NSY102
Conception de logiciels Intranet
Remote Method Invocation

Cnam Paris
jean-michel Douin, douin au cnam point fr
15 Mars 2016

Notes de cours consacrées à RMI

Principale **bibliographie**

- RMI

  https://docs.oracle.com/javase/tutorial/rmi/

  http://today.java.net/pub/a/today/2004/06/01/RMI.html?page=1RMI

  http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html

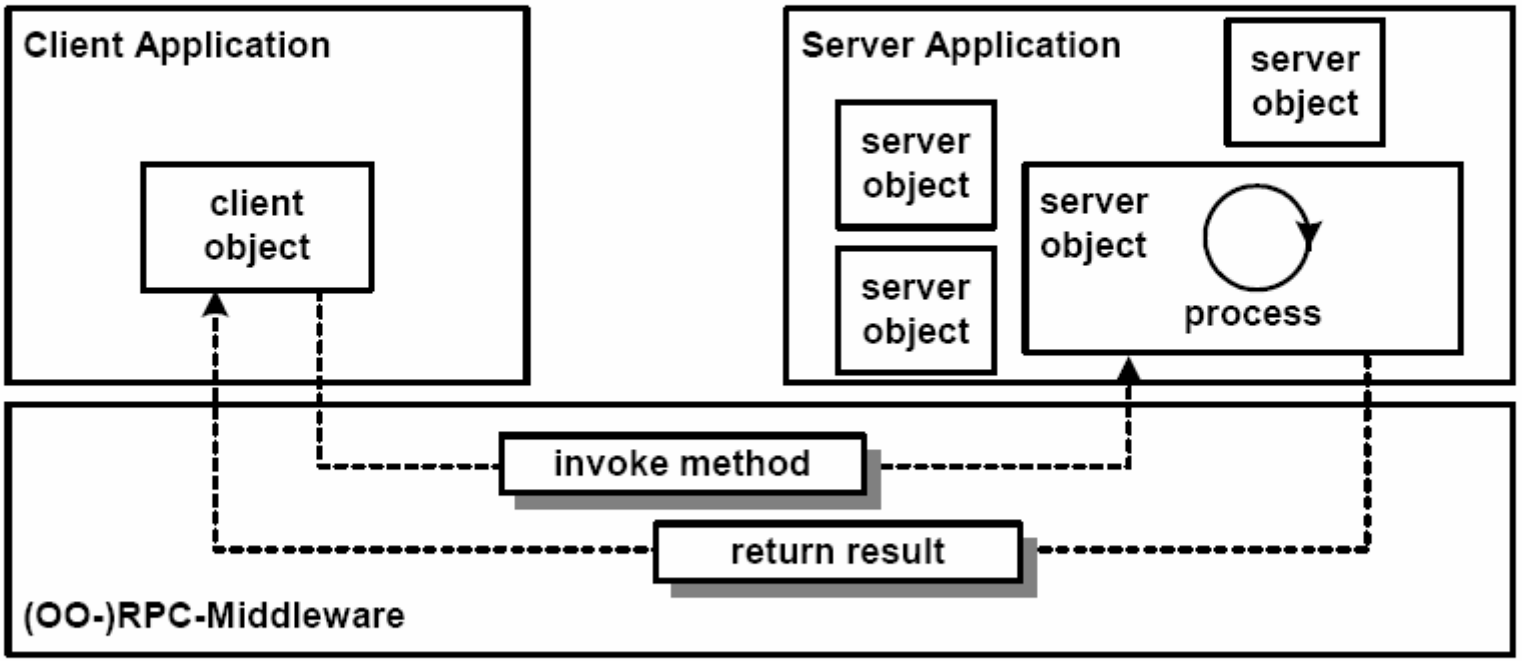
  http://java.sun.com/docs/books/tutorial/rmi/index.html

- Pattern : Proxy, Adapter

  http://www.transvirtual.com/users/peter/patterns/overview.html

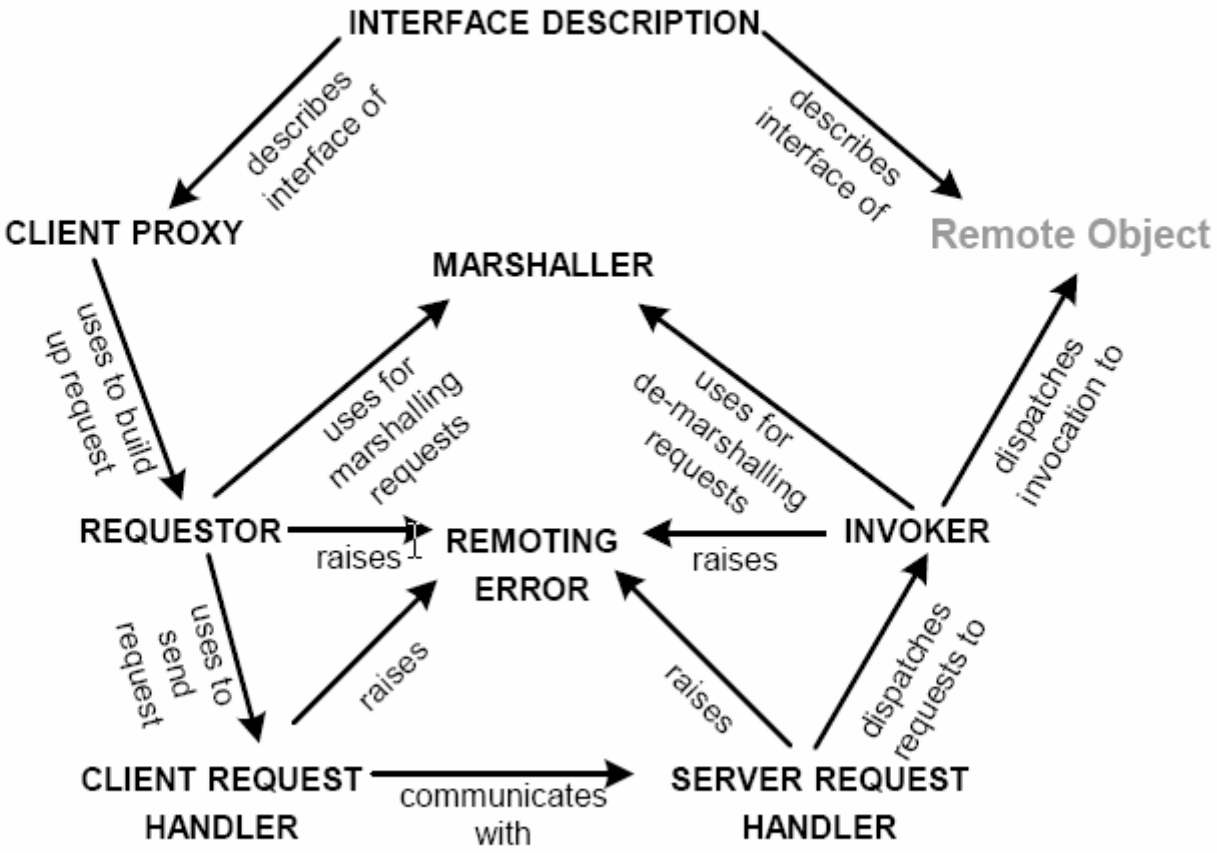
  http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html

Remoting Style: RPC



Patrons de Base

Basic Remoting Patterns



Différentes étapes

- **Côté serveur**
  1. **Création de l'annuaire,** *un service de nommage des services*
  2. **Création du service,** *enregistrement de celui-ci auprès de l'annuaire*
  3. **Un bail de ce service,** *convenu entre l'annuaire et le service effectif*
  4. **Le service reste actif,** *attend les requêtes des clients*
     - ***Une variante comme le chargement du service à la demande existe, la persistance peut être également mise en œuvre***

- **Côté client**
  1. **Interrogation de l'annuaire,** *à propos du service*
     - **L'annuaire se trouve sur la même machine que les services**
  2. **En retour,** *réception du mandataire chargé de la communication*
     - *Un « dynamicProxy » est téléchargé*
  3. **Exécution distante du service**
     1. **Emballage des paramètres, sélection du serveur et de la méthode**
     2. **Déballage des résultats retournés,**
        - **Incluant une levée éventuelle d'une exception**

Java - RMI : les bases

- **TCP/IP**
- **Java uniquement : RMI/JRMP**
  - **Le protocole Java Remote Method Protocol JRMP**
  - **JVM (Java Virtual Machine) clients comme serveurs**
- **Ouverture aux autres**

avec RMI/IIOP-CORBA/IIOP

  - **Internet Inter-ORB Protocol/Common**
  - **Voir les outils du jdk : *idlj, tnameserv, orbd***

Sommaire : mise en oeuvre

- **Étape 1**
  - Par l'exemple : Un client et un serveur de méthodes
  - téléchargement : le rôle de **rmiregistry**
  - Le serveur et (**rmic** ou le Pattern Proxy ***soit DynamicProxy)***
- **Étape 2**
  - Passage de paramètres et retour de fonctions
  - Les exceptions
- **Étape 3**
  - Déploiement, téléchargement de code depuis un client ou depuis le serveur
  - Un serveur http : pour le téléchargement de classes
- **Vers une méthode de développement**
  - Méthode : Usage du Pattern Adapter
  - Exemple : Un client et un serveur de tâches
- **Critiques**
  - Depuis le JDK1.2 **rmid** et la classe java.rmi.activation.Activatable
- **Annexe : un « chat »**

Objectifs en images : deux machines


```
public class Client{
  ServeurDeMethodes s= ...

  s.methodeLointaine();

}
```

RMI

```
public class ServeurDeMethodes{

  void methodeLointaine(){
    System.out.println("hellooo");
  }

}
```

==> <Quoi, .

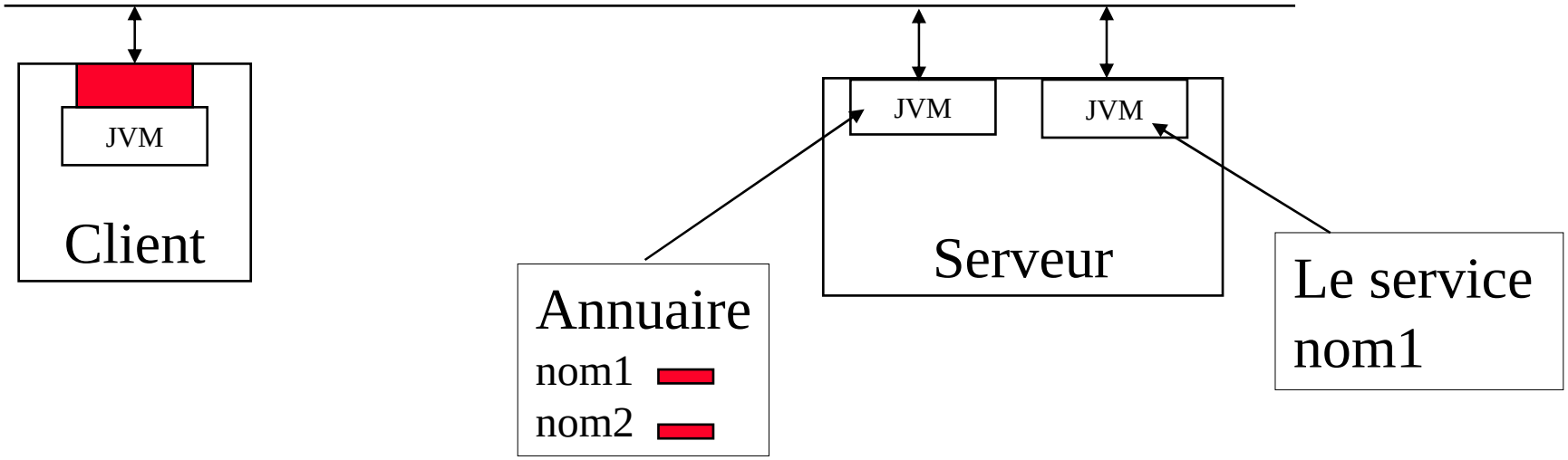
- *Où se trouve le serveur ?*
- *Comment déclenche t-on " void methodeLointaine()" ?*
- *Les accès sont-ils sécurisés ?*
- *Et les paramètres, les exceptions, les résultats ? ==> étape 2*

Comment>$_1$

- *Où se trouve le serveur ?*
- *Comment déclenche t-on " void methodeLointaine()" ?*
- *Les accès sont-ils sécurisés ?*

- La "machine Serveur" est identifiée par une adresse IP
  - un nom octroyé par l'administrateur du DNS
- La "classe ServeurDeMethodes" est associée à un nom répertorié
  - Un serveur de noms (un annuaire)
- La communication est prise en charge par un mandataire
  - mandataire fourni par le serveur au client , patron Client Proxy
- La classe serveur autorise les accès distants
  - précise les contraintes d 'accès aux fichiers en général un fichier ".**policy**"

« DynamicProxy » à la rescousse



- **Une instance du mandataire « qui s'occupe de tout » est téléchargée par le client**
  - **Cf. cours NSY102_Proxy**

Interface commune

interface Commune « AffichageLointain »

```
void methodeLointaine();
```

C'est le Langage commun

Développement en Java, principes

- **Une interface** précise les méthodes distantes,
  - Elle est commune au serveur et aux clients,
  - Elle hérite (au sens Java entre interfaces) de java.rmi.Remote
    - (un marqueur, `public interface Remote{}`)

Le serveur et ses clients

- **Le serveur** hérite* d'une classe prédéfinie
  package java.rmi.server et
  s'inscrit auprès d'un annuaire (gestionnaire de Noms/services)

- **Ses clients**
  recherchent le service proposé,
    en interrogeant l'annuaire
  obtiennent une référence de l'objet distant,
  effectuent les appels de méthodes,
    ces méthodes étant déclarées dans l'interface commune

** c'est une façon de faire, il y en a d'autres …*

Développement en Java : mode d'emploi

- **Un gestionnaire de noms/services (patron Registry)**
  - **rmiregistry est un outil pré-installé (inclus dans le jdk)**
    - **Annuaire, port 1099 par défaut**
    - **Une liste de couples <nom,service>**
  - **Ou bien usage de la classe Registry**
    - **Paquetage java.rmi.registry**

rmiregistry, l'annuaire par défaut (1099)

- **Table noms / services**
  - **String / Mandataire du service**
- **1) start rmiregistry, Côté serveur**
  - Enregistrement du service *(auprès de l'annuaire en 1099)*

    java.rmi.Naming.bind *(méthode de classe)*

    java.rmi.Naming.rebind
  - Obtention du service

    java.rmi.Naming.lookup

Registry

- **Table noms / services**
  - **String / Mandataire**
- **Création de l'annuaire sur le serveur**
  - **Registry registry = java.rmi.registry.LocateRegistry.createRegistry(1200);**
  - Enregistrement du service *(ici 1200)*

    registry.bind

    registry.rebind
  - Obtention du service

    registry.lookup

Développement en Java : mode d'emploi

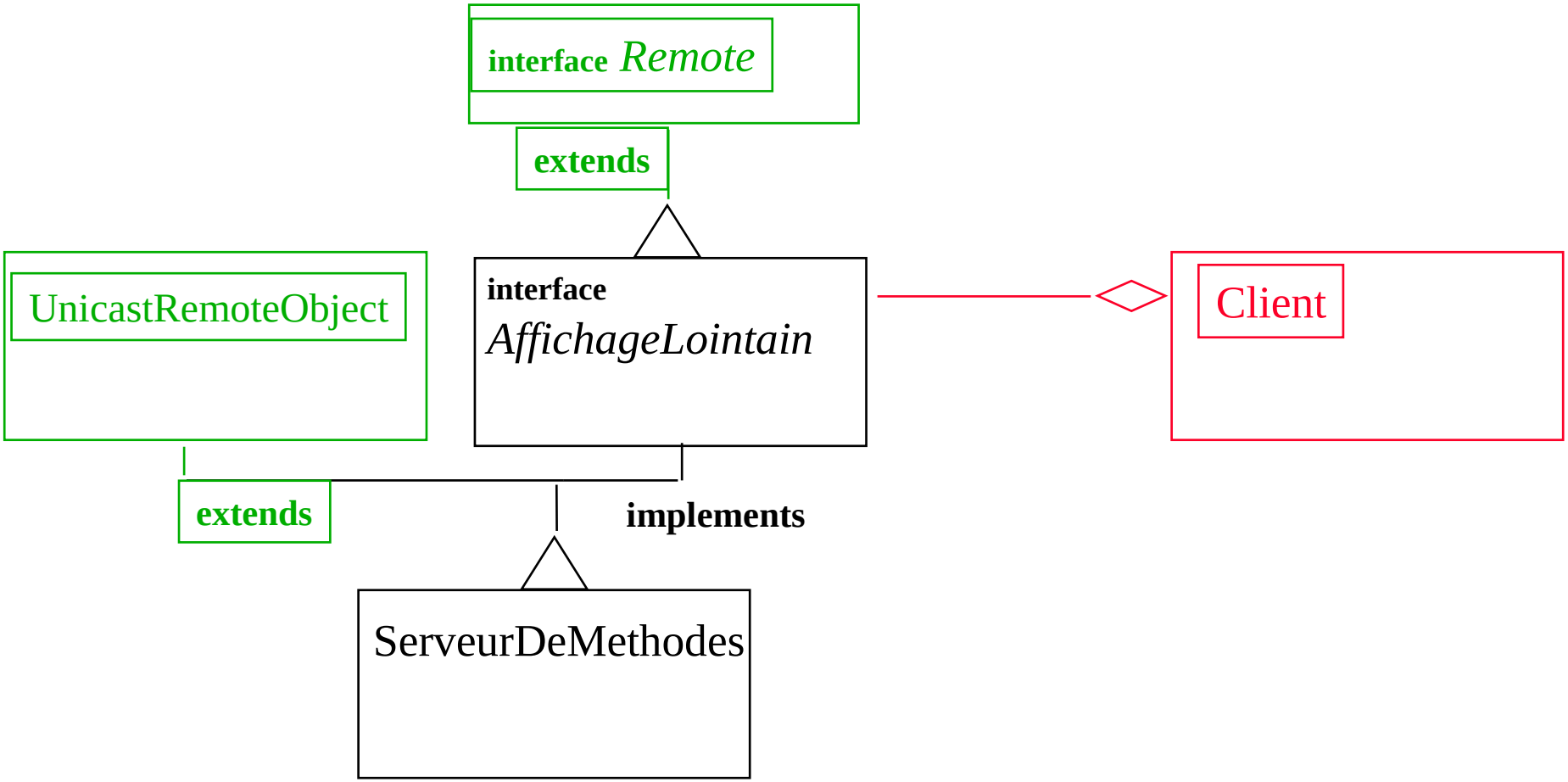
- **L ' interface est commune** aux clients et au serveur
  - hérite de ***java.rmi.Remote***
  - recense les méthodes distantes,
  - chaque méthode possède la clause ***throws java.rmi.RemoteException***

- **La classe « Serveur »**
  - hérite de ***java.rmi.server.UnicastRemoteObject***
  - implémente les méthodes de l 'interface commune,
    - sans oublier le constructeur qui possède également la clause *throws*.
  - Le serveur propose ses services
    - Voir ***java.rmi.Naming.rebind***

- **Les classes Clientes**
  - Utilisent une instance/référence de la classe « Serveur »
    - ***Voir java.rmi.Naming.lookup.***

- *Et c'est tout !!!*

Exemple : un serveur de méthodes



Le serveur de méthodes

un client

Exemple : Interface commune aux clients et au serveur

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

public interface AffichageLointain extends Remote{

  public void methodeLointaine() throws RemoteException;

  public static final String nomDuService = "leServeurDeMethodes";
}
```

L'interface commune doit :

- hériter de l'interface **`java.rmi.Remote`**
- pour chaque méthode ajouter la clause **`throws java.rmi.RemoteException`**
- être publique

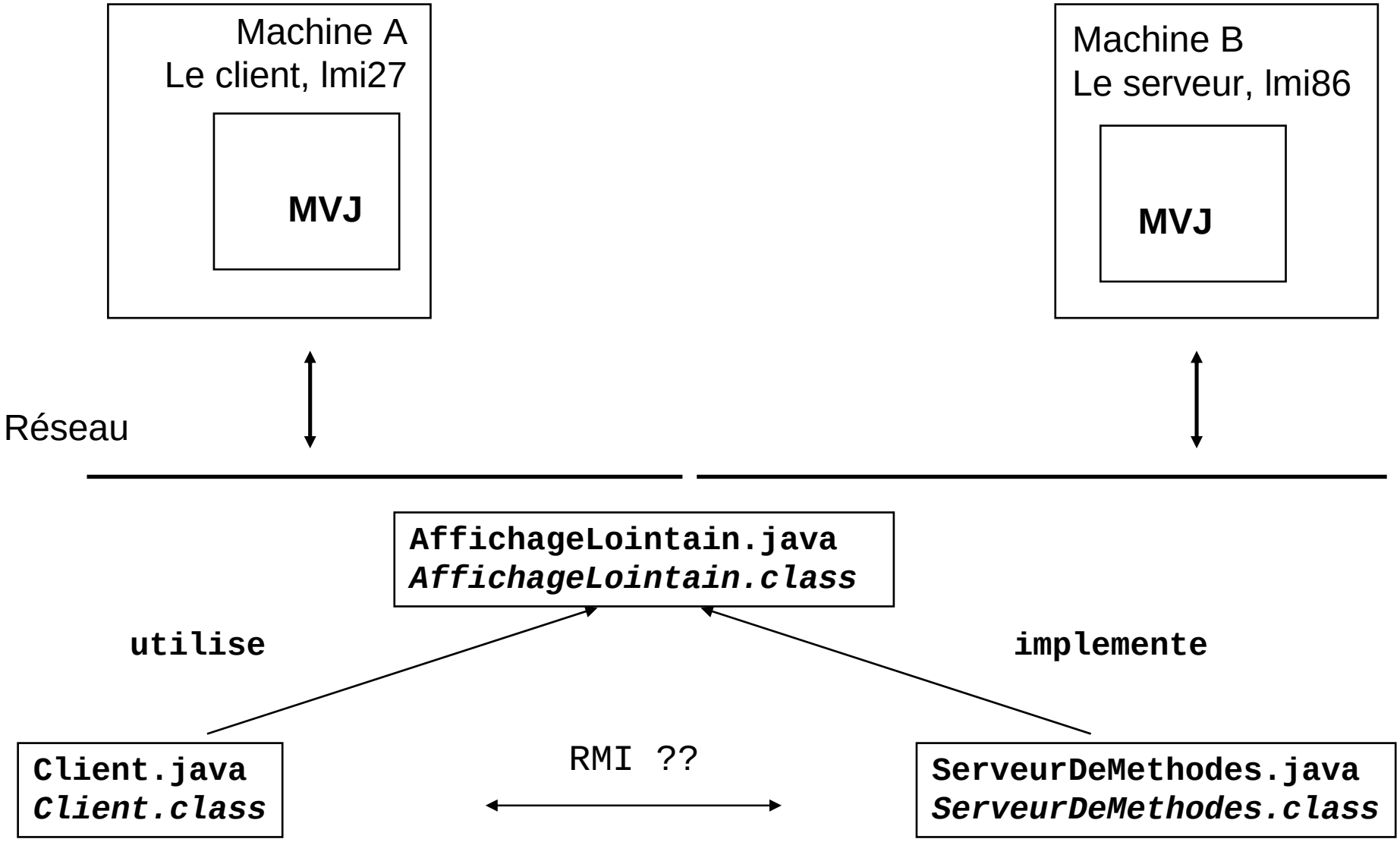
Exemple : Le serveur (machine lmi86)

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class ServeurDeMethodes    extends UnicastRemoteObject
                                  implements AffichageLointain{

  public void methodeLointaine() throws RemoteException{
    System.out.println("helloooo");
  }
  public ServeurDeMethodes () throws RemoteException{}

  public static void main(String[] args) throws Exception{
    System.setSecurityManager(new RMISecurityManager());
     try{
       AffichageLointain serveur = new ServeurDeMethodes();
       Naming.rebind(AffichageLointain.nomDuService, serveur);
       System.out.println("Le serveur lointain est pret");
    }catch(Exception e){throw e;}
  }}
```

Exemple : Le client (machine lmi27)

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

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

   AffichageLointain serveur = null;
    String nomComplet;

    nomComplet =
      "rmi://lmi86.cnam.fr:1099/" + AffichageLointain.nomDuService;
    try{
      serveur = (AffichageLointain) Naming.lookup(nomComplet);
      serveur.methodeLointaine();
    }catch(Exception e){ throw e;}
 }}
```

Architecture



Comment ? : les commandes, l'interface

- L’interface est partagée ou recopiée
  - ***Compilation de l'interface commune aux clients et au serveur***
  - > **javac** AffichageLointain.java
  - *Le fichier AffichageLointain.class est partagé entre **lmi86** et **lmi27** ou*
  - *recopié sur ces deux machines dans les répertoires*
    - *Par exemple*

      d:/rmi/serveurDeMethodes/serveur/

      d:/rmi/serveurDeMethodes/client/

Comment ?

- Où se trouve le code nécessaire ?
  - Le classpath pour rmi & rmiregistry, voir RMIClassLoader

  Ou bien
  - -Djava.rmi.server.**codebase=une URL file:// http:// ftp://**
    - *format du paramètre codebase : protocol://host[:port]/file (séparés par un blanc)*
    - *-Djava.rmi.server.useCodebaseOnly=true protocol ::= ftp | http |file*

    Un serveur web est installé sur le port 8086
    - -Djava.rmi.server.**codebase=http://localhost:8086/**
- Quelle stratégie de sécurité ?
  - Djava.security.policy=java.policy

  - Un exemple de stratégie des plus laxiste, le fichier java.policy

```
grant{
   permission java.security.AllPermission;
};
```

Comment ? : les commandes, le serveur

- **//lmi86/d:/rmi/serveurDeMethodes/serveur/**
  - ***Compilation du serveur***
  - > **javac** ServeurDeMethodes.java
  - > obsolète **rmic** ServeurDeMethodes (inutile si >=1.5, un proxy est généré dynamiquement)

  - ***Exécution***
  - > **start** rmiregistry ***-J-Djava.rmi.server.useCodebaseOnly=false***
  -
  - > **java** -Djava.rmi.server.**codebase**=file:/D:/rmi/ServeurDeMethodes/Serveur/
    -Djava.security.policy=java.policy ServeurDeMethodes

  - *Pour plus d'informations sur la console associée à rmiregistry :*
  - > **java** -Djava.rmi.server.**codebase**=file:/D:/rmi/ServeurDeMethodes/Serveur/
  - -Djava.rmi.server.logCalls=true
  - -Djava.security.policy=java.policy ServeurDeMethodes

  - voir java.rmi.server.RMIClassLoader

(rmic inutile si >=1.5)

- **rmic** : **rmi c**ompiler, génération des fichiers _Skel.class et _Stub.class (inutile si jdk > 1.5)
- **rmiregistry** : association nom et référence Java, port 1099 envoi de _Stub aux clients
- **java** -Djava.rmi.server.codebase=file:/D:/rmi/ServeurDeMethodes/Serveur/ *: accès au _stub.class*
- -Djava.security.policy=java.policy *: en 2.0, contraintes d 'accès*

Traces du Comment (>= jdk 1.7)

- **Traces avec un serveur http:**
  1) **>start rmiregistry –**J-D*java.rmi.server.useCodebaseOnly=false*
  2) **>start java ServeurWeb8086**

Coté serveur :

- **>** java -Djava.rmi.server.**codebase**=**http://localhost:8086/**
  -D*java.rmi.server.useCodebaseOnly=false*
  -Djava……..

- **La trace sur la console du serveur Web**
  - **Lors de l'inscription auprès de l'annuaire**

[LMI86/127.0.0.1] -- Request: GET /AffichageLointain.class HTTP/1.1

Comment ? : les commandes, le Client

- **//lmi27/d:/rmi/ServeurDeMethodes/Client/**
  - **> javac Client.java**
  - **> java -Djava.security.policy=java.policy Client**
  - *Le serveur de noms lmi86 est référencé dans le source du Client par :*
  - **…**

```
nomComplet ="rmi://lmi86/" + AffichageLointain.nomDuService;
try{
  serveur = (AffichageLointain) Naming.lookup(nomComplet);
```

rmic, _skel.clas(<jdk1.2)<, _stub.class(<jdk1.5)

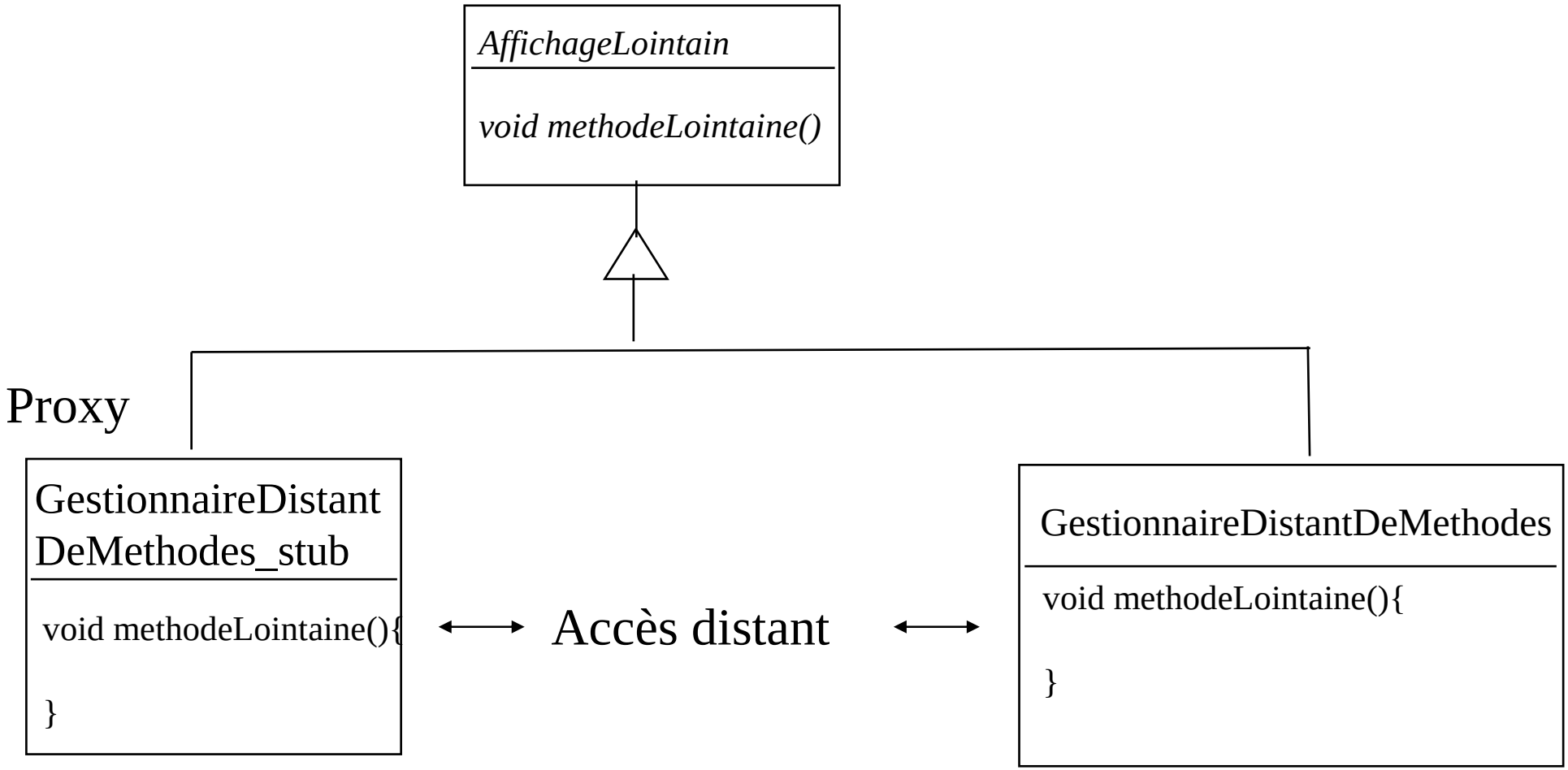
Communication et transmission des paramètres sur le réseau totalement transparents

_stub : interface réseau fournie au client

_skel : mis en forme des paramètres et résultats *(inutile en 1.2)*

_stub.class ou dynamic proxy si 1.5

Le pattern Client Proxy



En savoir un peu plus rmic -keep

- **rmic –keep ServeurDeMethodes**
  - **ServeurDeMethodes_stub.java**
  - **et**
  - **ServeurDeMethodes_skel.java**
- **rmic –keep –v1.2 ServeurDeMethodes**
  - **ServeurDeMethodes_stub.java**

rmiregistry

- **port 1099 par défaut**
  - **>start rmiregistry 1999**
  - *alors nomComplet ="rmi://lmi86:1999/" + ...*
- transfert du fichier _stub aux clients
- **Naming.rebind()** pour le serveur
- **Naming.lookup()** pour les clients

Note :

Si cet utilitaire en fonction de la variable CLASSPATH a accès aux fichiers _stub, la commande -Djava.rmi.server.codebase= xxxxx est ignorée

Voir java.rmi.registry.LocateRegistry

java.security.policy=java.policy

le fichier java.policy en exemples

```
grant{
  permission java.security.AllPermission;
};
```

ou bien *(mieux)*

```
grant{
  permission java.net.SocketPermission "localhost",
   "connect,accept,listen";
  permission java.net.SocketPermission "lmi86.cnam.fr:8080-8089",
   "connect,accept";
};
```

- http://java.sun.com/products/jdk/1.2/docs/guide/security/PolicyFiles.html
- http://java.sun.com/products/jdk/1.2/docs/guide/security/permissions.html

Les commandes en images …

```
D:\>java -version
java version "1.8.0_20"
Java(TM) SE Runtime Environment (build 1.8.0_20-b26)
Java HotSpot(TM) Client VM (build 25.20-b23, mixed mode)

D:\>start rmiregistry -J-Djava.rmi.server.useCodebaseOnly=false
```

```
D:\>cd temp

D:\temp>cd cours05_rmi

D:\temp\cours05_rmi>javac AffichageLointain.java

D:\temp\cours05_rmi>
```

```
D:\temp\cours05_rmi\client>copy ..\*.class .
..\AffichageLointain.class
        1 fichier(s) copié(s).

D:\temp\cours05_rmi\client>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\temp\cours05_rmi\client>
```

- **D:/temp/cours05_rmi**
- **D:/temp/cours05_rmi/client**
- **D:/temp/cours05_rmi/serveur**

Les commandes en images .. Suite et fin

```
D:\temp\cours05_rmi\serveur>copy ..\*.class .
..\AffichageLointain.class
        1 fichier(s) copié(s).

D:\temp\cours05_rmi\serveur>javac ServeurDeMethodes.java
Note: ServeurDeMethodes.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

D:\temp\cours05_rmi\serveur>

java -cp . -Djava.rmi.server.codebase=http://localhost:8086/serveur/

-D*java.rmi.server.useCodebaseOnly=false*

-Djava.security.policy=java.policy ServeurDeMethodes

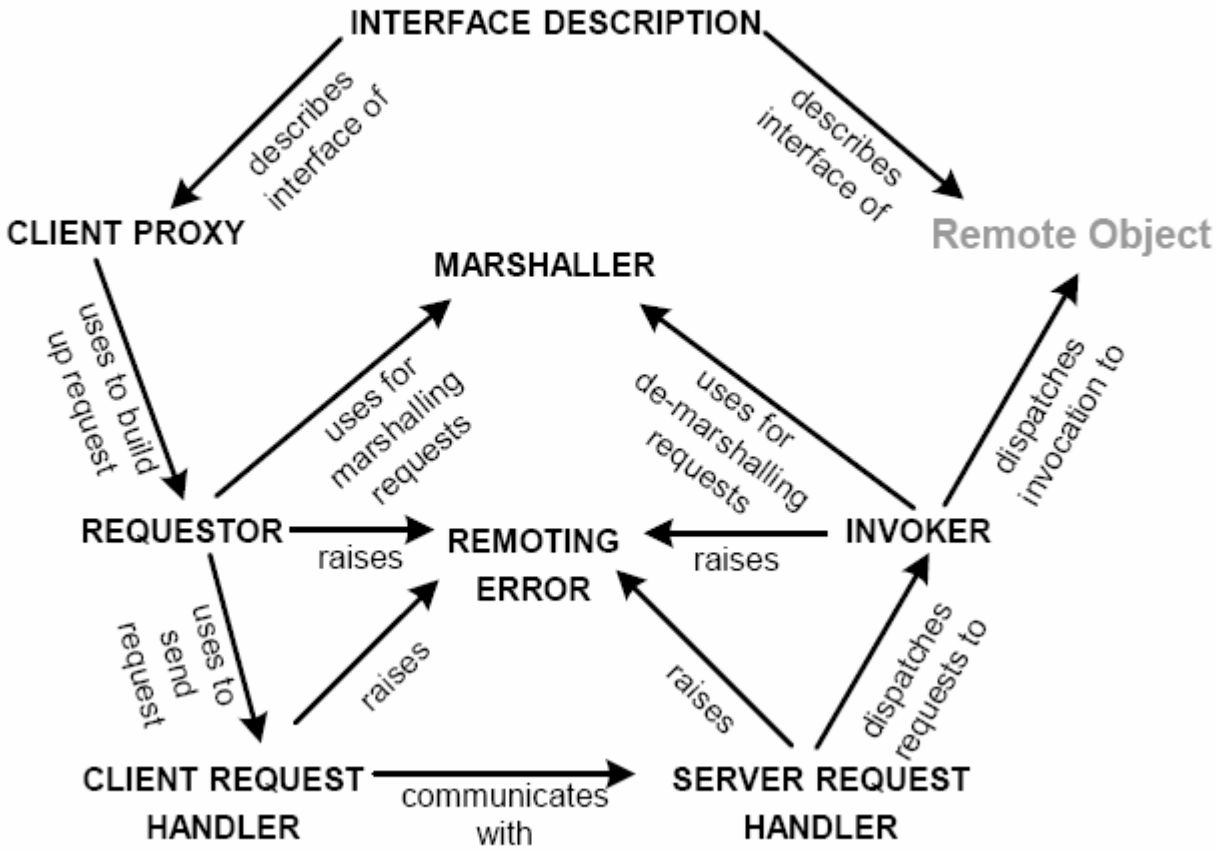
```
D:\temp\cours05_rmi\serveur>java -cp . -Dj
Le serveur lointain est pret
```

- **Démonstration**

- **Note: depuis JDK 7,** -D*java.rmi.server.useCodebaseOnly=false*
- **À lire: http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/enhancements-7.html**
-

Récapitulatif

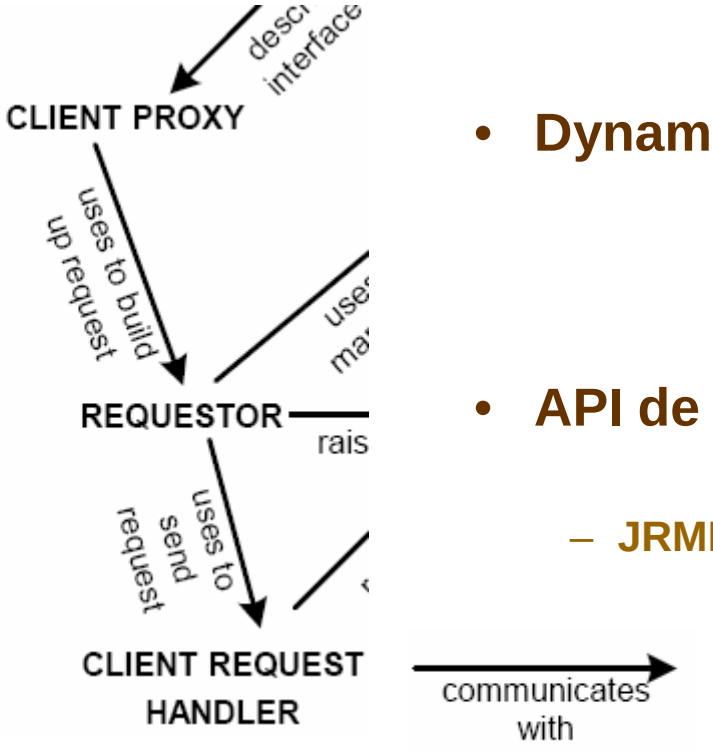
Basic Remoting Patterns



Interface commune en java

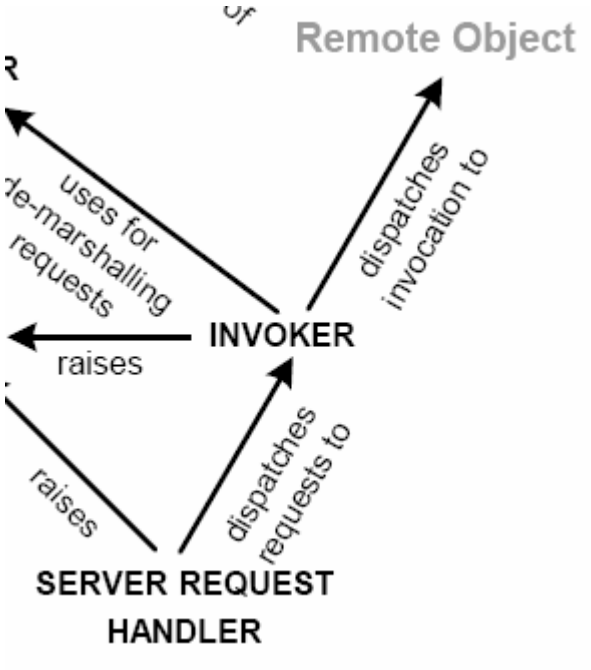
- **Interface commune ou langage commun**
  - **Entre les clients et le serveur**

Client Proxy



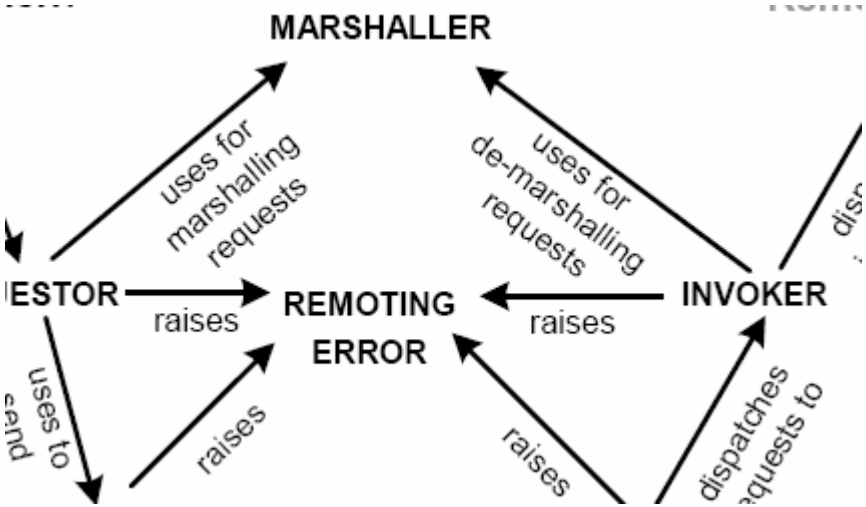
- **DynamicProxy**
- **API de java.net**
  - **JRMP & Socket**

Côté serveur



- **La classe « serveur »**
  - **extends UnicastRemoteObject**
- **API de java.net**
  - **JRMP & Socket**

Quid ?



- **Transmission des paramètres**
- **Levée d'exceptions**
  - **Marshaller comme Sérialisation …**

Etape 2 : Objectifs


```
public class Client{
  Calcul c = …

  res = c.moyenne(uneListe);

}
```

RMI

```
public class Calcul{

  Integer moyenne(List l){
    …

  }

}
```

==> <Quoi, .

- *Comment transmettre les paramètres ?*
- *Comment les résultats de fonction sont-ils retournés aux clients ?*
- *Une exception est levée côté serveur, comment informer le client ?*

Comment>2

- *Comment transmettre les paramètres ?*
- *Comment les résultats de fonction sont-ils retournés aux clients ?*
- *Une exception est levée côté serveur, comment informer le client ?*

- Aucune incidence sur le source Java
  - excepté que les paramètres transmis doivent être « des instances » de **java.io.Serializable**
- Une copie des paramètres en profondeur est effectuée,
  - Serializable à tous les niveaux
- En retour de fonction, une copie est également effectuée
  - Serializable
- Les Exceptions sont des objets comme les autres

Une interface marqueur

```
public interface java.io.Serializable {}
```

Sérialisation : principes (rappels ?)

- Le paramètre est une instance de **java.io.Serializable**

```
public class XXXX implements java.io.Serializable{...}
```

Opérations internes : **- écriture** par copie de l'instance en profondeur
- lecture de l'instance

- ***Ecriture*** *de l'instance : ce qui est transmis sur le réseau*

```
OutputStream out = ...
ObjectOutputStream o = new ObjectOutputStream( out);
o.writeObject(obj);
```

*Les données d'instance sont copiées sauf les champs "**static**" et "**transient**"*

- ***Lecture*** *de l'instance : ce qui est lu depuis le réseau*

```
InputStream in  = ...
ObjectInputStream i = new ObjectInputStream( in);
Object obj = i.readObject();
```

Démonstration

Chapitre 3 : Objectifs


```
public class Client{
  ServeurDeTaches s; ...
  Horloge uneHorloge = ...

  s.executer(uneHorloge);

}
```

RMI

```
public class ServeurDeTaches{

  void executer(Runnable r){
    new Thread(r).start();

  }

}
```

==> <Quoi, .

- *Et si les paramètres utilisent des classes inconnues du serveur ?*

Comment>2

Et si les paramètres utilisent des classes inconnues du serveur ?

- Le serveur les « demande » au client
  - -Djava.rmi.server.**codebase=une URL file:// http:// ftp://**
    - **Côté client**

Chargement du .class : Horloge.java

```
public class Horloge extends Thread
                     implements java.io.Serializable{

  public void run(){
    int sec=0,mn=0,h=0;

    while (true){
       try{
         Thread.sleep(1000);
         sec++;
         if(sec==59){
           mn++;sec =0;
           if(mn==59) { h++; mn=0; if (h==24) h=0;}
         }
        System.out.println(h + ":" + mn + ":" + sec);
       }catch(Exception e){}}}}
```

Le serveur demande "Horloge.class"

Horloge.class

```
public class ExecuteurDeTaches{

  void executer(Runnable r){
    new Thread(r).start();
  }

}
```

Comment est-il satisfait ?

- //lmi27/d:/rmi/ServeurDeMethodes/Client/
  - **>** java **-Djava.rmi.server.codebase=file:/D:/rmi/ServeurDeMethodes/Client/**
  - -Djava.security.policy=java.policy Client
  - ***ou avec l'aide d'un serveur http***
  - **>** start java **SimpleHttpd 8088**
  - **>** java **-Djava.rmi.server.codebase= http://lmi27:8088/D:/rmi/ServeurDeMethodes/Client/**
    -Djava.rmi.server.useCodebaseOnly=false
    -Djava.security.policy=java.policy Client
  - Sans oublier **`System.setSecurityManager(new RMISecurityManager());`**
    - Autorise le téléchargement de code ici depuis le client, qui lui seul possède les classes (.class) manquantes

format du paramètre codebase : protocol://host[:port]/file (séparés par un blanc)
-Djava.rmi.server.useCodebaseOnly=true (depuis jdk 1.7)

==> <Quoi, .

- *Et si les paramètres utilisent des classes inconnues du client ?*

Comment>$_2$

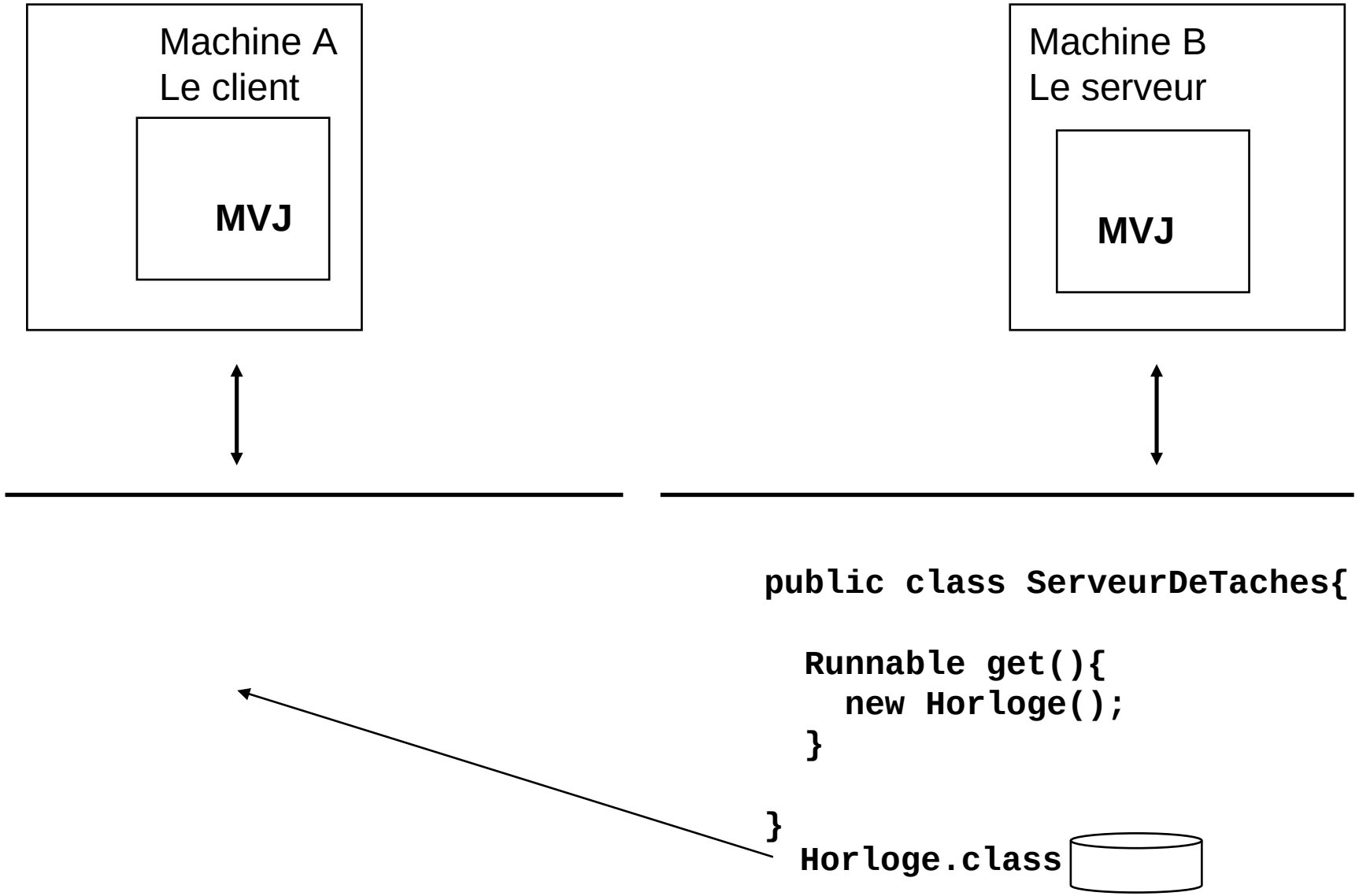
Et si les paramètres utilisent des classes inconnues du Client ?

- Le client les « demande » au serveur
  - -Djava.rmi.server.**codebase=une URL file:// http:// ftp://**
    - **Côté serveur**

Chargement du .class : Horloge.java

```
public class Horloge extends Thread
                     implements java.io.Serializable{

  public void run(){
    int sec=0,mn=0,h=0;

    while (true){
       try{
         Thread.sleep(1000);
         sec++;
         if(sec==59){
           mn++;sec =0;
           if(mn==59) { h++; mn=0; if (h==24) h=0;}
         }
        System.out.println(h + ":" + mn + ":" + sec);
       }catch(Exception e){}}}}
```

Le client demande "Horloge.class"



Comment est-il satisfait ?

- //lmi27/d:/rmi/ServeurDeMethodes/Serveur/

  – **>** start java **SimpleHttpd 8088**

  – **>** java **-Djava.rmi.server.codebase= "http://lmi27:8088/D:/rmi/ServeurDeMethodes/ http://lmi27:8088/D:/rmi/ServeurDeMethodes/serveur/"**

  -Djava.rmi.server.useCodebaseOnly=false

  -Djava.security.policy=java.policy ServeurDeTaches

  – Sans oublier **`System.setSecurityManager(new RMISecurityManager());`**

    - Autorise le téléchargement de code ici depuis le client, qui lui seul possède les classes (.class) manquantes

format du paramètre codebase : protocol://host[:port]/file (séparés par un blanc)

-Djava.rmi.server.useCodebaseOnly=true

(comme son nom l'indique, inhibe les accès de rmiregistry)

Conclusion intermédiaire

- **1.5, mise en œuvre facilitée,** par un proxy
  - **JavaRmiRuntime au lieu de l ’étape rmic**
  - le proxy est généré soit **en héritant** de la classe **UnicastRemoteObject**
  -
```
public class GroupeDeDsicussion
  extends UnicastRemoteObject implements Remote{ ...
```
  - soit en « exportant dynamiquement » le service

```
IndividuImpl lambda = new IndividuImpl(args[0]);
Remote stub = UnicastRemoteObject.exportObject(lambda, 0);
```

- Recherce du service, soit en utilisant la classe java.rmi.Naming
  - soit directement

```
Registry registry = LocateRegistry.getRegistry();
registry.rebind(args[0], stub);
```

Conclusion intermédiaire

- **Reprendre les exemples de l'annexe 3**

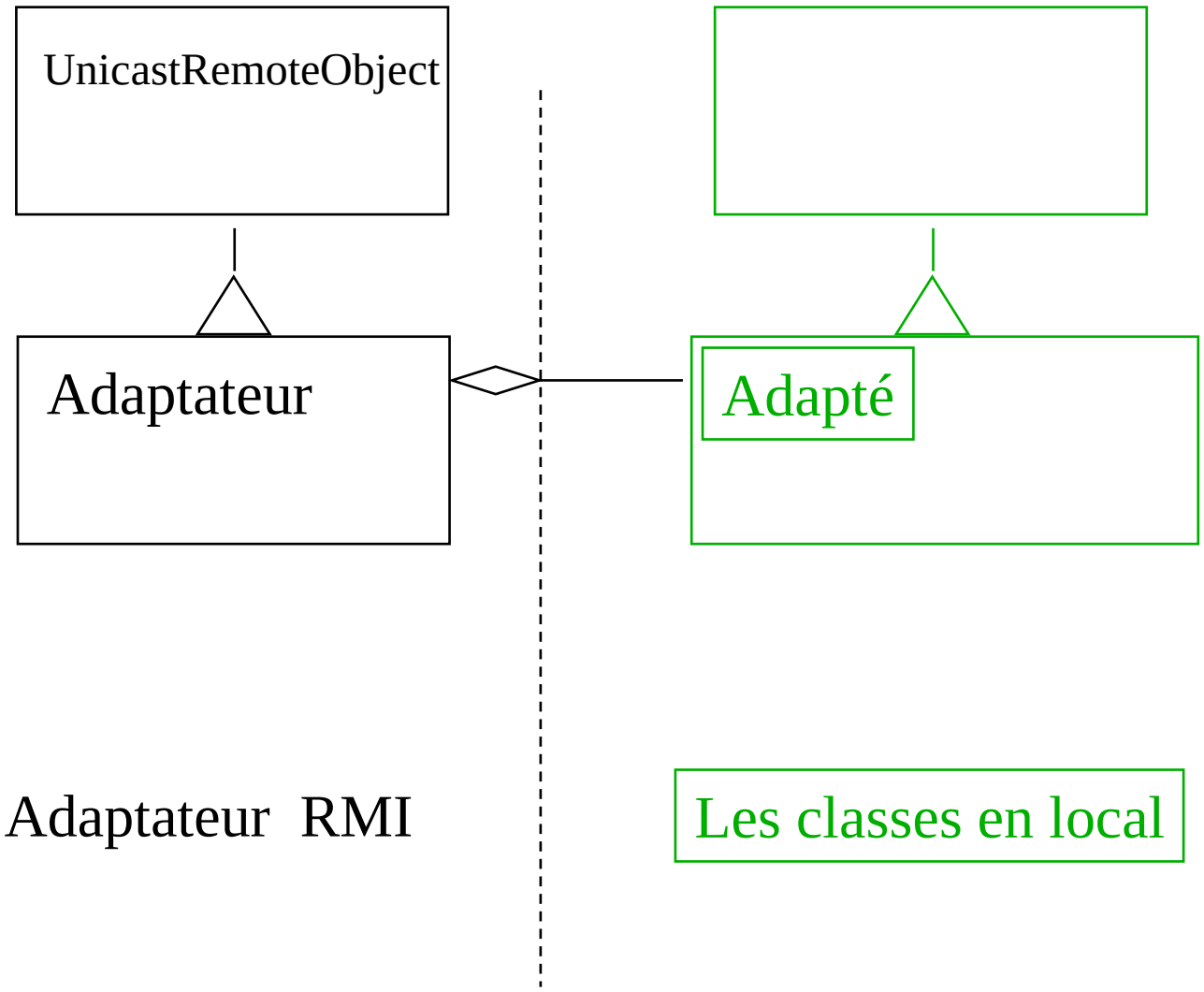
- **test1**
  - **Le premier exemple de ce support, appel d'une méthode sans paramètre**
- **test2**
  - **test1 + appel d'une méthode avec paramètre et retour d'un résultat**
- **test3**
  - **test2 + levée d'une exception côté serveur**
- **test4**
  - **les clients souhaitent exécuter des « thread » côté serveur,**
  - **le serveur demande si nécessaire, aux clients les classes manquantes**
- **test5**
  - **les serveur délivre un « thread » à exécuter côté client,**
  - **le client demande si nécessaire, au serveur les classes manquantes**
- **test6**
  - **test3 sans hériter de UnicastRemoteObject**
- **test7**
  - **test3 sans utiliser rmiregistry**

Développement : une ligne de conduite

- **Peut-on effectuer un développement en local et mettre en œuvre simplement le mécanisme RMI ?**
- **Comment assurer un faible couplage des classes assurant le RMI et les autres classes ?**
- *Peut-on oublier RMI ?*
- **Une solution : usage du Pattern Adapter**
  - **Un adaptateur RMI/version locale**

`http://www.transvirtual.com/users/peter/patterns/overview.html`
`http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html`

En 2 étapes


UnicastRemoteObject
Adaptateur
Adapté
Adaptateur RMI
Les classes en local

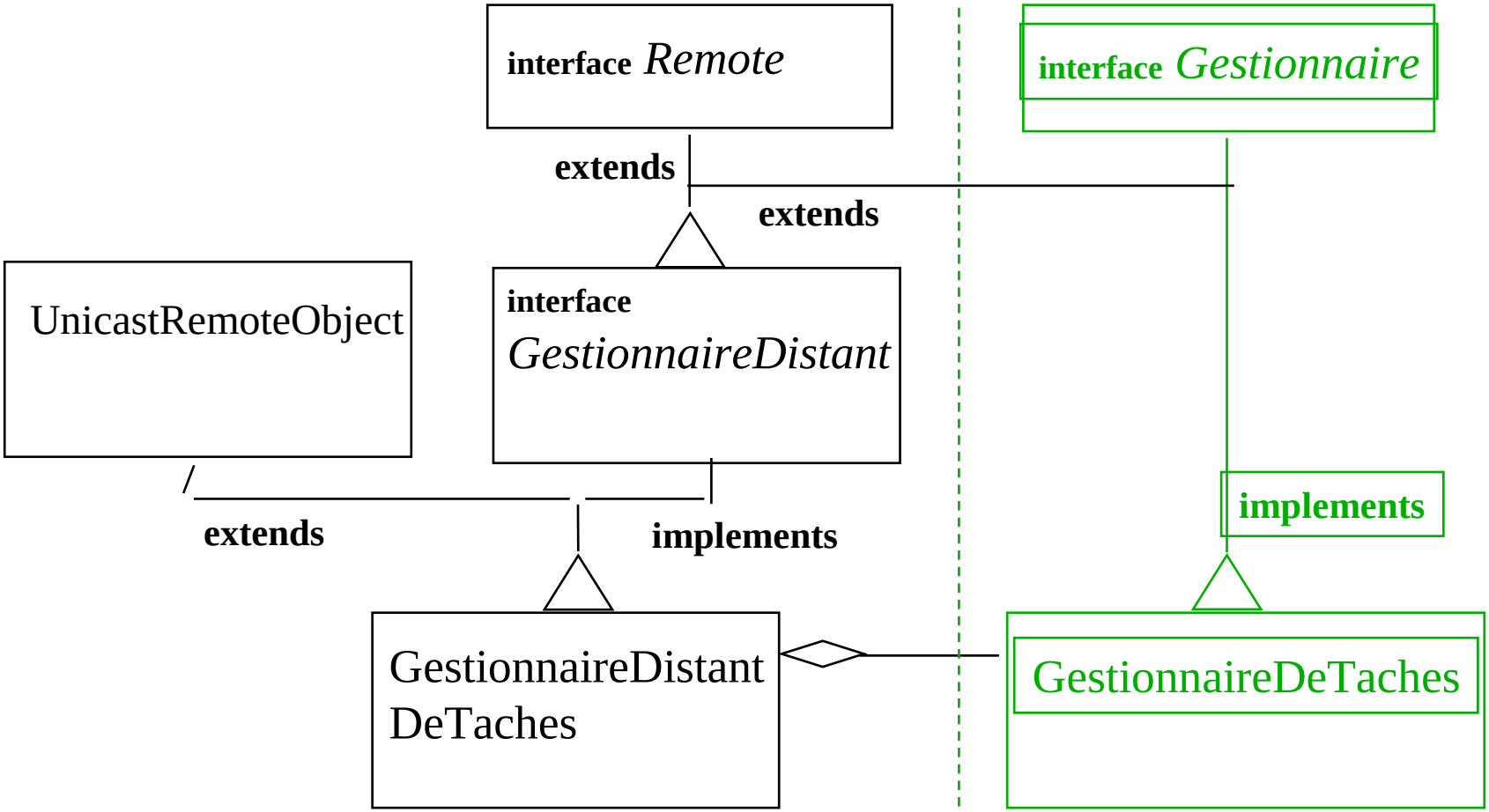
Exemple le serveur de tâches


```
public class Client{
  ServeurDeTaches s; ...
  Horloge uneHorloge = ...

  s.executer(uneHorloge);

}
```

RMI

```
public class ServeurDeTaches{

  void executer(Runnable r){
    new Thread(r).start();

  }

}
```

Le serveur de tâches : diagrammes UML



Adaptateur RMI

Les classes en local : l'adapté

local : GestionnaireDeTaches (Adapté)

```
public interface Gestionnaire{

  public void executer( Runnable r) throws Exception;

  public java.util.Vector liste() throws Exception;
}
```

Légère contrainte syntaxique :

Chaque méthode possède la clause throws Exception

local : GestionnaireDeTaches (Adapté)

```
public class GestionnaireDeTaches implements Gestionnaire{
   private ThreadGroup table;
   private Vector      liste;

   public GestionnaireDeTaches(String nom){
     table = new ThreadGroup(nom);
     liste = new Vector();
   }

   public void executer( Runnable r) throws Exception{
     Thread t = new Thread(table,r); t.start();
     liste.addElement(r);
   }
   public java.util.Vector liste() throws Exception {
     return liste;
   }
 }
```

local : Test de l 'adapté

```
public class TestGestionnaireDeTaches{

  public static void main(String [] args) throws Exception{

    Gestionnaire mon1 = new GestionnaireDeTaches("mon1");

    mon1.executer(new Horloge());
    mon1.executer(new Horloge());

    System.out.println("mon1 : " + mon1.liste());

    mon1.executer(new Horloge());
    System.out.println("mon1 : " + mon1.liste());

  }
}
```

Comme toujours :

Effectuer avec soins « tous » les tests du futur adapté

GestionnaireDistant.java (Adaptateur)

- **Développement de l ’adaptateur**

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface GestionnaireDistant extends Remote,Gestionnaire{

  public static final String nomDuService = "leServeurDeTaches";

  public void executer( Runnable r) throws RemoteException,Exception;

  public java.util.Vector liste() throws RemoteException,Exception;

}
```

La Cohérence des 2 interfaces est assurée par :

GestionnaireDistant extends **Remote,Gestionnaire**

GestionnaireDistantDeTaches.java (Adaptateur)

```
public class GestionnaireDistantDeTaches extends UnicastRemoteObject
                                         implements GestionnaireDistant{

  private Gestionnaire gestionnaire;  // l'adapté


  public GestionnaireDistantDeTaches(Gestionnaire gestionnaire)
                                                 throws RemoteException{
     this.gestionnaire = gestionnaire;
  }

  public void executer( Runnable r) throws RemoteException,Exception{
     gestionnaire.executer(r);
  }

  public Vector liste()  throws RemoteException,Exception{
     return gestionnaire.liste();
  }
}
```

Systématique :
Chaque méthode se contente d'exécuter l'adapté

ServeurDeTaches.java

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

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


    try{
      GestionnaireDistant mon1 = new GestionnaireDistantDeTaches(
                                      new GestionnaireDeTaches("mon1"));



      Naming.rebind( GestionnaireDistant.nomDuService, mon1);
      System.out.println("le serveur: " +
                         GestionnaireDistant.nomDuService +
                         " a demarre ");

     }catch(Exception e){throw e;}
   }
}
```

Le Client : TacheCliente.java

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

   public static void main(String [] args) throws Exception{
     String machine = "lmi27";
     if (args.length == 1) machine = args[0];
     System.setSecurityManager( new RMISecurityManager());
      GestionnaireDistant mon1 = null;

     String nom = "rmi://"+ machine +"/"+ GestionnaireDistant.nomDuService;

    try{
       mon1 = (GestionnaireDistant)Naming.lookup(nom);
    }catch(Exception e){throw e;}


   try{
     mon1.executer(new Horloge());
     mon1.executer(new Horloge()); System.out.println(mon1.liste());

     mon1.executer(new Horloge()); System.out.println(mon1.liste());
    }catch(Exception e){throw e;}}}
```

Comment ? : les commandes, le serveur

- **//lmi86/d:/rmi/ServeurDeTaches/Serveur/**

  - > set CLASSPATH= ...
  - > start **rmiregistry**

  - Ou bien à l'aide d'un serveur au protocole http (sur la même machine)

  - **java** -Djava.rmi.server.codebase=http://localhost:8086/
  - -Djava.security.policy=java.policy ServeurDeTaches

Comment ? : les commandes, le client

- **//lmi27/d:/rmi/ServeurDeTaches/Client/**
  - **> start java SimpleHttpd 8088**
  - **> java -Djava.rmi.server.codebase=http://lmi27:8088/D:/rmi/ServeurDeTaches/Client/**
  - **-Djava.security.policy=java.policy TacheCliente**

- **codebase = {http://machine/repertoire/ | ftp:/repertoire/ | file:/}**

Activatable, Objectifs

- Activation des objets distants à la demande des clients sur le serveur
- L ’utilitaire jdk **rmid** engendre des instances à la demande
- En conséquence la classe ServeurDeTaches n’est plus résidente

java.rmi.Activatable

- **import** java.rmi.activation.Activatable;
- Il faut (entre autres)
  - installer **rmid**, sur le serveur (par défaut le port 1098)
  - ***> start rmid***
  - ***en Java***
  - indiquer le répertoire dans lequel se trouve les classes du serveur
  - soit ***file:/d:/jfd/rmi/ServeurDeTaches_Activatable/Serveur/***
  - identifier le nom de la classe qui est chargée dynamiquement
  - soit ***"GestionnaireDistantDeTaches"***
  - ajouter un constructeur d'arité 2 en respectant cette signature
  - soit *public* ***GestionnaireDistantDeTaches(ActivationID id, MarshalledObject data){***
  - *super(id,0); ...*
  - *}*
  - transmettre éventuellement des paramètres d'initialisation, à l 'enregistrement du service
  - soit *m = new* ***MarshalledObject(parametre)***

Exemple : le gestionnaire de taches

- La classe GestionnaireDistantDeTaches
  - Elle doit hériter de la classe **Activatable** et ajouter le constructeur d'arité 2
  - *public* ***GestionnaireDistantDeTaches(ActivationID id, MarshalledObject data){..}***
  - Cette classe sera chargée dynamiquement
  - Usage en interne de *getClass* et *newInstance*
  - Le constructeur d'arité 2 sera exécuté par **rmid**
  - Des paramètres peuvent être transmis par l 'intermédiaire d'une instance de la classe java.rmi.MarshalledObject
  - MarshalledObject(Object parametre) extends Object
  - Le paramètre est sérialisé *(soit une instance de java.io.Serializable)*
  - La méthode ***get*** effectue une copie du paramètre et celui-ci est ensuite dé-sérialisé
  - les autres méthodes sont : ***equals*** et ***hashcode***
  -

extends java.rmi.activation.Activatable

```
import java.rmi.*;
import java.rmi.activation.*;
public class GestionnaireDistantDeTaches extends Activatable
                                         implements GestionnaireDistant{

  private GestionnaireDeTaches gestionnaire;

  public GestionnaireDistantDeTaches(ActivationID id, MarshalledObject m)
     throws RemoteException, java.io.IOException, ClassNotFoundException{

     super(id,0);
     this.gestionnaire = (GestionnaireDeTaches) m.get();
  }

  public void executer( Runnable r) throws RemoteException,Exception{
    gestionnaire.executer(r);
  }
  public java.util.Vector liste()  throws RemoteException,Exception{
    return gestionnaire.liste();
  }}
```

La classe GestionnaireDeTaches

- **L ’adapté s ’est adapté**

- *this.gestionnaire = (GestionnaireDeTaches) m.get();*

```
import java.io.Serializable;
public class GestionnaireDeTaches  implements Gestionnaire,
                                               Serializable{

  private transient ThreadGroup table;
  private Vector      liste;

....
```

ServeurDeTaches$_1$

```
import java.rmi.*;
import java.rmi.activation.*;
import java.util.Properties;

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

   // référence sur "java.policy" , les contraintes de sécurité doivent
   // doit être fournies à l 'ActivationGroup, ici par l 'intermédiaire
   // d'une instance de Properties (qui hérite de HashTable)

    Properties props = new Properties();
    props.put("java.security.policy", "d:/jfd/rmi/java.policy");

     ActivationGroupDesc.CommandEnvironment ace = null;
     ActivationGroupDesc exampleGroup = new ActivationGroupDesc(props,ace);
```

ServeurDeTaches$_2$

```
    // Obtention de son ID, (port)
    ActivationGroupID agi;
    agi = ActivationGroup.getSystem().registerGroup(exampleGroup);
    // Création
    ActivationGroup.createGroup(agi, exampleGroup, 0);


    // Création d 'une instance de ActivationDesc, cette instance
    // fournit toutes les informations à rmid

String location;
// le répertoire dans lequel se trouve la classe gérant les objets distants
location  = "file:/d:/jfd/rmi/ServeurDeTaches_Activatable/Serveur/";
// le paramètre d 'initialisation ici l'adapté adapté
MarshalledObject m = new MarshalledObject(new GestionnaireDeTaches("mon1"));

// Création
ActivationDesc desc;
desc = new ActivationDesc ("GestionnaireDistantDeTaches",location,m);
```

ServeurDeTaches$_3$

```
// enregistrement du service
try{

  GestionnaireDistant mon1 =(GestionnaireDistant)Activatable.register(desc);

// au lieu de
// GestionnaireDistant mon1 = new GestionnaireDistantDeTaches(
//                                      new GestionnaireDeTaches("mon1"));


  Naming.rebind( GestionnaireDistant.nomDuService, mon1);
  System.out.println("le serveur : " + GestionnaireDistant.nomDuService +
                     "  est enregistre ");
  }catch(Exception e){throw e;}}
  System.exit(0);
}

// le programme se termine
    //( une fenêtre liée à rmid mentionne les appels distants)
```

Les commandes, client et serveur

- ***Compilation*** : javac et rmic GestionnaireDistantDeTaches

- ***Exécution du serveur***

```
set CLASSPATH=
start rmiregistry
start rmid
set CLASSPATH=d:/jfd/rmi/ServeurDeTaches_Activatable/Serveur/
java
  -Djava.rmi.server.codebase=file:/d:/jfd/rmi/ServeurDeTaches_Activatable/
 Serveur/ -Djava.security.policy=java.policy ServeurDeTaches
```

- ***Exécution du client***

```
start java SimpleHttpd 8088
java
  -Djava.rmi.server.codebase=lmi27:8088/d:/jfd/rmi/ServeurDeTaches_
Activatable/Client/ -Djava.security.policy=java.policy TacheCliente
```

Activatable et plus

- **La classe peut ne pas hériter de Activatable**
  - `Activatable.exportObject(this, id, 0);`
  - `sera présente dans le corps du constructeur d'arité 2 de la classe distante`
  - *public* ***ClasseDistante(ActivationID id, MarshalledObject data){..}***

- **On peut se servir de l 'instance de MarshalledObject afin de rendre certaines données du serveur persistantes**
  - **auprès de rmid**
  - **m = new MarshalledObject(new File("d:/rmi/persistent.ser"));**
  - **et au sein du constructeur**
  - **f =(File) m.get();**
  - `voir http://java.sun.com/products/jdk/1.2/docs/guide/rmi/activation.html`
  -

Conclusion

- **RMI: Remote Method Innovation ?**
  - **passage de paramètres**
  - **transmission de code**
  - **Bail et ramasse miettes distribué**
- **RMI 1.2, Activatable**
- **J2SE >=1.5 : encore plus de transparence**
- **Suite logique ?**
  - **JINI**
    - **« Plug and work »**
    - **En désuétude …**

Conclusion

- **Finir les exemples de l'annexe 3**

- **test8**
  - **test3, le service est chargé à la demande, le serveur hérite de Activatable**
- **test9**
  - **test8, le serveur n'hérite pas de Activatable**
- **test10**
  - **test9, avec la persistance du service**

Annexe0: JNDI et « rmiregistry »

JNDI, Java Naming and Directory Interface *(http://java.sun.com/products/jndi/tutorial/index.html)*

Service de nommage ou de répertoire de manière uniforme

Indépendant de toute implémentation

opérations comme bind, unbind, rebind, lookup, list

```
// avec rmiregistry
import java.rmi.Naming;
…
 Remote stub = (Remote) Naming.lookup("rmi://server/nomDeService");

// avec JNDI,
// http://java.sun.com/j2se/1.5.0/docs/guide/jndi/jndi-rmi.html
import javax.naming.*;
…
Hashtable<String,String> env = new Hashtable<String,String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.sun.jndi.rmi.registry.RegistryContextFactory" );
env.put(Context.PROVIDER_URL,"rmi://localhost:1099");
InitialContext ctxt = new InitialContext(env);
Remote stub = (Remote) ctxt.lookup("nomDeService");
```

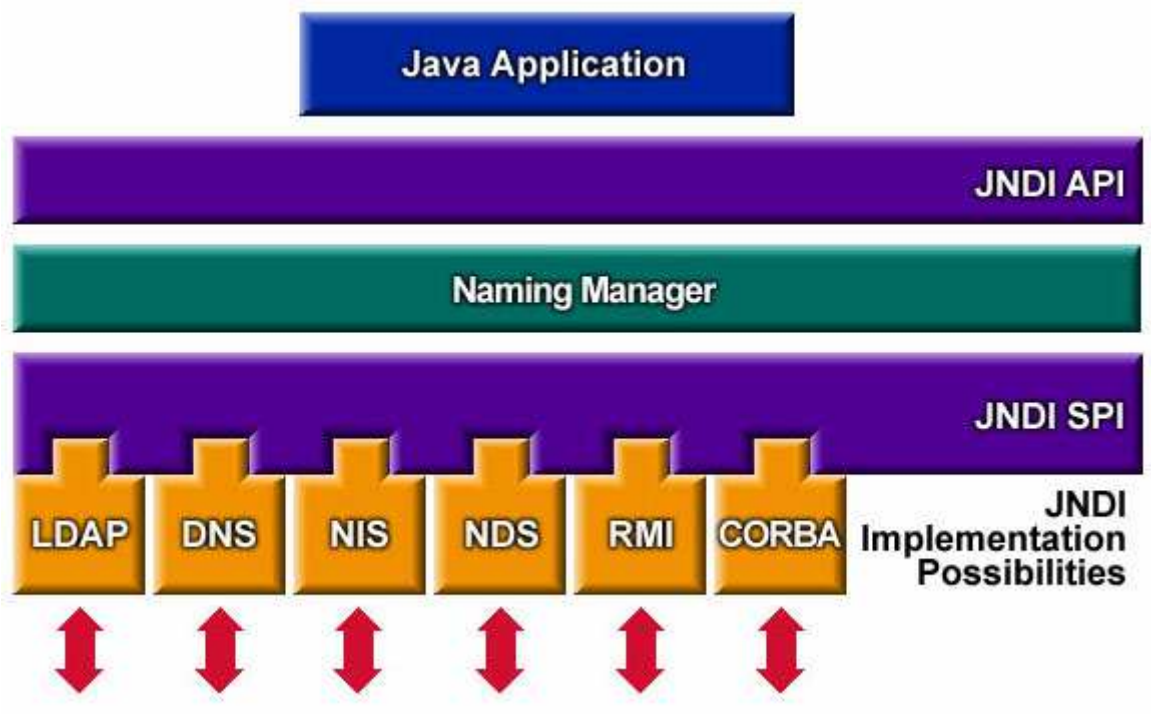
Annexe1 : interrogation de « rmiregistry »

```
import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RMISecurityManager;

public class ListRegistry{

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

    try{
      String[] list = Naming.list(args[0]);
      for(int i =0; i<list.length;i++){

        Remote remote = (Remote) Naming.lookup(list[i]);
        System.out.println(list[i] + ", remote: " + remote);
      }
    }catch(Exception e){e.printStackTrace(); throw e;
    }
  }
}
```

Annexe1 : interrogation de « rmiregistry » via JNDI

```
import javax.naming.*;
import …;
public class ListRegistryViaJNDI{

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

    Hashtable<String,String> env = new Hashtable<String,String>();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
  "com.sun.jndi.rmi.registry.RegistryContextFactory");
    env.put(Context.PROVIDER_URL,"rmi://localhost:1099");
    InitialContext ctxt = new InitialContext(env);

    try{
      NamingEnumeration<NameClassPair> ne = ctxt.list("");
      while(ne.hasMore()){
        NameClassPair ncp = ne.next();
        Remote remote = (Remote) ctxt.lookup(ncp.getName());
        System.out.println(ncp.getName()+ ", remote: " + remote);
      }
    }catch(Exception e){e.printStackTrace(); //throw e;
    }}}
```

Annexe1 JNDI : providers



- **LDAP, DNS, COS et RMI registry présents avec le J2SE**
- **Les autres ici**
  - **http://java.sun.com/products/jndi/downloads/index.html**

Annexe2: « bootstrap » du client

http://www.aw.com/cseng/titles/0-20170043-3

```
public interface Bootstrap extends Remote{
  Runnable getClient() throws RemoteException;
}


public class RMIClientBootstrap{
  private final static String server="rmi://localhost/boot";

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

    Bootstrap bootstrp = (Bootstrap)Naming.lookup(server); // (args[0])
    Runnable client = bootstrap.getClient();
    Thread t = new Thread(client); // ou client.run();
    t.start();
   }
}
```

téléchargement et exécution … (mobilité du code)

Annexe3 : 10 scenarii au complet

- **Téléchargez cette archive :**
  - **http://jfod.cnam.fr/NSY102/rmi/exemples_cours_rmi.jar**
    - **Un répertoire cours_rmi a été créé :**
      - **./**
        |____test1/
        |____test2/
        |____…/
        |____.../
        |____test10/

Placez-vous dans le répertoire créé et exécutez cette commande,
jar xvf exemples_cours_rmi.jar

si besoin recompilez par la commande compil, sous unix remplacez le ; par le :

1. **>runOnce**

démarre l'annuaire (en 1099) et un serveur web (en 8086)
vérifiez avec la commande > jps (3 JVM)

test1..test7 sommaire

- **test1**
  - Le premier exemple de ce support, appel d'une méthode sans paramètre
- **test2**
  - test1 + appel d'une méthode avec paramètre et retour d'un résultat
- **test3**
  - test2 + levée d'une exception côté serveur
- **test4**
  - les clients souhaitent exécuter des « thread » côté serveur,
  - le serveur demande si nécessaire, aux clients les classes manquantes
- **test5**
  - les serveur délivre un « thread » à exécuter côté client,
  - le client demande si nécessaire, au serveur les classes manquantes
- **test6**
  - test3 sans hériter de UnicastRemoteObject
- **test7**
  - test3 sans utiliser rmiregistry

test8..test10 sommaire

- **test8**
  - **test3, le service est chargé à la demande, le serveur hérite de Activatable**
- **test9**
  - **test8, le serveur n'hérite pas de Activatable**
- **test10**
  - **test9, avec la persistance du service**

testX

- **Commandes à exécuter dans une console**
- **Pour chaque scenario,**
  - **ce sont les mêmes commandes à enchaîner**

1. **Placez-vous dans le répertoire testX**
   - *si besoin* **testX> compil** *préalable*
2. **Placez-vous dans le répertoire serveur**
   - **testX/serveur> run** *si besoin* **testX/serveur> compil** *préalable*
3. **Placez-vous dans le répertoire client**
   - **testX/client> run** *si besoin* **testX/client> compil** *préalable*

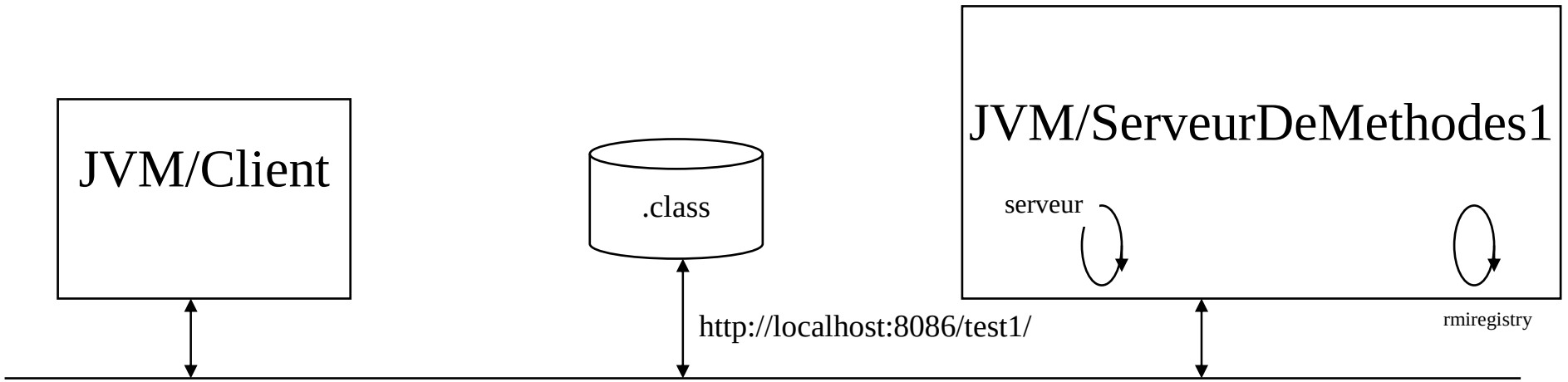
- **Vérifiez avec les traces obtenues, essayez sur deux machines**
  - **Il suffit de remplacer localhost par l'adresse du serveur**

Remplacer les ; par des : dans chaque .bat si vous êtes sous unix
start est l'équivalent sous windows du « & » sous unix

test1

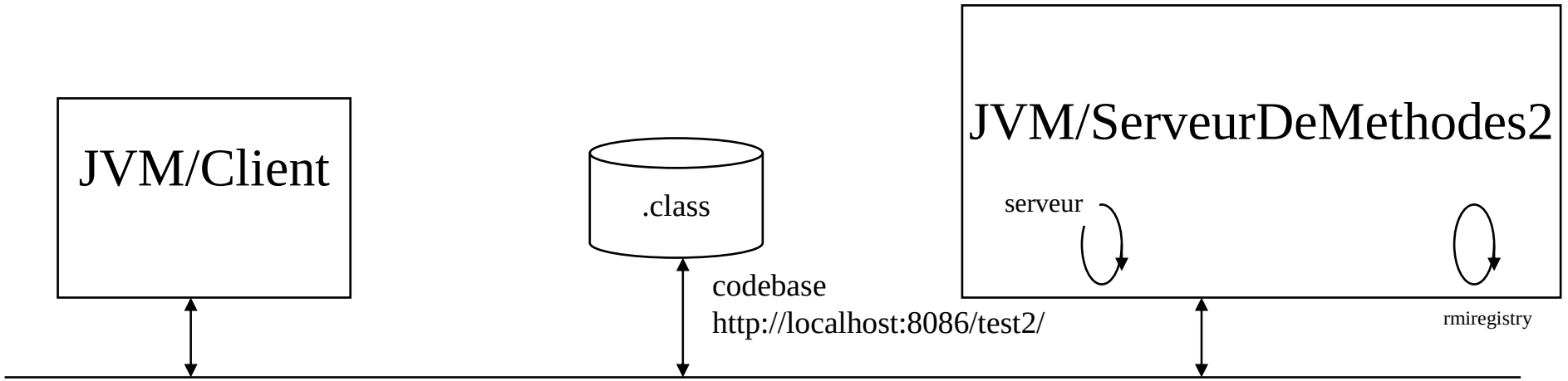
- **Le premier exemple de ce support**
  - **sur une même machine (localhost)**

  – **Un service rmi se contente d'afficher hello côté serveur**
    - **Appel distant de serveur.methodeLointaine()**



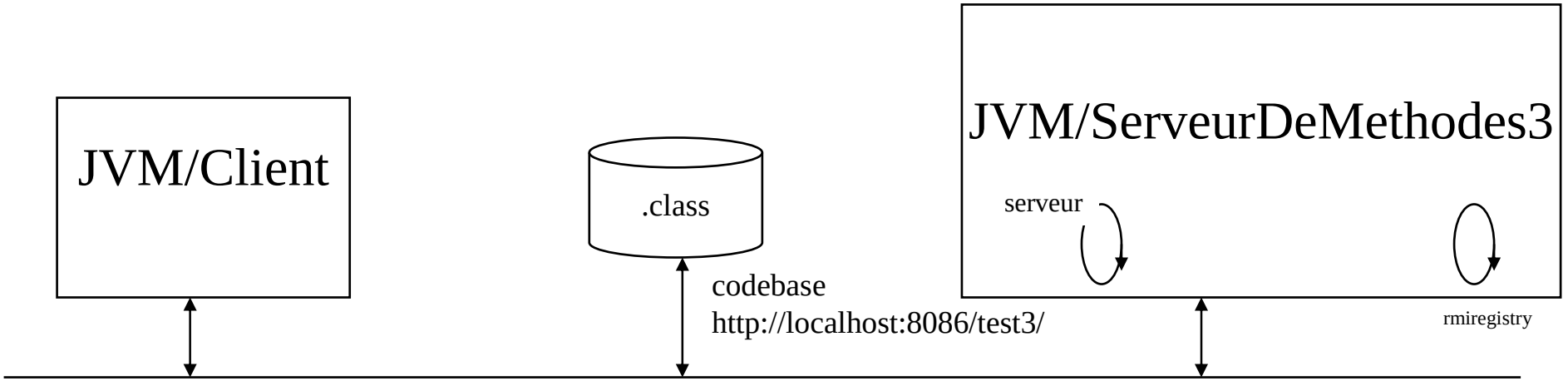
test2

- **L'exemple précédent**
  - **avec des paramètres et un résultat retourné**
  - **Le service rmi se contente**
    - **d'afficher la liste transmise côté serveur, le client affiche le résultat retourné**
    - **Appel distant de serveur.methodeLointaine(une liste)**



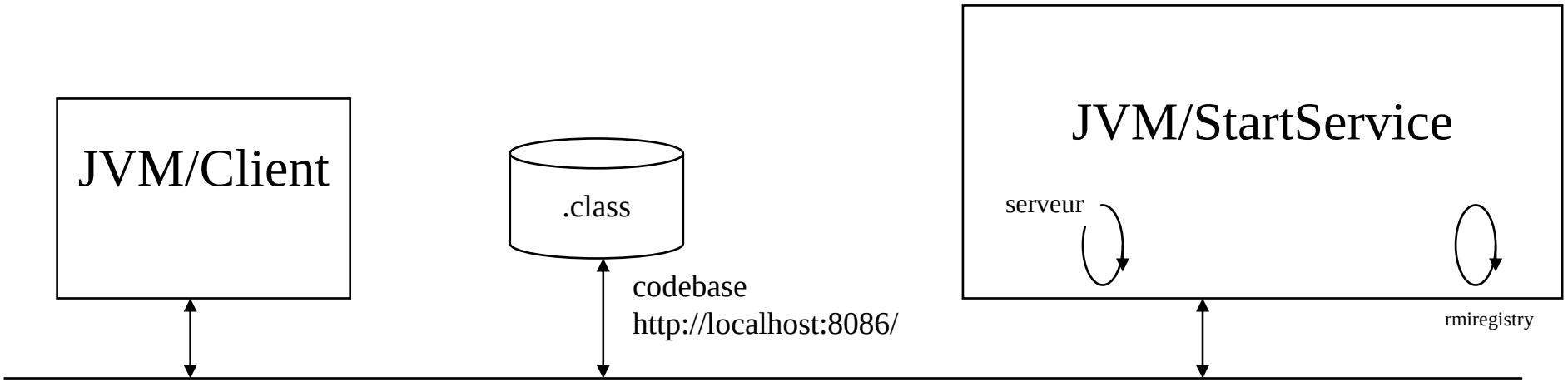
test3

- **L'exemple précédent**
  - **avec des paramètres et un résultat retourné**
  - **et une exception levée côté serveur si la liste transmise est vide**
    - **test2/client>run identique à test2**
    - **test2/client>run2 lève une exception côté serveur…**
    - **Appel distant de serveur.methodeLointaine(une liste)**



test4

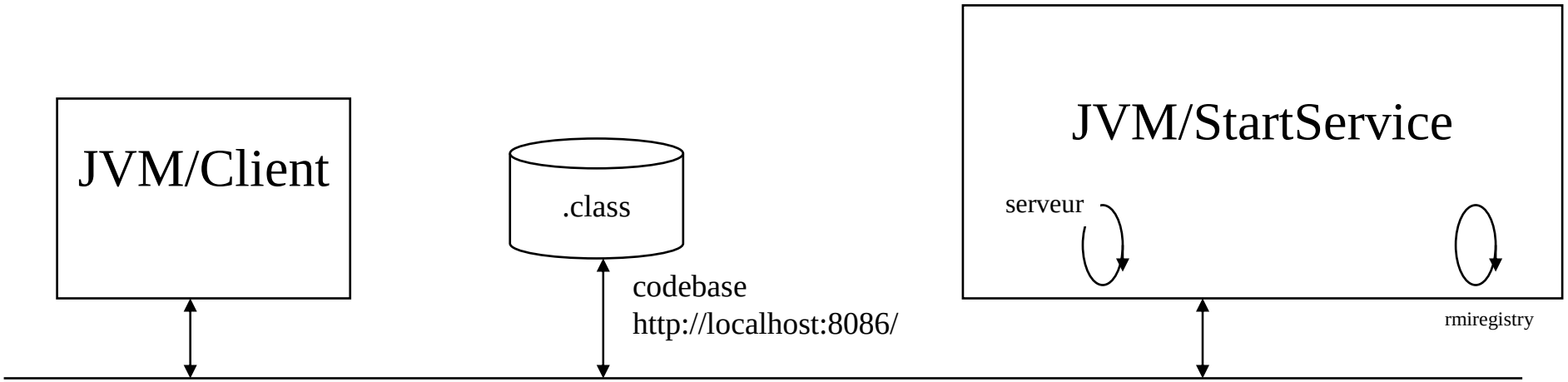
- **Le client souhaite exécuter des « Thread » côté serveur**
  - **Le service : serveur.start(Runnable r)**
  - **Appel côté client de serveur.start(new Compteur())**



- Le client doit indiquer au serveur où se trouve les .class nécessaires ici Compteur.class (que lui seul connaît)
  - Voir run.bat du client et la propriété –Djava.rmi.server.codebase=…

test5

- **Le client exécute les « Thread » côté client**
  - **Le service : Runnable serveur.getRunnable()**
  - **Appel côté client de**
    - **new Thread(serveur.getRunnable()).start()**



- Le serveur doit indiquer aux clients où se trouve les .class nécessaires ici Compteur.class (que lui seul connaît ..)
  - Voir run.bat du serveur et la propriété –Djava.rmi.server.codebase=…

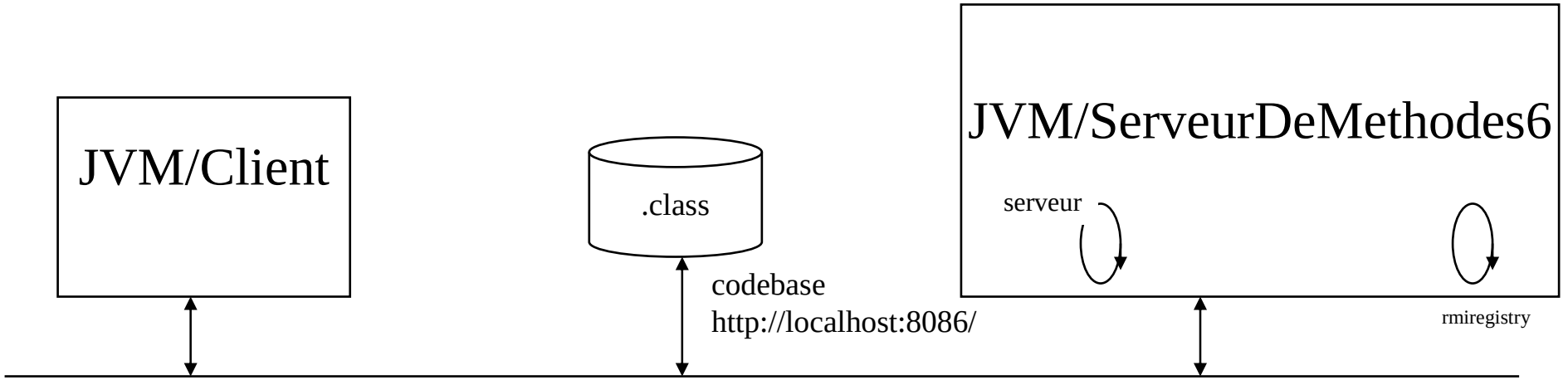
En fin de test5

- **Au tout dernier test**
  - **6 JVM dont 4 serveurs RMI**

```
H:\NSY102\cours_rmi\test5\client>run

H:\NSY102\cours_rmi\test5\client>java -cp ..;. -Djava.security.policy=java.policy Client
0 1 2 3 4 5 6 7 8 9

H:\NSY102\cours_rmi\test5\client>jps
2164 RegistryImpl
2096 ServeurDeMethodes3
1820 ServeurDeMethodes1
3144 ServeurDeMethodes2
2280 Serveur
3344 Jps
3440 ServeurWeb8086
2512 Serveur
```

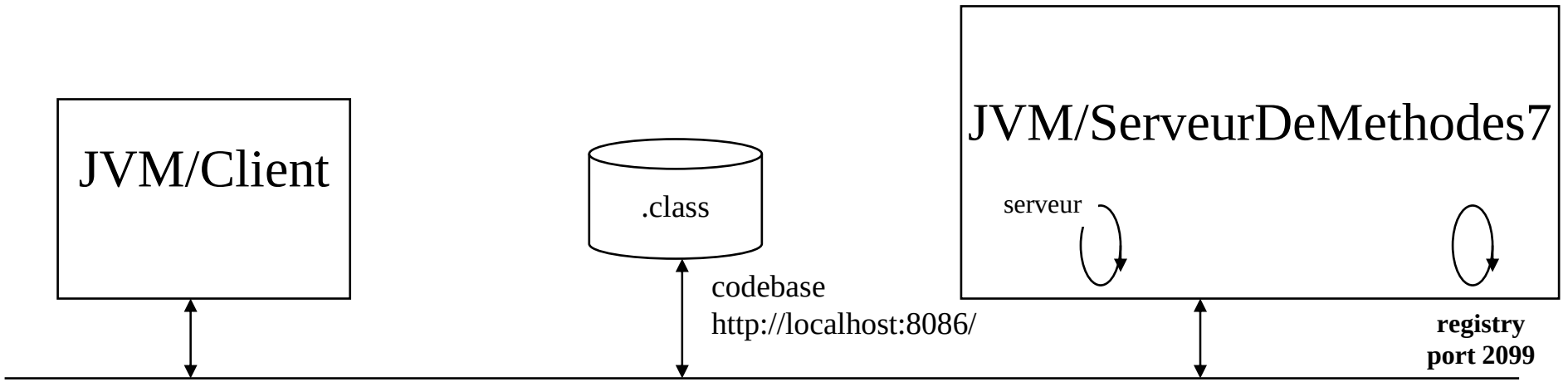
test6

- **L'exemple test3 sans l'héritage de « UnicastRemoteObject »**
  - **Usage de UnicastRemoteObject.exportObject**
    - **Enregistrement du service**
      - **Remote stub = UnicastRemoteObject.exportObject(serveur, 0);**
      - **Naming.rebind(AffichageLointain6.nomDuService, serveur);**
    - **Appel distant de serveur.methodeLointaine(une liste)**



test7

- **L'exemple test6 sans utiliser rmiregistry**
  - **Usage de createRegistry côté serveur**
    - **Registry registry = java.rmi.registry.LocateRegistry.createRegistry(2099);**
  - **Côté client**
    - **Registry registry = java.rmi.registry.LocateRegistry.getRegistry("localhost" ,2099);**
    - **serveur = (AffichageLointain7)registry.lookup(AffichageLointain7.nomDuService);**



test8, test9, test10

- **Critiques de test1 à test5**
  - **UnicastRemoteObject**
    - **Le service demeure actif, une JVM active par service**

- **Activation, Activatable, …**
  - **Service chargé à la demande**
  - **Persistance possible en cas de « crash »**

- **Nécessite un démon (rmid),**
  - **qui charge les services rmi au fur et à mesure des besoins**

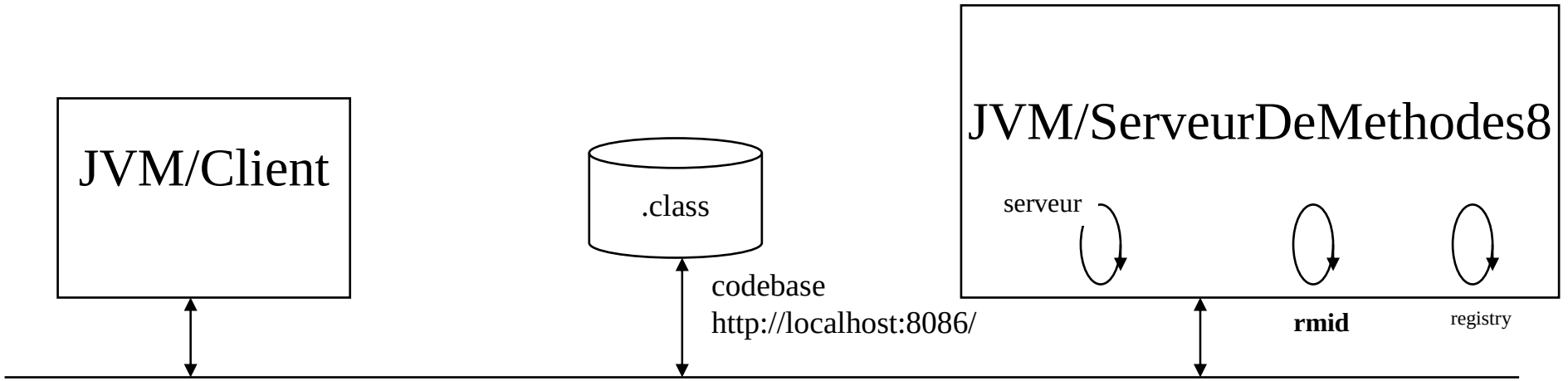
  - Préambule indispensable : **exécuter une seule fois**
    - **>cours_rmi>runOnce_rmid**
      démarre le démon (par défaut en 1098)

test8

- **L'exemple test3 avec « extends Activatable », en utilisant rmid**
  - **côté serveur le constructeur de ServeurDeMethodes8 s'est adapté**

```
public ServeurDeMethodes8 (ActivationID id, MarshalledObject data) throws RemoteException{
  super(id, 0);
 }
```

  - **Client inchangé**



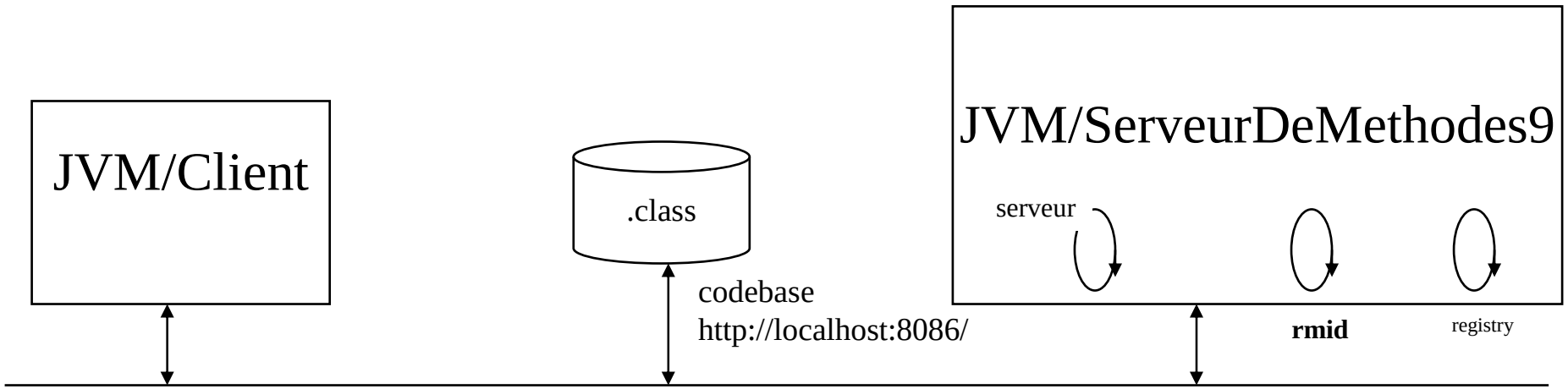
•Lire si besoin http://java.sun.com/javase/6/docs/technotes/guides/rmi/activation/overview.html

test9

- **L'exemple test8 sans l'héritage de Activatable,**
  - **côté serveur le constructeur de ServeurDeMethodes9 s'est adapté …**

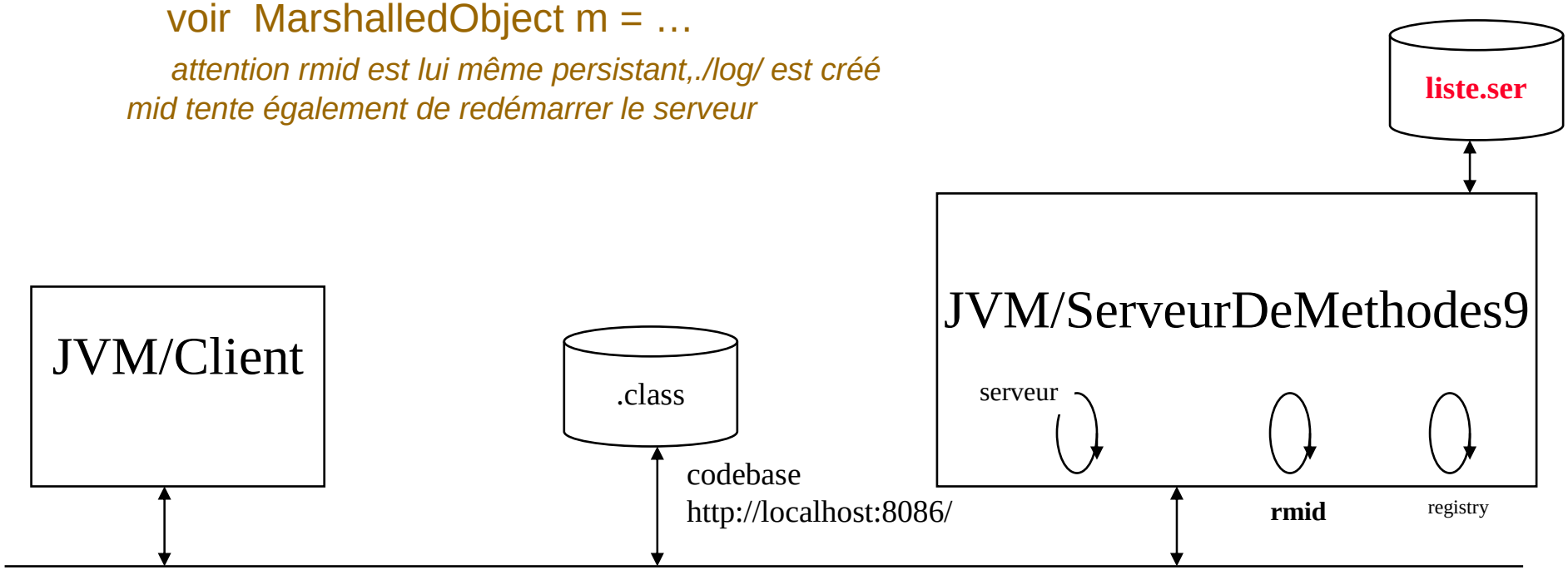
```
public ServeurDeMethodes8 (ActivationID id, MarshalledObject data) throws RemoteException{
  this.id = id;
  Activatable.exportObject(this, id, 0);
 }
```

  - **Client inchangé**



test10

- **L'exemple test9 avec la persistance**
  Persistance de la dernière liste côté serveur (pour l'exemple …)
  Dès le redémarrage du serveur la dernière liste reçue est affichée
  voir MarshalledObject m = …
  attention rmid est lui même persistant,./log/ est créé
  mid tente également de redémarrer le serveur



Activatable, test10 et jps

- **Après l'exécution du serveur (test10/serveur/run)**
  - **jps**

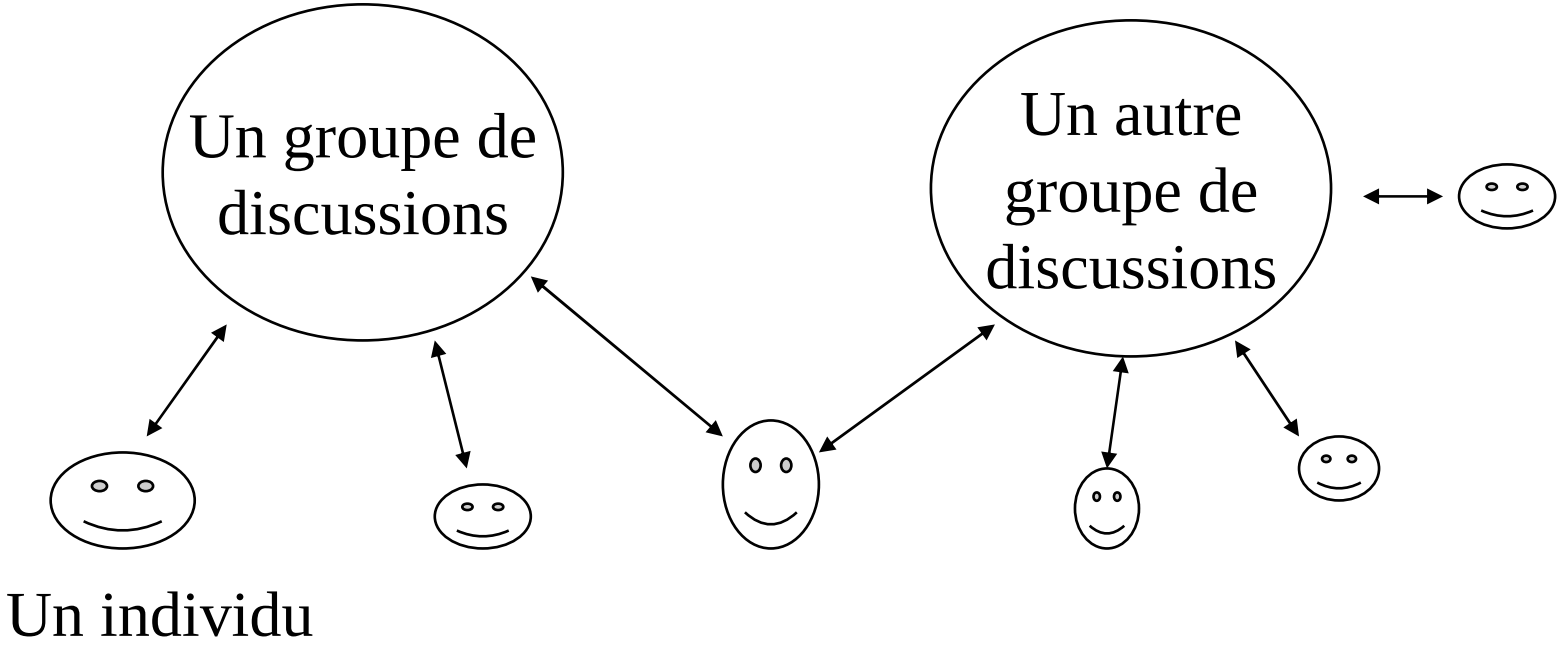
```
2912 ExecServer
2440 Activation
3068 RegistryImpl
2548 Jps
2892 ServeurWeb8086
2316 Boot
```

- **Après l'exécution du client le service est chargé (test10/client/run)**
  - **ActivationGroupInit, a chargé le service, exécuté le constructeur ServeurDeMethode10**

```
2912 ExecServer
1836 ActivationGroupInit
2440 Activation
3068 RegistryImpl
2260 Jps
2892 ServeurWeb8086
2316 Boot
```

- **Après 3 exécutions du client le service s'arrête !**
- **Un nouveau client engendre la restauration du service, persistance inclus**

Annexe4: un Chat



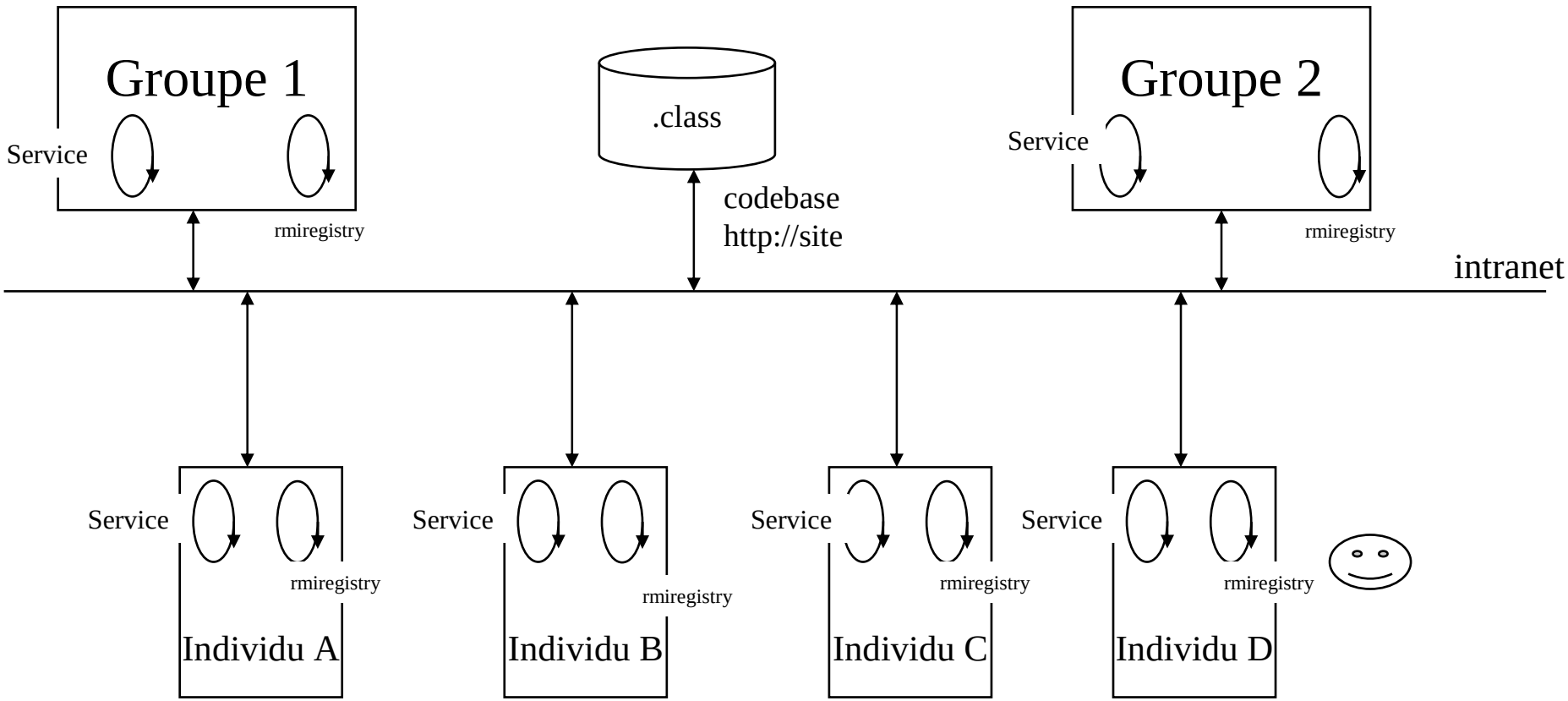
- **Mise en œuvre d'un « Chat »**
  - **Des groupes**
  - **Des individus**

  - **Une instance du Publish/Subscribe**

Archtecture retenue

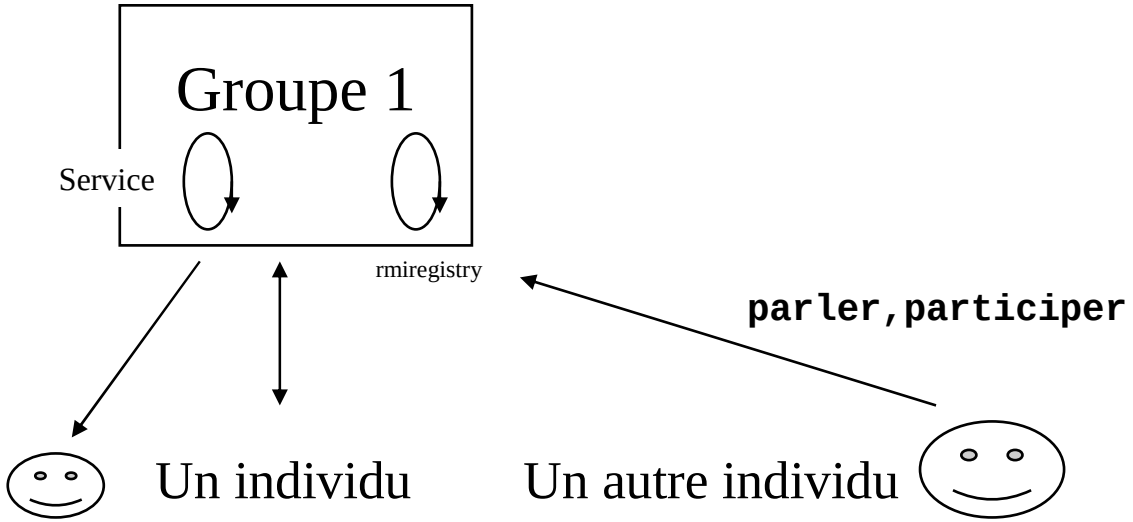
- **Serveur Web pour le codebase**
- **Un serveur rmiregistry par machine**
  - contrainte de sécurité
- **Un serveur par groupe,**
  - Enregistrements des individus comme participant
  - Diffusion des messages aux participants
- **Un serveur par individu**
  - Enregistrement auprès d'un ou de plusieurs groupes
  - Envoi d'un message au groupe

Architecture RMI



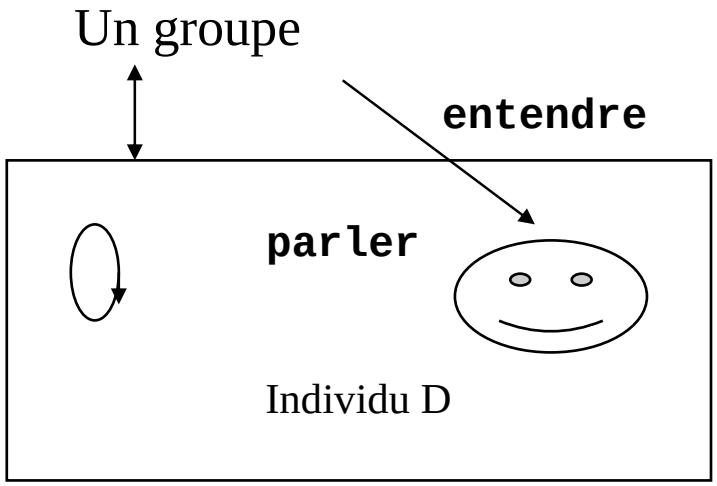
- rmiregistry, sur chaque machine; un serveur rmi par groupe, et par individu

interface GroupeDeDiscussion



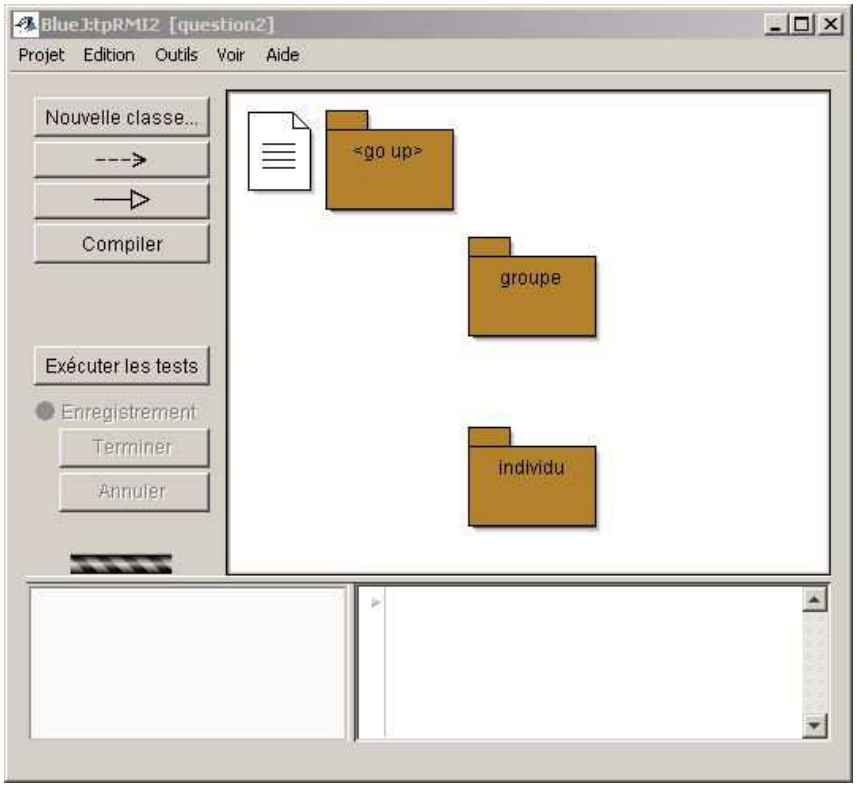

```
public interface GroupeDeDiscussion extends Remote{

 public void participer(Individu individu) throws RemoteException;
 public void sortir(Individu individu) throws RemoteException;

 public void saluer(Individu individu) throws RemoteException;
 public void parler(Individu individu, String phrase) throws RemoteException;
 public void chuchoter(Individu source,
                       String   phrase,
                       Individu destinataire) throws RemoteException;
 public Set<Individu> listeDesParticipants() throws RemoteException;
}
```

interface Individu



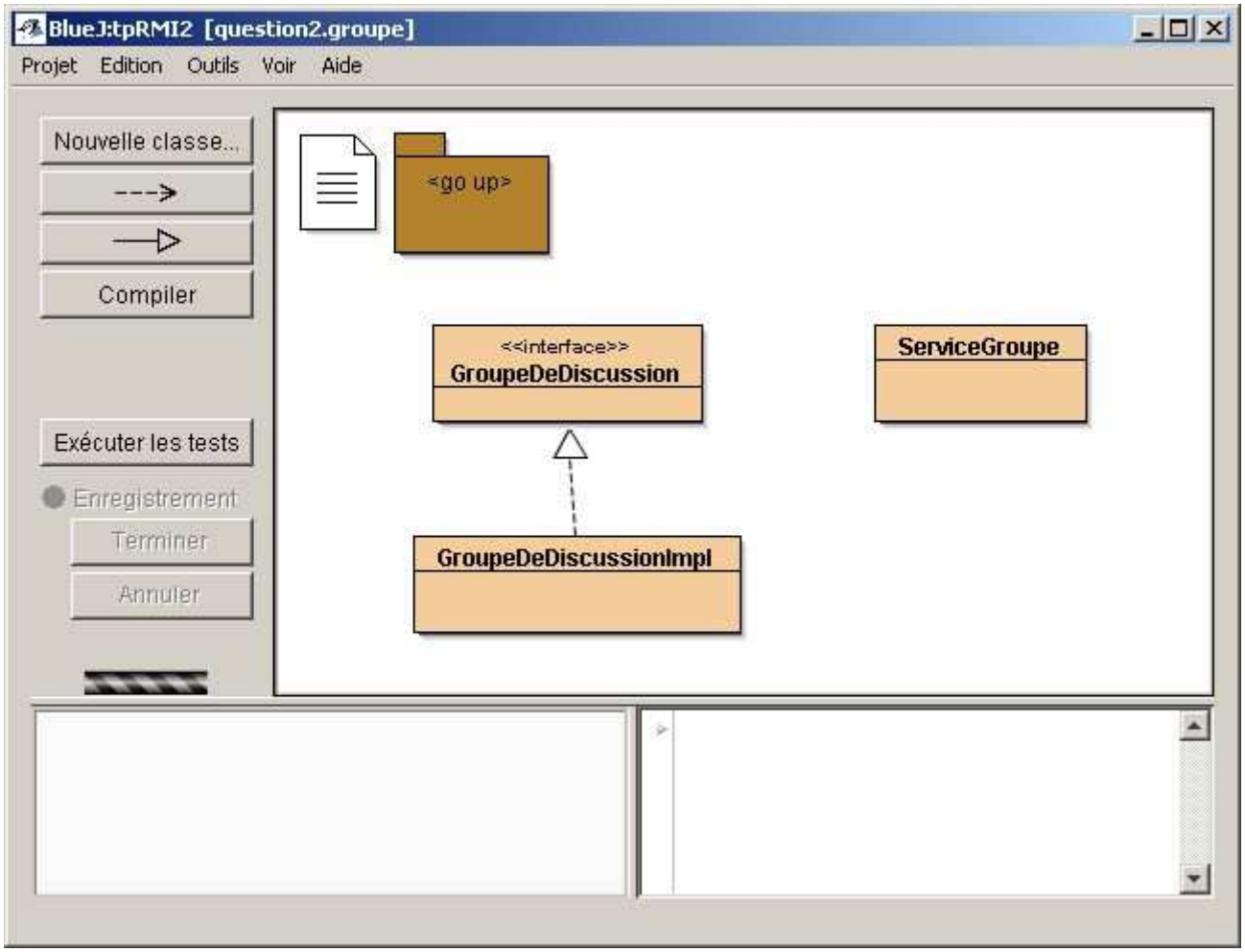

```
public interface Individu extends Remote, Serializable{

  public String nom() throws RemoteException;


  public void entendre(String phrase) throws RemoteException;

}
```

En Java : 2 paquetages



- groupe
  - interface GroupeDeDiscussion
  - classe GroupeDeDiscussionImpl
  - classe ServiceGroupe
- **individu**
  - **interface Individu**
  - **classe IndividuImpl**
  - **classe ServiceIndividu**

le paquetage groupe



Classe GroupeDeDiscussionImpl, un extrait

```
public class GroupeDeDiscussionImpl
  extends UnicastRemoteObject
       implements GroupeDeDiscussion{
         private Set<Individu> participants;
         private String        nomDuGroupe;
```

*Un individu **participe** à ce groupe de discussion*

```
public void participer(Individu individu) throws RemoteException{
    participants.add(individu);
  }
```

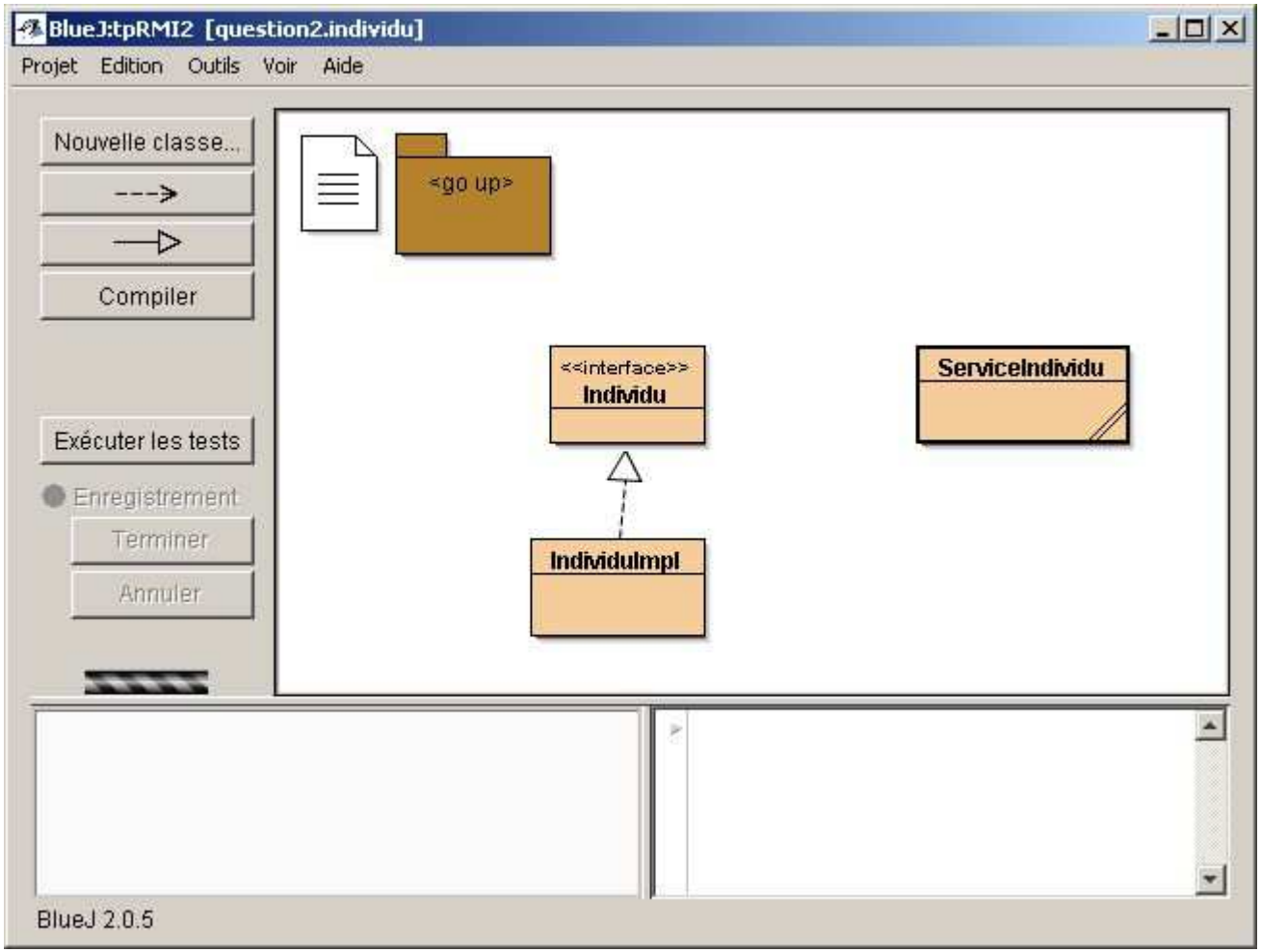
*Un individu **parle** au groupe*

```
public void parler(Individu individu, String phrase) throws RemoteException{
    for( Individu i : participants){
     try{
        if(!i.equals(individu)) // il demande aux autres de l'entendre
          i.entendre(nomDuGroupe + "_" + individu.nom() + " << " + phrase + " >>");
      }catch(RemoteException e){
        sortir(i); // parti ?
      }
    }
  }
```

Classe ServiceGroupe

```
public class ServiceGroupe{

  public static void main(String[] args) throws Exception{
    if(args.length>0)
     try{
        Naming.rebind(args[0], new GroupeDeDiscussionImpl(args[0]));
        System.out.println("Le groupe installé en " +
                           InetAddress.getLocalHost().getHostAddress() +
                           "/" + args[0] + " est créé ...");

      }catch(Exception e){
        e.printStackTrace();
      }
    else
      System.out.println("usage >java -Djava.security.policy=... "  +
                                    "-Djava.rmi.server.codebase=...  " +
                                      "ServiceGroupe un_nom");

  }
}
```

le paquetage Individu



Classe IndividuImpl, un extrait

```
public class IndividuImpl implements Individu, Runnable{
```

*Un individu **rejoint** un groupe*

```
  public void rejoindre(GroupeDeDiscussion groupe){
    try{
        groupe.participer((Individu)Naming.lookup("rmi://" +
                InetAddress.getLocalHost().getHostAddress() + "/" + nom));
        this.groupe = groupe;
        this.groupe.saluer(this);
        ...
      }catch(Exception e){
        e.printStackTrace();
      }
    }
```

*Un individu **entend** ...*

```
    public void entendre(String phrase)  throws RemoteException{
      System.out.println(phrase);
    }
```

Classe IndividuImpl, suite de l 'extrait

*Un individu **parle** ...*

```
private void parler(String phrase) {
   try{
    this.groupe.parler(this, phrase);
   }catch(RemoteException re){……}
 }
```

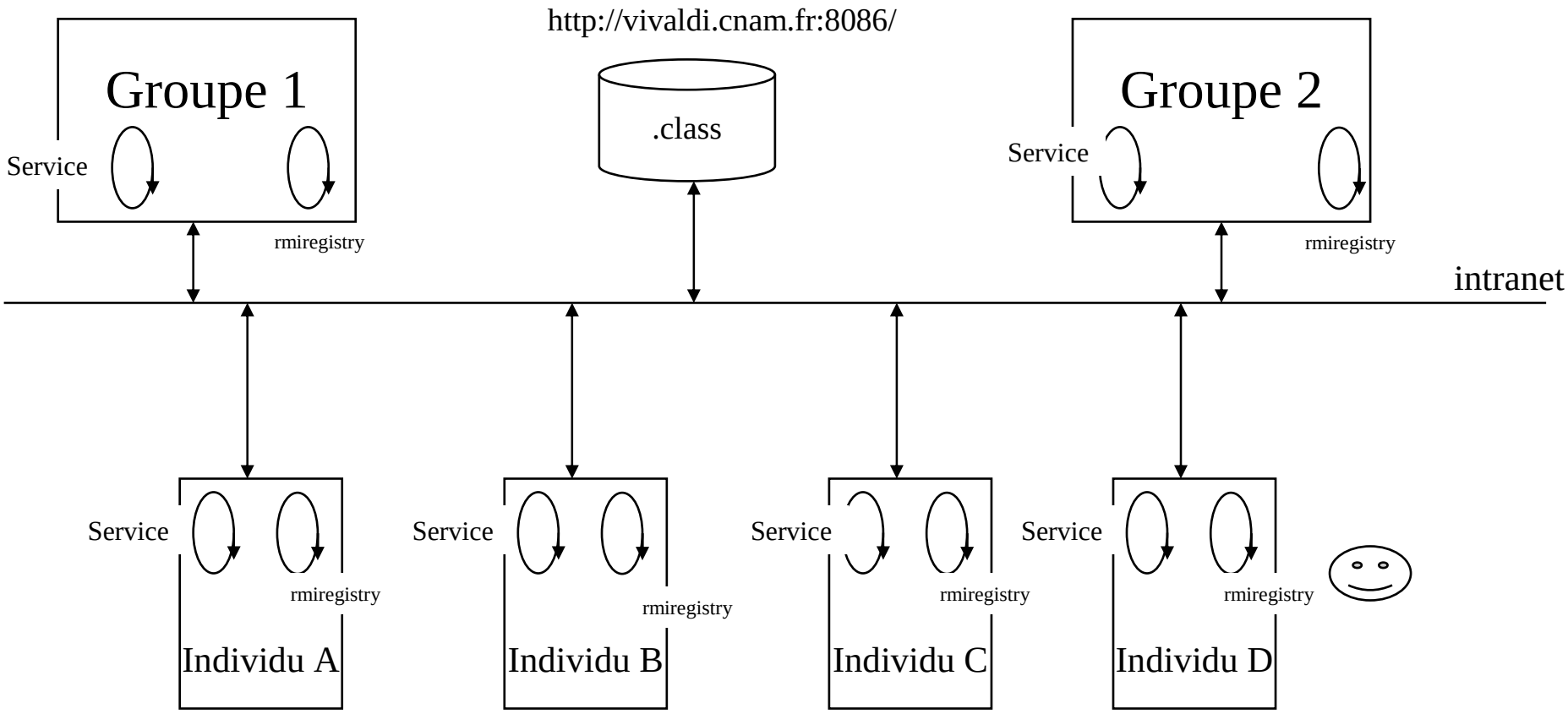
*Un individu **chuchote***

```
private void chuchoter(String destinataire, String phrase) throws
 RemoteException{
   Set<Individu> liste = this.groupe.listeDesParticipants();
   for(Individu i : liste)
     if(i.nom().equals(destinataire)){
       this.groupe.chuchoter(this, phrase, i);
       break;
     }
}
```

ClasseServiceIndividu

```
public class ServiceIndividu{

  public static void main(String[] args) throws Exception{
    if (args.length>=2)
      try{
        GroupeDeDiscussion groupe=(GroupeDeDiscussion)Naming.lookup("rmi://"+ args[1]);

        Individu lambda = new IndividuImpl(args[0]);
// IndividuImpl extends UnicastRemoteObject
// et Naming(args[0], lambda);
// ou bien les 3 lignes ci-dessous
        Remote stub = UnicastRemoteObject.exportObject(lambda, 0);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind(args[0], stub);

        lambda.rejoindre(groupe);

        System.out.println("L'individu nommé " + lambda.nom() + " est entré en " +
                           args[1] + " et est prêt <<< à parler >>>");
      }catch(Exception e){throw e;}

    else
      System.out.println("usage >java -Djava.security.policy=...  " +
    " -Djava.rmi.server.codebase=...  " +
    " ServiceIndividu alfred @IP/nom_du_groupe (ex. 163.173.228.59/AmphiA) ou localhost)
  ");
  }
```

Architecture RMI Chat au Cnam



- Un essai en intranet, reproductible chez vous ….

les commandes

- **le répertoire**
  - http://jfod.cnam.fr/rmi/
  - start rmiregistry (sur chaque machine groupe comme individu)
  - un groupe
  - **java -cp chat.jar -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://jfod.cnam.fr/rmi/chat.jar question2.groupe.ServiceGroupe nfp120**
    - *Le groupe installé en 163.173.228.59/nfp120 est créé …*
  - deux individus
  - **java -cp chat.jar -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://jfod.cnam.fr/rmi/chat.jar question2.individu.ServiceIndividu paul 163.173.228.59/nfp120**
  - **java -cp chat.jar -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://jfod.cnam.fr/rmi/chat.jar question2.individu.ServiceIndividu jean 163.173.228.59/nfp120**