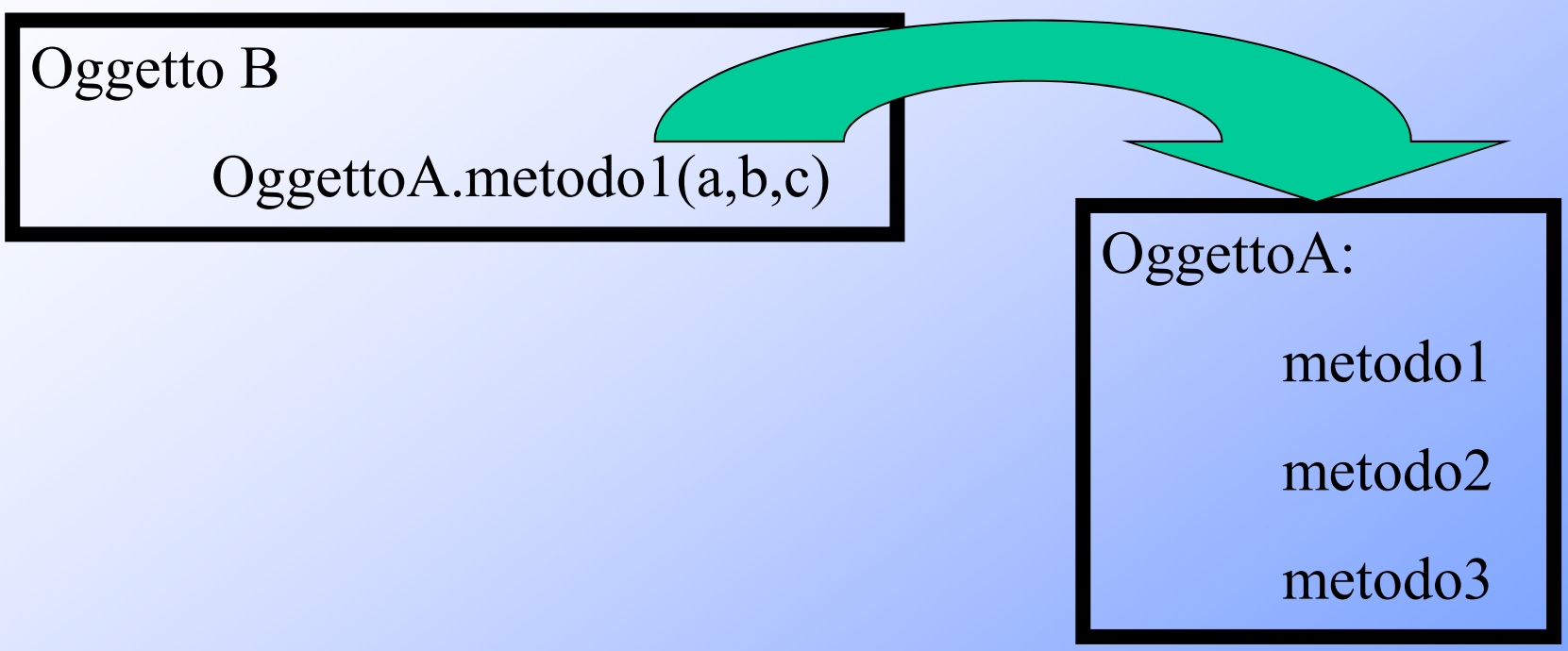
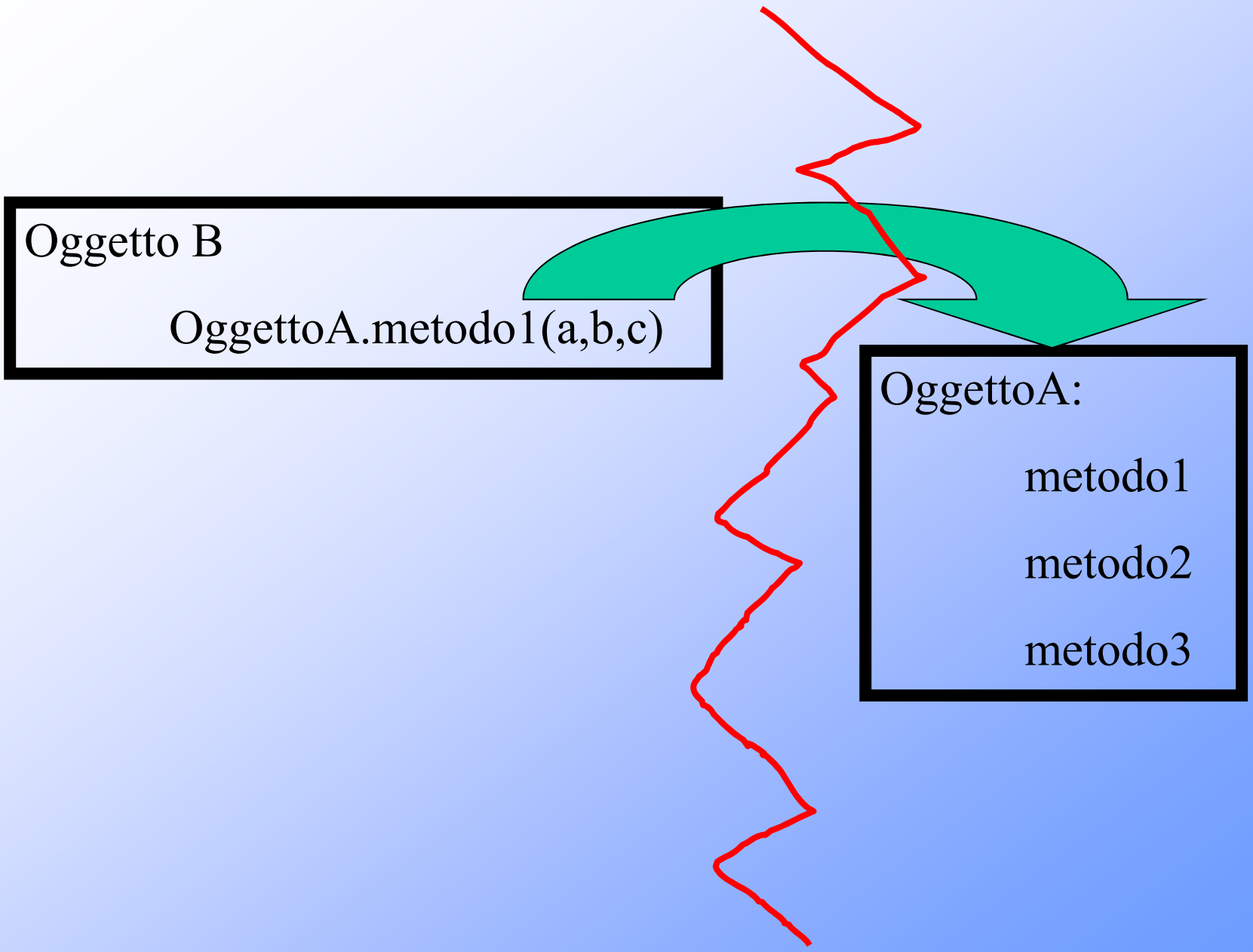
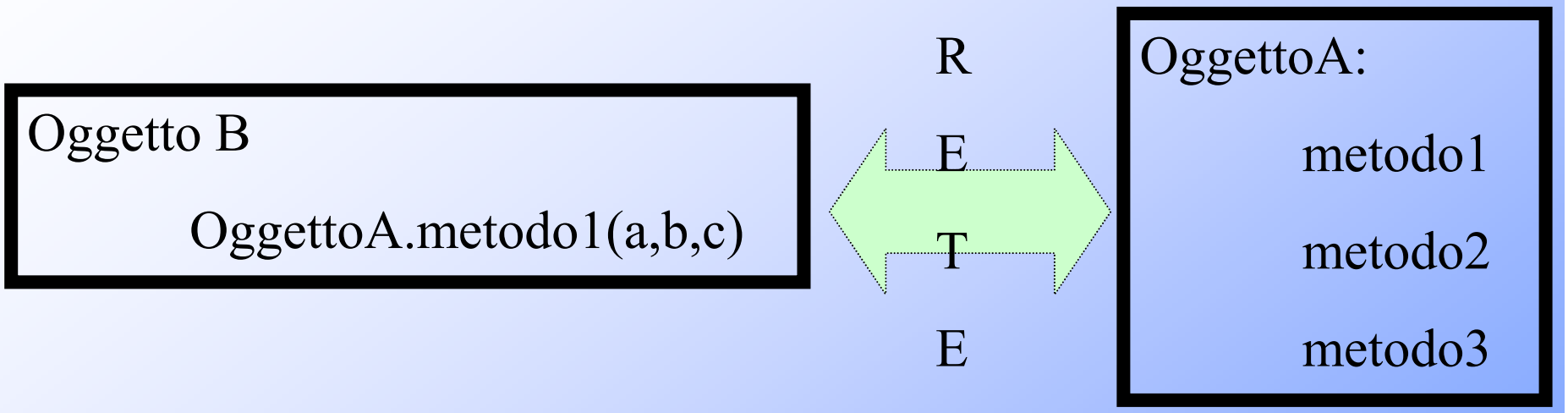

Oggetto B
OggettoA.metodo1(a,b,c)
OggettoA:
metodo1
metodo2
metodo3


Oggetto B
OggettoA.metodo1(a,b,c)
OggettoA:
metodo1
metodo2
metodo3


Oggetto B
OggettoA.metodo1(a,b,c)
R
E
T
E
OggettoA:
metodo1
metodo2
metodo3

Architetture Software 1

Remote Method Invocation

Mauro Migliardi Ph. D.

RMI

- invocazione locale o remota trasparentemente
  - stessa sintassi, ma attenzione ai dettagli
- object oriented
- puo' accedere solo ad ambienti Java "puri"
- puo' essere acceduta da ambienti language independent tramite CORBA
- e' molto piu' semplice di CORBA e COM/DCOM

Concetti base

- stub/skeleton (Java 1.x)
- stub/invocazione dinamica tramite reflections (Java 2)
- riferimenti possono essere persistenti e non
- e' possibile la attivazione lazy
- default su TCP, ma e' possibile usare socket speciali
  - e.g. SSL
- RemoteObject ridefinisce alcuni dei metodi della classe Object per garantire una semantica corretta

Object vs. RemoteObject

- equals confronta riferimenti remoti e non solo riferimenti locali
- hashCode e' lo stesso per i riferimenti ad uno stesso oggetto remoto (non e' piu' un concetto locale)
- toString da' informazioni sull'oggetto remoto e non sull'oggetto proxy (locale)
- Non c'e' metodo clone
  - oggetti che vogliono supportare l'interfaccia Cloneable devono specificare la semantica implementando il loro proprio metodo
- wait e notify hanno comportamento locale (sul proxy)

Livelli di RMI

- stub/skeleton
- remote reference
- trasporto
- applicazione

Stub/Skeleton

- stub/skeleton
  - marshall/unmarshall dei parametri
  - usa object serialization
  - oggetti locali passati per copia
  - oggetti remoti passati come riferimento remoto

Remote Reference

- gestisce la semantica dell'invocazione
  - oggetto singolo/oggetti replicati
- permette di specificare la strategia di replicazione
- permette di specificare strategie di riconnessione
- di fatto esiste solo l'implementazione per la chiamata unicast a oggetto singolo transitorio o persistente
- completa lo stack e permette di estenderlo senza rendere inutilizzabili le applicazioni correnti

Trasporto

- gestisce la connessione allo spazio di indirizzi remoto
  - establishment
  - monitoring
  - tear down

Separazione interfaccia/implementazione

- Uso di un'interfaccia per definire quali servizi l'oggetto deve fornire

```
public interface Honker extends java.rmi.Remote
{
    public void honk() throws
       java.rmi.RemoteException;
    }
```

Separazione interfaccia/implementazione (Cont)

```
public class HonkerImpl
        extends java.rmi.server.UnicastRemoteObject
        implements Honker
{
        int counter = 0;
        public void Honk() throws java.rmi.RemoteException
        {
                System.out.println("HONK!!!");
                counter++;
        }

        public int howManyTimes()
        {
                return counter;
        }
}
```

Garbage Collection Distribuita

- Java non ha deallocazione esplicita, quindi si rende necessaria
- Il server tiene conto dei riferimenti remoti
- Se si perde l'ultimo riferimento remoto viene inviato un messaggio unreferenced
- La possibilita' di problemi a livello di trasporto la rende inaffidabile

Compilatore di classi remote

- rmic produce la stub (oggetto proxy locale) e, se necessario/richiesto, skeleton
- se si usa l'opzione -v1.1 vengono generati stub e skeleton compatibili solo con JDK 1.x
- se si usa l'opzione -vcompat vengono generati stub e skeleton compatibili con JDK 1.x e Java2
- se si usa l'opzione -v1.2 viene generato solo lo stub compatibile con Java2 e non con JDK 1.x

Naming

- RMI puo' usare diversi servizi di Naming
  - es. Java Naming and Directory Interface (JNDI)
- SDK fornisce un servizio di default
  - rmiregistry
- spartano
  - non fornisce servizi sofisticati
  - locale
- semplice
- sufficiente per far funzionare il sistema

Registro RMI

- rmiregistry e' un servizio di naming non persistente
- si accede ad esso tramite metodi statici della classe java.rmi.Naming

  bind(nome, oggetto)

  rebind(nome, oggetto)

  lookup(nome)

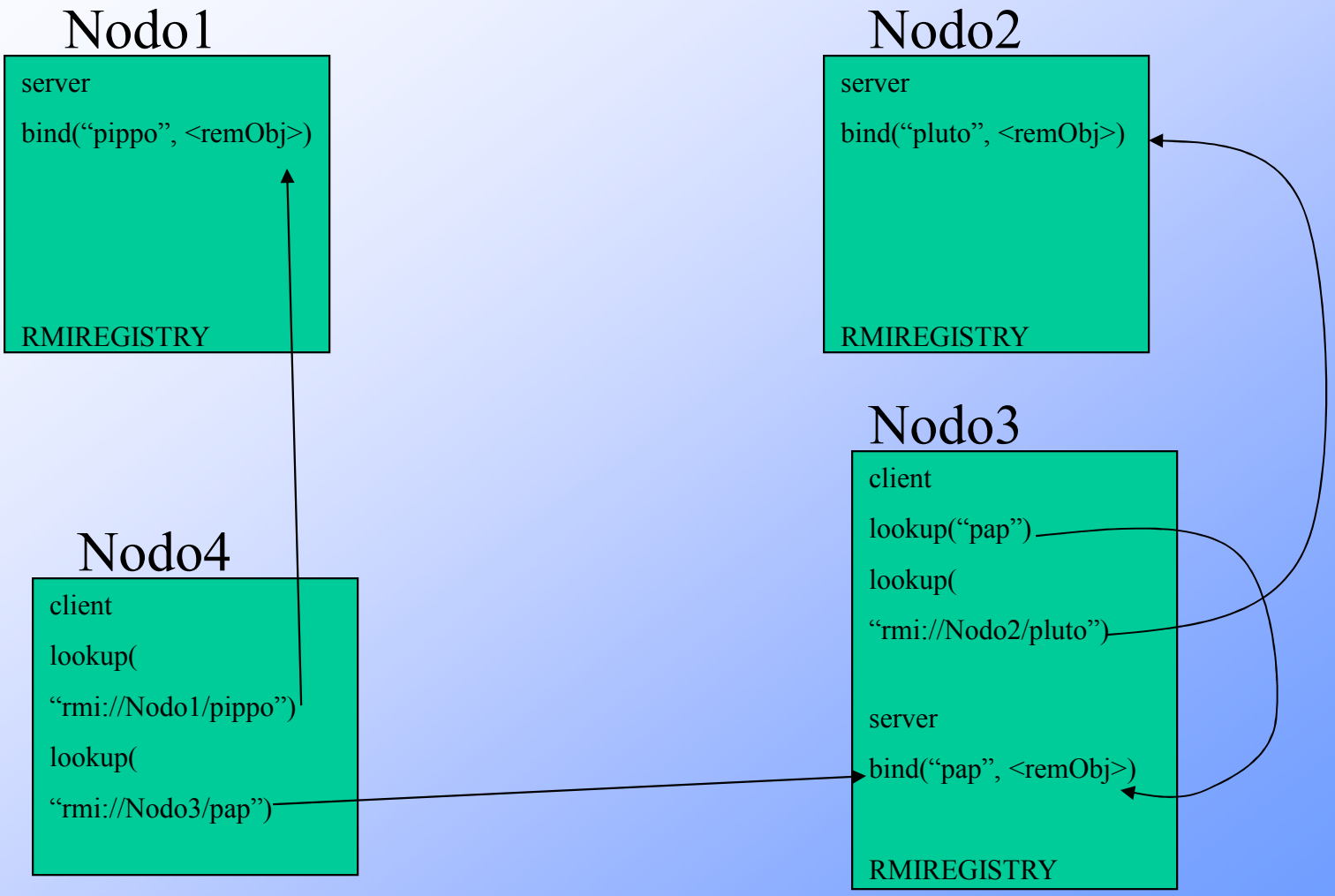
  unbind(nome)

  list(URL del registry)

rmiregistry

- Ogni nodo che ospita un server RMI DEVE avere un'istanza attiva di rmiregistry
- bind e rebind accettano un nome ed eseguono il bind presso il registry locale
- lookup accetta un URL nel formato
  - rmi://host:port/boundName
- E' quindi possibile effettuare il lookup su un registry remoto
- Non e' possibile effettuare il bind su un registry remoto
  - Eccezione di controllo degli accessi

Esempio


Nodo1
server
bind(“pippo”, <remObj>)
RMIREGISTRY
Nodo2
server
bind(“pluto”, <remObj>)
RMIREGISTRY
Nodo3
client
lookup(“pap”)
lookup(
“rmi://Nodo2/pluto”)
server
bind(“pap”, <remObj>)
RMIREGISTRY
Nodo4
client
lookup(
“rmi://Nodo1/pippo”)
lookup(
“rmi://Nodo3/pap”)

Caricamento delle classi in ambiente RMI

- Lato server
  - Interfacce remote che definiscono il servizio
  - Classi implementanti l'interfaccia remota del servizio
  - classi skeleton (solo Java 1.x)
  - classi stub
  - altre classi del server

Caricamento delle classi in ambiente RMI (Cont.)

- Lato client
  - Interfacce remote che definiscono il servizio
  - classi stub
  - classi dei parametri
  - altre classi del lato client

Caricamento da siti remoti

- Esiste RMIClassLoader che estende il classLoader di default
- e' in grado di caricare classi da server ftp e http
- usa la proprieta' codebase per identificare il server da contattare
- altre proprieta' per definire se e come contattare server remoti
  - permettono di ottenere diverse configurazioni

Gestione delle eccezioni remote

- Ogni metodo remoto deve essere dichiarato capace di generare l'eccezione java.rmi.RemoteException
- L'eccezione java.rmi.RemoteException e' un contenitore per qualsiasi eccezione generata dal server
- Di fatto la capacita' del cliente di recuperare i problemi che hanno generato un'eccezione remota e' (quasi sempre) limitatissima

Oggetti transeunti

- Esistono solo fintanto che e' attivo il processo che li ha istanziati
- java.rmi.server.UnicastRemoteObject
  - colla tra classe del server e livello di rete
  - si occupa di attivare il livello di rete del server (si registra con lo RMI server locale)

Oggetti persistenti

- Esistono in modo indipendente dal processo che li ha istanziati
- java.rmi.activation.Activatable
- usa un oggetto transitorio per “attivare” su richiesta i server
- RMID
  - implementazione di riferimento delle interfacce di sistema

Concetto di attivazione

- Classe astratta Activatable estende java.rmi.server.RemoteServer
  - come UnicastRemoteObject
- Permette di definire
  - ActivationID
    - identifica l'oggetto nelle sue diverse incarnazioni (attivazioni)
  - ActivationGroup
    - la JVM che esegue il codice dell'oggetto
  - I dati di attivazione dell'oggetto
    - inizializzazione