                                  Exception
      {
         String host = args[0];
         Scanner s = new Scanner(System.in);
         String next = s.next();
```

IL CLIENT RMI

```
try {
    Registry registry = LocateRegistry.getRegistry(host);
    EchoInterface stub = (EchoInterface)registry.lookup("Echo");
    String response = stub.getEcho(next);
    System.out.println("response: " + response);
} catch (Exception e) {
    System.err.println("Client exception: " +e.toString());
    e.printStackTrace();
}}}
```

LOCALIZZARE IL REGISTRY

- Forma generale del metodo LocateRegistry.getRegistry

```
public static Registry getRegistry(String host,  int port)
                                          throws RemoteException
```

Restituisce un riferimento (stub) ad un oggetto Registry attivato sull'host e sulla porta specificata

- Il metodo può essere utilizzato dal client per individuare il servizio di registry attivato sul server

L'esecuzione del client richiede

- la classe EchoRMIClient.class, risultante della compilazione del client
- la classe EchoInterface.class

ESERCIZIO

Sviluppare una applicazione RMI per la gestione di un'elezione. Il server esporta un insieme di metodi

- `public void vota (String nome)`. Accetta come parametro il nome del candidato. Non restituisce alcun valore. Registra il voto di un candidato in una struttura dati opportunamente scelta.

- `public int risultato (String nome)` Accetta come parametro il nome di un candidato e restituisce i voti accumulati da tale candidato fino a quel momento.

- un metodo che consenta di ottenere i nomi di tutti i candidati, con i rispettivi voti, ordinati rispetto ai voti ottenuti

RMI: IL MECCANISMO DELLE CALLBACK

RMI utilizza una comunicazione

- sincrona a rendez vouz esteso: il client invoca un metodo remoto e si blocca finchè il metodo non termina

Molte applicazioni fanno riferimento an pattern di interazione asincrono, in cui

- il client è interessato ad un evento che si verifica sul server e notifica il suo interesse al server (ad esempio utilizzando RMI)
- il server registra che il client è interessato in quell'evento
- il client prosegue la sua elaborazione dopo la notifica al server
- quando l'evento si verifica, il server lo notifica ai clients interessati l'accadimento dell'evento

RMI : IL MECCANISMO DELLE CALLBACK

Esempi di applicazioni:

- un utente partecipa ad una un gruppo di discussione (es: Facebook) e vuol essere avvertito quando un nuovo utente entra nel gruppo.

- lo stato di un gioco multiplayer viene gestito da un server. I giocatori notificano al server le modifiche allo stato del gioco. Ogni giocatore deve essere avvertito quando lo stato del gioco subisce delle modifiche.

- gestione distribuita di un'asta: un insieme di utenti partecipa ad un'asta distribuita. Ogni volta che un utente fa una nuova offerta, tutti i partecipanti all'asta devono essere avvertiti

RMI: IL MECCANISMO DELLE CALLBACK

Soluzioni possibili per realizzare nel server un servizio di notifica di eventi al client:

- Polling: il client interroga ripetutamente il server, per verificare se l'evento atteso si è verificato. L'interrogazione avviene mediante l'invocazione di un metodo remoto (mediante RMI).
  - Svantaggio: alto costo per l'uso non efficiente delle risorse del sistema

- Registrazione dei client interessati agli eventi e successiva notifica (asincrona) del verificarsi dell'evento ai client da parte del server
  - Vantaggio: il client può proseguire la sua elaborazione senza bloccarsi ed essere avvertito, in modo asincrono, quando l'evento si verifica
  - Problema: quale meccanismo utilizza il server per risvegliare il client?

RMI: IL MECCANISMO DELLE CALLBACK

- E' possibile utilizzare RMI sia per l'interazione client-server (registrazione del client) che per quella server-client (notifica del verificarsi di un evento) utilizzando il meccanismo delle callback
- Il server definisce una interfaccia remota ServerInterface che include un metodo remoto che può essere utilizzato dal client per registrarsi
- Il client definisce una interfaccia remota ClientInterface che definisce un metodo remoto utilizzato dal server per notificare un evento al client
- Il client ha a disposizione la ServerInterface e reperisce il puntatore all'oggetto remoto tramite il registry
- Il server ha a disposizione la ClientInterface e riceve al momento della registrazione del client il riferimento all'oggetto remoto sul client

RMI: IL MECCANISMO DELLE CALLBACK

- Il client, al momento della registrazione sul server, passa al server un riferimento RC ad un oggetto che implementa la ClientInterface
- Il server memorizza RC in una sua struttura dati (ad esempio, un vector)
- al momento della notifica, il server utilizza RC per invocare il metodo remoto di notifica definito dal client.
- In questo modo rendo 'simmetrico' il meccanismo di RMI, ma…

  il client non registra l'oggetto remoto in un rmiregistry, ma passa un riferimento a tale oggetto al server, al momento della registrazione

RMI CALLBACKS

Un client può richiedere servizi ad un servente mediante RMI. Talvolta è utile poter consentire al servente di contattare il client

- il servente notifica degli eventi ai propri client per evitare il polling effettuato dai clients
- il servente accede allo stato della sessione se questo è memorizzato presso il client meccanismo, analogo a quello dei cookie

La soluzione è offerta dal meccanismo delle callback in cui il servente può richiamare il client

In questo modo il meccanismo di RMI viene utilizzati in modo bidirezionale

- dal client al server (oggetto reperito via registry)
- Dal server al client (oggetto reperito mediante passaggio di parametri

RMI CALLBACKS

- Il client crea un oggetto remoto, oggetto callback OC, che implementa un'interfaccia remota che deve essere nota al servente
- Il servente definisce un oggetto remoto OS, che implementa una interfaccia remota che deve essere nota al client
- Il client reperisce OS mediante il meccanismo di lookup di un registry
- OS contiene un metodo che consente al client di registrare il proprio OC presso il server
- quando il servente ne ha bisogno, può contattare **OC**, reperendo un riferimento ad OC dalla struttura dati in cui lo ha memorizzato al momento della registrazione

CALLBACKS: UN ESEMPIO

Server:

- Definisce un oggetto remoto che fornisce ai clients metodi per
  - Registrare/cancellare una callback
  - Contattare il server (metodo SayHello)

Client

- Registra una callback presso il server. La callback consente al server di notificare ai clients registrati ogni contatto stabilito dai clients mediante il metodo SayHello
- Effettua un numero casuale di richieste del metodo sayHello
- Cancella la propria registrazione

L'INTERFACCIA DEL CLIENT

```
import java.rmi.*;
public interface CallbackHelloClientInterface extends Remote
   {
    /* Metodo invocato dal  server per effettuare
     una callback a un client remoto. */
    public void notifyMe(String message) throws
                                                RemoteException;
    { ...    }
```

notifyMe(…)

è il metodo esportato dal client e che viene utilizzato dal server per la notifica di un nuovo contatto da parte di un qualsiasi client. Viene notificato il nome del client che ha contattato il server.

L'INTERFACCIA DEL CLIENT:IMPLEMENTAZIONE

```
import java.rmi.*; import java.rmi.server.*;
public class CallbackHelloClientImpl  implements
                              CallbackHelloClientInterface {
  /* crea un nuovo callback client */
  public CallbackHelloClientImpl( ) throws RemoteException
         { super( ); }
        /* metodo che può essere richiamato dal servente */
  public void notifyMe(String message) throws RemoteException
  {
     String returnMessage = "Call back received: " + message;
     System.out.println( returnMessage); }
   }
```

ATTIVAZIONE DEL CLIENT

```
import java.rmi.*;
public class CallbackHelloClient {
public static void main(String args[ ]) {
  try {vedi lucido successivo.....} catch (Exception e)
       {
       System.err.println("HelloClient exception:"+
                                  e.getMessage(  ));
       }
           }

                                                    }
```

IL CODICE DEL CLIENT

```
System.out.println("Cerco CallbackHelloServer");
Registry registry = LocateRegistry.getRegistry("localhost",2048);
String name = "CallbackHelloServer";
CallbackHelloServerInterface h =
        (CallbackHelloServerInterface) registry.lookup(name);
/* si registra per il callback */
System.out.println("Registering for callback");
CallbackHelloClientImpl callbackObj =
                              new CallbackHelloClientImpl();
CallbackHelloClientInterface stub =
     (CallbackHelloClientInterface)
             UnicastRemoteObject.exportObject(callbackObj, 0);
h.registerForCallback(stub);
```

IL CODICE DEL CLIENT

```
/* accesso al server - fa una serie casuale di 5-15richieste */
int n = (int) (Math.random( )*10+5);
String nickname=  ”mynick”;
for (int i=0; i< n; i++) {
        String message = h.sayHello(nickname);
        System.out.println( message);
        Thread.sleep(1500);};
/* cancella la registrazione per il callback */
System.out.println("Unregistering for callback");
h.unregisterForCallback(callbackObj);
```

L'INTERFACCIA DEL SERVER

```
import java.rmi.*;
public interface CallbackHelloServerInterface extends Remote
    {
     /* metodo di notifica */
     public String sayHello(String name) throws RemoteException;
     /* registrazione per il callback */
     public void registerForCallback
                 (CallbackHelloClientInterface callbackClient)
                                    throws RemoteException;
     /* cancella registrazione per il callback */
     public void  unregisterForCallback
            (CallbackHelloClientInterface callbackClient)
                                    throws RemoteException;
```

L'IMPLEMENTAZIONE DEL SERVER

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
public class CallbackHelloServerImpl implements
                                        CallbackHelloServerInterface
    /* lista dei client registrati */
    private List<CallbackHelloClientInterface> clients;
    /* crea un nuovo servente */
    public CallbackHelloServerImpl()throws RemoteException
            {super( );
   clients = new ArrayList<CallbackHelloClientInterface>( );

                                                    }
```

L'IMPLEMENTAZIONE DEL SERVER

```
public synchronized void registerForCallback
 (CallbackHelloClientInterface callbackClient) throws RemoteException
  {if (!clients.contains(callbackClient))
      { clients.add(callbackClient)
        System.out.println(" New client registered." );}
}
/* annulla registrazione per il callback */
public synchronized void unregisterForCallback
  (CallbackHelloClientInterface callbackClient) throws RemoteException
  {if (clients.remove(callbackClient)) {System.out.println("Client
                                                    unregistered");}
  else { System.out.println("Unable to unregister client."); }
  }   }
```

L'IMPLEMENTAZIONE DEL SERVER

```
/* metodo di notifica
* quando viene richiamato, fa il callback a tutti i client
   registrati */
public String sayHello (String name) throws RemoteException {
                doCallbacks(name);
                return "Hello, " + name + "!";
                                                              }
```

IL SERVER:IMPLEMENTAZIONE

```
private synchronized void doCallbacks(String name ) throws
                                                    RemoteException
{ System.out.println("Starting callbacks.");
  Iterator i = clients.iterator( );
  int numeroClienti = clients.size( );
   while (i.hasNext()) {
         CallbackHelloClientInterface client =
                                    (CallbackHelloClientInterface)
         i.next();
         client.notifyMe(name);
                          }
    System.out.println("Callbacks complete.");}
         }
```

IL CODICE DEL SERVER

```
import java.rmi.server.*; import java.rmi.registry.*;
public class CallbackServer   {
public static void main(String[  ] args) {
try{ /*registrazione presso il registry */
   System.out.println("Binding CallbackHello");
   CallbackHelloServerImpl server =  new CallbackHelloServerImpl( );
   CallbackHelloServerInterface stub=(CallbackHelloServerInterface)
                        UnicastRemoteObject.exportObject (server,39000);
   String name = "CallbackHelloServer";
   Registry registry = LocateRegistry.getRegistry ("localhost",2048);
   registry.bind  (name, stub);
   System.out.println("CallbackHello bound");
} catch (Exception e) { System.out.println("Eccezione" +e);   }}}}
```

RMI:ECCEZIONI

- Eccezione che viene sollevata se non trova un servizio di registry su quella porta. Esempio:

```
HelloClient exception: Connection refused to host:
192.168.2.103; nested exception is:
java.net.ConnectException: Connection refused: connect
```

- Eccezione sollevata e si tenta di registrare più volte lo stesso stub con lo stesso nome nello stesso registry

Esempio

```
CallbackHelloServer exception:
java.rmi.AlreadyBoundException:
CallbackHelloServer java.rmi.AlreadyBoundException:
CallbackHelloServer
```

ESERCIZIO 1: CALLBACKS

Considerare l'esercizio precedente in cui si richiede di implementare la gestione Elettronica di una elezione a cui partecipano un numero prefissato di candidati. Si chiedeva di realizzare un server RMI che consentisse al client di votare un candidato e di richiedere il numero di voti ottenuti dai candidati fino ad un certo punto.

Modificare l'esercizio in modo che il server notifichi ogni nuovo voto ricevuto a tutti i clients che hanno votato fino a quel momento. La registrazione dei clients sul server avviene nel momento del voto.

ESERCIZIO 2: GESTIONE FORUM

Si vuole implementare un sistema che implementi un servizio per la gestione di forum in rete. Un forum è caratterizzato da un argomento su cui diversi utenti possono scambiarsi opinioni via rete. Il sistema deve prevedere un server RMI che fornisca le seguenti funzionalita':

a) apertura di un nuovo forum, di cui è specificato l'argomento (esempio: giardinaggio)

b) inserimento di un nuovo messaggio indirizzato ad un forum identificato dall'argomento (es: è tempo di piantare le viole, indirizzato al forum giardinaggio)

c) reperimento dell'ultimo messaggio inviato ad un forum di cui è specificato l'argomento.

Il messaggio può essere richiesto esplicitamente dal client oppure può essere notificato ad un client precedentemente registrato