Lezione 15
Remote Method Invocation - 3

Vittorio Scarano
Corso di Programmazione Distribuita (2003-2004)
Laurea di I livello in Informatica
Università degli Studi di Salerno

Organizzazione della lezione

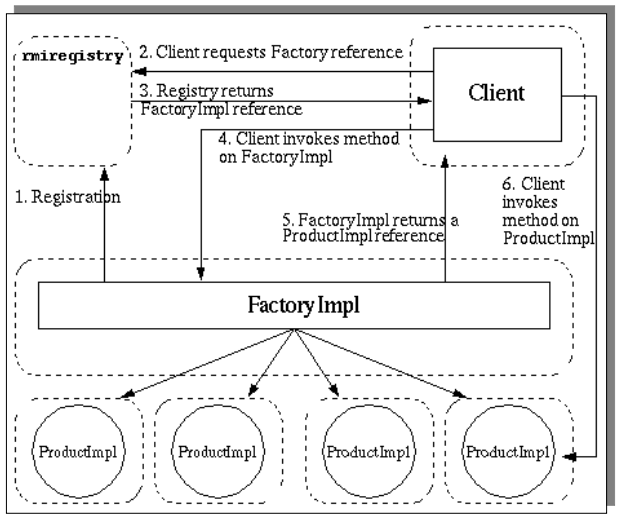
- Il design pattern della factory in RMI
- Garbage collection distribuita
- Il logging dei metodi

Factory design pattern

- Uno dei *Design Patterns* più comuni
- Una factory è un oggetto che controlla la creazione e/o l'accesso ad oggetti
- Particolarmente utile in vari contesti:
  - nel caso di oggetti remoti, permette la registrazione solamente della factory
  - rendendo minimo l'overhead di comunicazione con il registry
  - permettendo il controllo (da parte del server) sul numero, tipo e qualità dei servizi offerti dal server

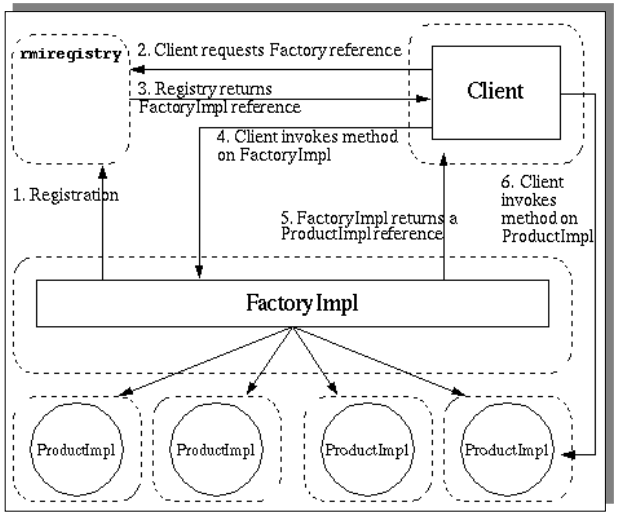
Lo schema del Factory Design Pattern - 1

- L'oggetto Factory si registra sul registry
- Il Client ottiene un riferimento remoto alla Factory
- …e chiede alla Factory di creare/dare accesso ad un oggetto Prodotto



Lo schema del Factory Design Pattern - 2

- … il cui riferimento remoto viene restituito al Client
- che può usarne i servizi



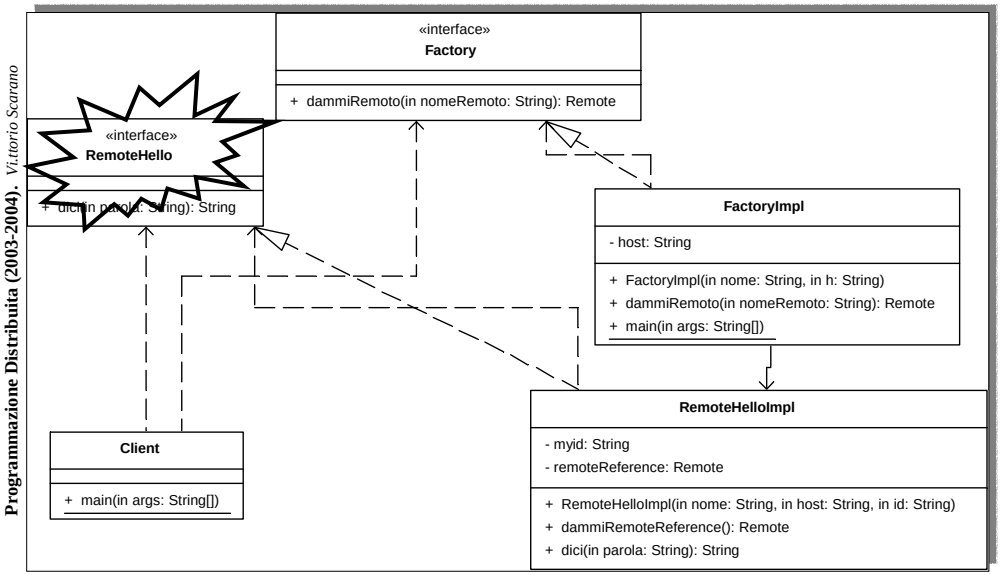
Alcuni requirements per RMI

- Per usare il Factory Design Pattern in RMI
  - sia la classe Factory che la classe Product devono avere una interfaccia remota
  - che ne specifichi i servizi da usare da remoto
- Tra i servizi della Factory
  - un metodo per avere un riferimento remoto di un Product
- Tra i servizi del Product
  - i metodi per accedere ai servizi del Product

Un esempio per la Factory in RMI: RemoteHello

- Il “solito” servizio di saluto remoto
- Composto da una Factory di oggetti RemoteHello
  - il cui riferimento viene passato al Client
  - che può effettuare le chiamate richieste
- Due interface:
  - una per specificare i servizi remoti offerti dalla Factory
    - ottenere un riferimento remoto di un oggetto RemoteHello creato
  - una per specificare i servizi remoti offerti da RemoteHello
    - rispondere con una stringa di saluto

Il diagramma di FactoryExample

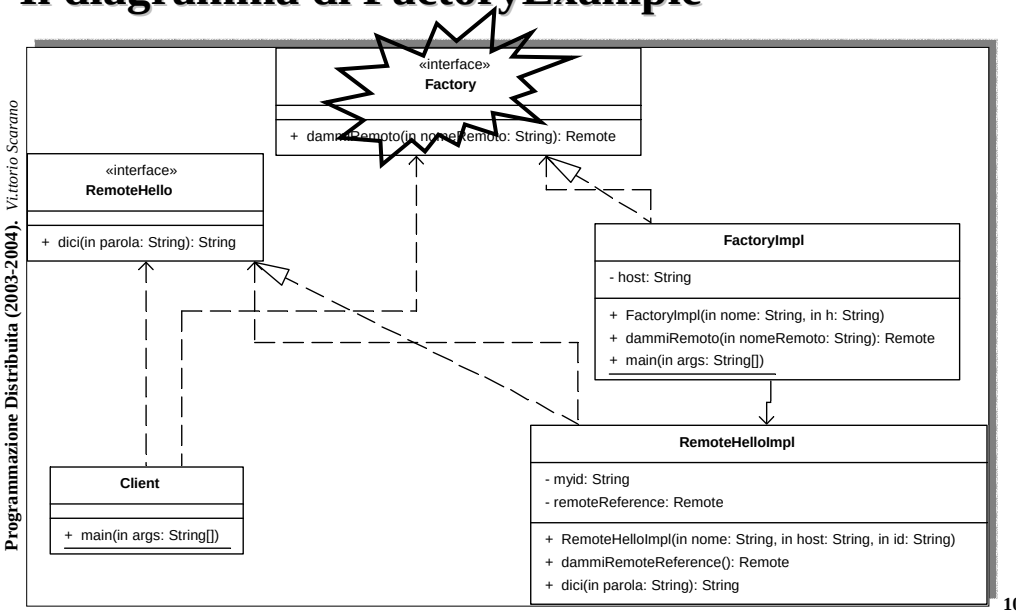


La interface remote: RemoteHello.java


```
import java.rmi.Remote;
import java.rmi.RemoteException;
/**
 * Interfaccia RemoteHello: la factory
 * @author Vittorio Scarano
 * @version 1.0 12/5/2002
 */
public interface RemoteHello
     extends Remote {
   /**
    * Saluta il client, con il proprio nome
    *
    * @param  parola  tringa inviata dal client
    * @return una stringa di benvenuto!
    */
   String dici(String parola)
      throws RemoteException;
}
```

- Import per uso delle interface
- Commenti per JavaDoc
- Unico metodo remoto
  - lancia una eccezione remota

Il diagramma di FactoryExample

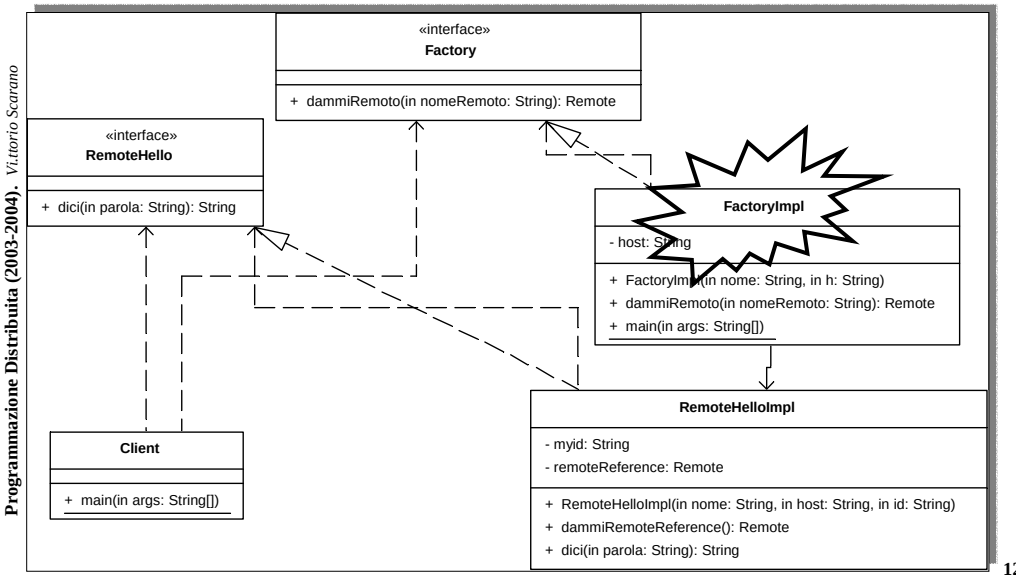


La interface remote: Factory.java


```
import java.rmi.Remote;
import java.rmi.RemoteException;
/**
 * Interfaccia per la factory di Remote Hello
 * @author     Vittorio Scarano
 * @created    14 maggio 2002
 * @version    1.0 12/5/2002
 */
public interface Factory
          extends Remote {
    Remote dammiRemoto(String
                  nomeRemoto )
                       throws RemoteException;
}
```

- Import per uso delle interface
- Commenti per JavaDoc
- Unico metodo remoto
  - restituisce un oggetto la classe implementa Remote
  - lancia una eccezione remota

Il diagramma di FactoryExample



La factory: FactoryImpl.java (1)


```
import java.rmi.*;
import java.rmi.server.*;
public class FactoryImpl
   extends UnicastRemoteObject
    implements Factory {
   /* args[0] e' l'host su cui creare factory */

  public static void main (String args[]) {
   System.setSecurityManager (new
                    RMISecurityManager());
   try {
      FactoryImpl f = new FactoryImpl (
                "FactoryRemoteHello", args[0]);
      } catch (Exception e)
         {System.out.println ("Eccezione:"+e);}
   System.out.println ("Pronto!");
   } // end main
    // continua…
```

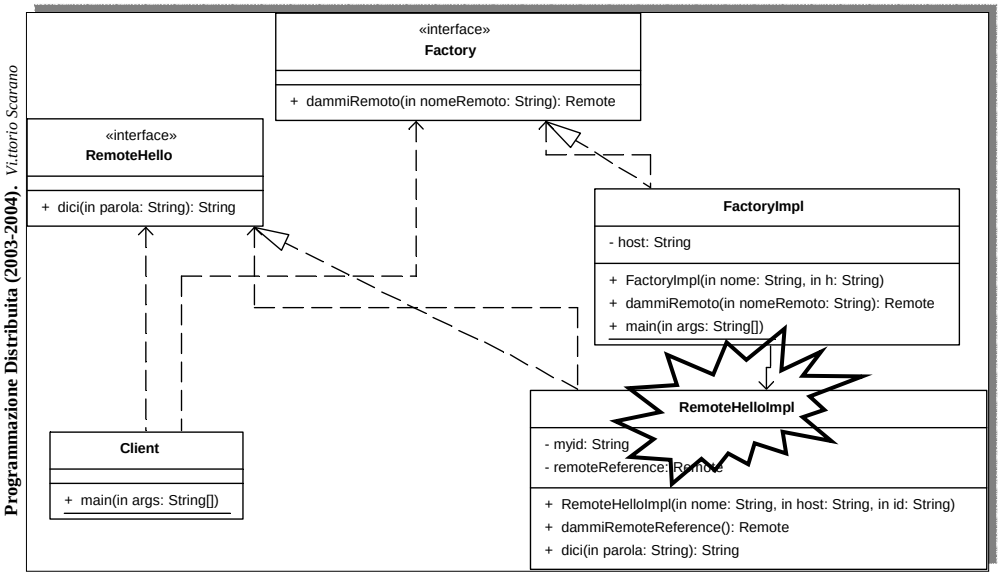
- Server per Factory
  - comprende il main()
- Set SM
- Creazione di una FactoryImpl
  - passando l'host come parametro

La factory: FactoryImpl.java (2)


```
  public FactoryImpl (String nome, String h)
                throws RemoteException{
   super();
   host = h;
   try { Naming.rebind(
                "rmi://"+host+"/"+nome,this);
   } catch (Exception e)
    {System.out.println ("Eccezione:"+e);}
  } // fine costruttore
  public Remote dammiRemoto (String
                                  nomeRemoto)
     throws RemoteException {
       RemoteHelloImpl rh =
            new RemoteHelloImpl("RemoteHello",
                                host, nomeRemoto);
     return rh.dammiRemoteReference();
     }
   private String host;
}
```

- Costruttore
  - costruttore di UnicastRemoteObject
- Bind sul registry
  - con il nome passato
- Metodo remoto
  - factory: istanzia un RemoteHello
    - con il nome passato dal client
  - restituisce il suo riferimento remoto
- Variabile istanza:
  - host

Il diagramma di FactoryExample



Il servizio remoto: RemoteHelloImpl.java (1)


```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class RemoteHelloImpl
      extends UnicastRemoteObject
      implements RemoteHello {
   public RemoteHelloImpl (String nome,
                           String host, String id)
      throws RemoteException{
         super();
         try {
            Naming.rebind
                 ("rmi://"+host+"/"+nome,this);
            myid = id;
            remoteReference =
                           Naming.lookup (nome);
         } catch (Exception e)
            {System.out.println ("Ecc.:"+e);}
   } // fine costruttore… continua
```

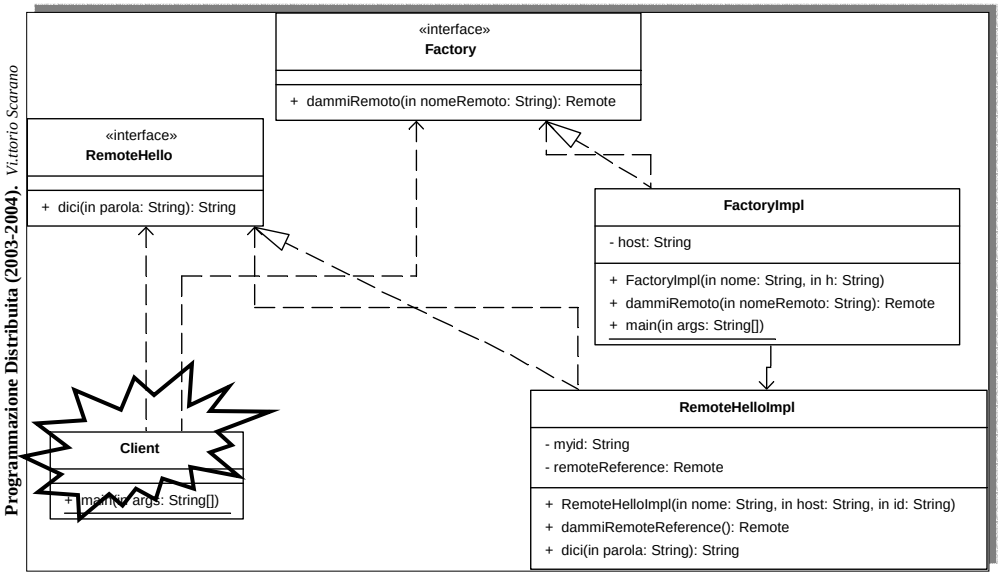
- Import per uso delle interface e superclasse
- Costruttore
  - chiamata costruttore di UnicastRemoteObject
  - bind sul registry
  - setting delle variabili istanza
    - id dato dal Client
    - riferimento remoto

Il servizio remoto: RemoteHelloImpl.java (2)

```
    public String dici (String parola)
            throws RemoteException {
        System.out.println(myid+": Ricevuto "+
                                        parola);
        return ("Sono "+myid+" e ti saluto!");
        }

    public Remote dammiRemoteReference() {
        return remoteReference;
    }

    private String myid;
    private Remote remoteReference;
}
```

- Implementazione metodo remoto
  - restituisce anche l'id
- Metodo locale
  - serve per la Factory per prelevare il riferimento remoto
- Variabili istanza
  - stringa di ID
  - riferimento remoto

Il diagramma di FactoryExample

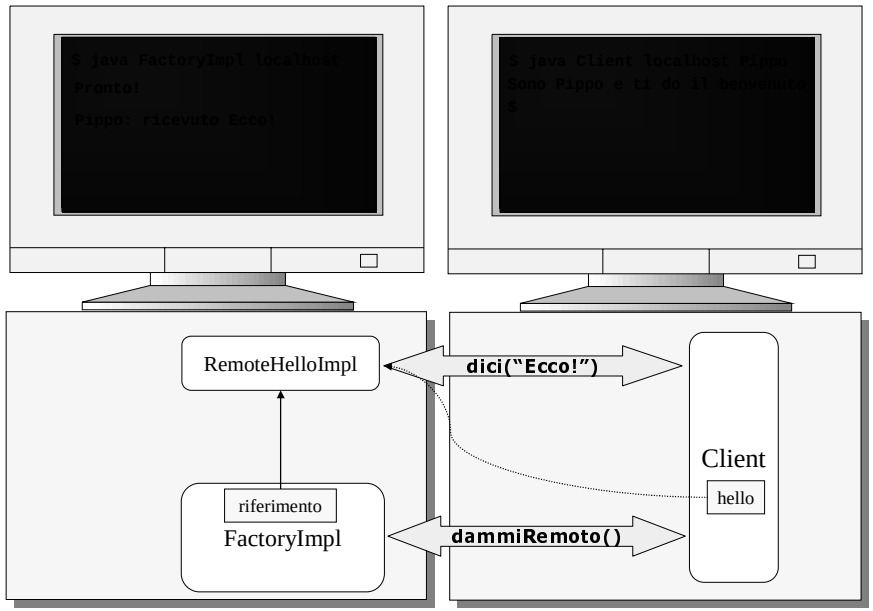


Il client: Client.java

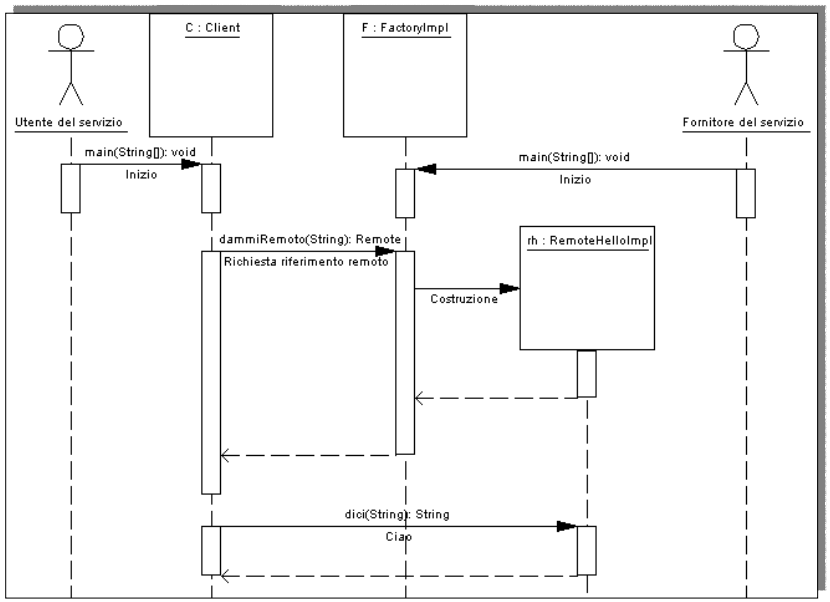
```
import java.rmi.*;
public class Client {

public static void main (String args[]) {
  System.setSecurityManager (new
                    RMISecurityManager());

  try {
     Factory fact = (Factory)
       Naming.lookup (
     "rmi://"+args[0]+"/FactoryRemoteHello");
     RemoteHello hello = (RemoteHello)
                    fact.dammiRemoto(args[1]);
     System.out.println(hello.dici ("Ecco!"));
  } catch (Exception e)
  { System.out.println("Eccezione"+e);}
}
}
```

- Parametri del main
  - host del Factory
  - nome del client
- Set SM
- Ricerca della Factory
- Chiede un riferimento ad un oggetto di RemoteHello
- Chiama il metodo remoto

Il funzionamento del programma



Il sequence diagram dell'esempio



Organizzazione della lezione

- Il design pattern della factory in RMI
- Garbage collection distribuita
- Il logging dei metodi

La Garbage Collection

- Due filosofie nei linguaggi di programmazione
- La filosofia C:
  - "*La gestione della memoria è troppo importante per essere gestita dal sistema*"
- La filosofia Smalltalk (e Java)
  - "*La gestione della memoria è troppo importante per essere gestita dal programmatore*"
- Vediamo un esempio della garbage collection in locale

Esempio di Garbage Collection

```
public class Garbage
{
public static void main(String args[ ])
{
    String s;
    s = new String("Ciao");

    s = s + " Vittorio";
    // chiamiamo il garbage collector
    System.gc();

    System.out.println(s);
    return;
}
}
```

- Allochiamo un oggetto stringa "Ciao" riferito dalla variabile s
- Concateniamo questa stringa ad una altra stringa
  - la variabile s adesso punta ad un nuovo oggetto stringa "Ciao Vittorio"
  - quindi la chiamata al garbage collector la elimina

Garbage Collection distribuita - 1

- Se scrivere un algoritmo (ed implementarlo) per la Garbage Collection di oggetti locali è difficile…
- … scriverlo per oggetti distribuiti è davvero improbo
- La politica di Java RMI
  - ereditata dalla politica degli oggetti di Modula 3
- Ogni JVM mantiene il conto dei riferimenti remoti che referenziano un oggetto

Garbage Collection distribuita - 2

- Quando l'ultimo riferimento viene eliminato, il riferimento diventa "weak"
  - possibile eliminare l'oggetto se non esistono riferimenti locali all'oggetti
- E' possibile la interazione con il GC distribuito:
  - Se un oggetto implementa la interface java.rmi.server.Unreferenced
    - deve implementare il metodo unreferenced()
    - che viene chiamato quando l'ultimo riferimento "live" viene eliminato

Garbage Collection distribuita - 3

- Quando un client ottiene un riferimento remoto
  - il DGC considera l'oggetto come *dirty*
  - affidando però un *lease* (fitto) per una durata di tempo determinata
    - settabile con la proprietà java.rmi.dgc.leaseValue (in millisecondi)
  - al termine del quale il riferimento viene considerato "weak"
  - e quindi l'oggetto può essere collezionato dal GC
- E' responsabilità del client "rinnovare" il *lease*
  - e considerare nel codice (attenzione!) la possibilità che per qualche problema sulla connessione, l'oggetto remoto non ci sia più

Implementazione del DGC

- Implementa una interface java.rmi.dgc che eredita da Remote
- Due metodi: dirty() e clean()
- Le chiamate a dirty() vengono effettuate dal client quando una reference remota viene *unmarshaled*
  - sono chiamate remote nascoste nel meccanismo di RMI
- Le chiamate a clean() vengono fatte dal client quando non esistono più riferimenti remoti all'oggetto

Il metodo dirty()

public Lease dirty(ObjID[ ] ids, long sequenceNum, Lease lease) throws RemoteException

- ObjID sono assegnati dal server agli oggetti remoti
  - unici
- sequenceNum
  - serve per scartare eventuali chiamate “giunte in ritardo”
- lease
  - contiene un identificatore unico per la JVM del client (VMID) ed un periodo richiesto
- viene restituito un lease

L’identificazione dei client: il VMID

- Una classe che permette di costruire identificatori univoci per una macchina virtuale Java
  - basato anche su IP address della macchina reale
- In alcuni casi, quindi, non può essere generato
  - se, ad esempio, per sicurezza non viene permesso l’accesso a questa informazione
- Requisiti per la univocità
  - stessi requisiti di java.rmi.server.UID (ID uniche su JVM)
    - host che non fa il reboot in un millisecondo e il cui clock non viene mai messo all’indietro
  - indirizzo IP unico e costante per la durata dell’oggetto

Organizzazione della lezione

- Il design pattern della factory in RMI
- Garbage collection distribuita
- Il logging dei metodi

Logging delle chiamate

- Possibile chiedere alla JVM di effettuare il logging delle chiamate “nascoste” di RMI
  - meccanismo che integra la capacità di costruire oggetti “logger” fornite da JDK 1.4
    - che permette di effettuare logging in testo o formattato verso la memoria, stream di output, console, file e socket
- Offerto (in maniera minimale) da RMI
  - package java.rmi
- Implementato (in maniera vendor-specific) da SUN
  - package sun.rmi

Uso del logging per RMI

- Mediante passaggio di parametri su linea di comando alla JVM
- Possibile
  - monitorare sia il server (RMI) che il client (SUN)
  - scegliere il dettaglio (tra SILENT, BRIEF, VERBOSE)
- Nei log compaiono, tra l'altro
  - dettagli sulle chiamate al registry per bind e lookup
  - dettagli sulle chiamate remote effettuate
  - dettagli sulle chiamate per il Distributed Garbage Collector

Come usare il logging: le proprietà

- java.rmi.server.logCalls
  - logging effettuato lato server verso System.err
- sun.rmi.server.logLevel
  - controlla il livello di log, può assumere valori di SILENT, BRIEF e VERBOSE
- sun.rmi.client.logCalls
  - logging effettuato lato client.

Il funzionamento del log lato server (1)

```
C:\Sincronia File\Asdsd\Progetti\FactoryExample>java -Djava.security.
        policy=policyall -Djava.rmi.server.logCalls=true
        -Dsun.rmi.server.logLevel=Brief FactoryImpl localhost
15-mag-2002 18.49.08 sun.rmi.server.UnicastRef newCall
FINE: main: get connection
15-mag-2002 18.49.11 sun.rmi.server.UnicastServerRef logCall
MOLTO FINE: RMI TCP Connection(1)-10.0.0.201: [10.0.0.201: sun.rmi.transport.DGC
Impl[0:0:0, 2]: java.rmi.dgc.Lease dirty(java.rmi.server.ObjID[], long, java.rmi
.dgc.Lease)]
15-mag-2002 18.49.11 sun.rmi.server.UnicastRef done
FINE: main: free connection (reuse = true)
Pronto!
```


Il funzionamento del log lato server

```
15-mag-2002 18.53.10 sun.rmi.server.UnicastServerRef logCall
MOLTO FINE: RMI TCP Connection(2)-10.0.0.201: [10.0.0.201: sun.rmi.transport.DGC
Impl[0:0:0, 2]: java.rmi.dgc.Lease dirty(java.rmi.server.ObjID[], long, java.rmi
.dgc.Lease)]
15-mag-2002 18.53.10 sun.rmi.server.UnicastServerRef logCall
MOLTO FINE: RMI TCP Connection(2)-10.0.0.201: [10.0.0.201: FactoryImpl[0]: publi
c abstract java.rmi.Remote Factory.dammiRemoto(java.lang.String) throws java.rmi
.RemoteException]
15-mag-2002 18.53.10 sun.rmi.server.UnicastRef newCall
FINE: RMI TCP Connection(2)-10.0.0.201: get connection
15-mag-2002 18.53.10 sun.rmi.server.UnicastServerRef logCall
MOLTO FINE: RMI TCP Connection(3)-10.0.0.201: [10.0.0.201: sun.rmi.transport.DGC
Impl[0:0:0, 2]: java.rmi.dgc.Lease dirty(java.rmi.server.ObjID[], long, java.rmi
.dgc.Lease)]
15-mag-2002 18.53.10 sun.rmi.server.UnicastRef done
….
Pippo: Ricevuto Finalmente!
```

Il funzionamento del log lato client (1)


```
C:\Sincronia File\Asdsd\Progetti\FactoryExample>java -Djava.security.policy=poli
cyall -Dsun.rmi.client.logCalls=true -Dsun.rmi.client.logLevel=BRIEF Client loca
lhost Pippo
15-mag-2002 19.06.27 sun.rmi.server.UnicastRef logClientCall
MOLTO FINE: main: outbound call: [endpoint:[localhost:1099](remote),objID:[0:0:0
, 0]] : sun.rmi.registry.RegistryImpl_Stub[0:0:0, 0]: java.rmi.Remote lookup(jav
a.lang.String)
15-mag-2002 19.06.29 sun.rmi.server.UnicastRef logClientCall
MOLTO FINE: main: outbound call: [endpoint:[10.0.0.201:1320](remote),objID:[0:0:
0, 2]] : sun.rmi.transport.DGCImpl_Stub[0:0:0, 2]: java.rmi.dgc.Lease dirty(java
.rmi.server.ObjID[], long, java.rmi.dgc.Lease)
15-mag-2002 19.06.29 sun.rmi.server.UnicastRef invoke
MOLTO FINE: main: method: public abstract java.rmi.Remote Factory.dammiRemoto(ja
va.lang.String) throws java.rmi.RemoteException
15-mag-2002 19.06.29 sun.rmi.server.UnicastRef logClientCall
MOLTO FINE: main: outbound call: [endpoint:[10.0.0.201:1320](remote),objID:[f336
75:edd5086870:-8000, 0]] : FactoryImpl_Stub[f33675:edd5086870:-8000, 0]: public
abstract java.rmi.Remote Factory.dammiRemoto(java.lang.String) throws java.rmi.R
emoteException
15-mag-2002 19.06.30 sun.rmi.server.UnicastRef logClientCall
MOLTO FINE: main: outbound call: [endpoint:[10.0.0.201:1320](remote),objID:[0:0:
0, 2]] : sun.rmi.transport.DGCImpl_Stub[0:0:0, 2]: java.rmi.dgc.Lease dirty(java
.rmi.server.ObjID[], long, java.rmi.dgc.Lease)
15-mag-2002 19.06.30 sun.rmi.server.UnicastRef invoke
```

Il funzionamento del log lato client (2)


```
….
MOLTO FINE: main: method: public abstract java.lang.String
RemoteHello.dici(java
.lang.String) throws java.rmi.RemoteException
15-mag-2002 19.06.30 sun.rmi.server.UnicastRef logClientCall
MOLTO FINE: main: outbound call:
[endpoint:[10.0.0.201:1320](remote),objID:[f336
75:edd5086870:-8000, 2]] : RemoteHelloImpl_Stub[f33675:edd5086870:-8000,
2]: pub
lic abstract java.lang.String RemoteHello.dici(java.lang.String) throws
java.rmi
.RemoteException
Sono Pippo e ti do' il benvenuto

C:\Sincronia File\Asdsd\Progetti\FactoryExample>
```