

# Master Informatique

Programmation distribuée  
M1 / 178UD02

## C3 – Java RMI

Thierry Lemeunier

[thierry.lemeunier@univ-lemans.fr](mailto:thierry.lemeunier@univ-lemans.fr)

[www-lium.univ-lemans.fr/~lemeunie](http://www-lium.univ-lemans.fr/~lemeunie)

# Plan du cours

- Communication par objet distribué :
  - Principes
  - Référenciation des objets distants
  - Passage des paramètres
  - Java *RMI*
    - Principes
    - Implémentation
    - API Java RMI

# Plan du cours

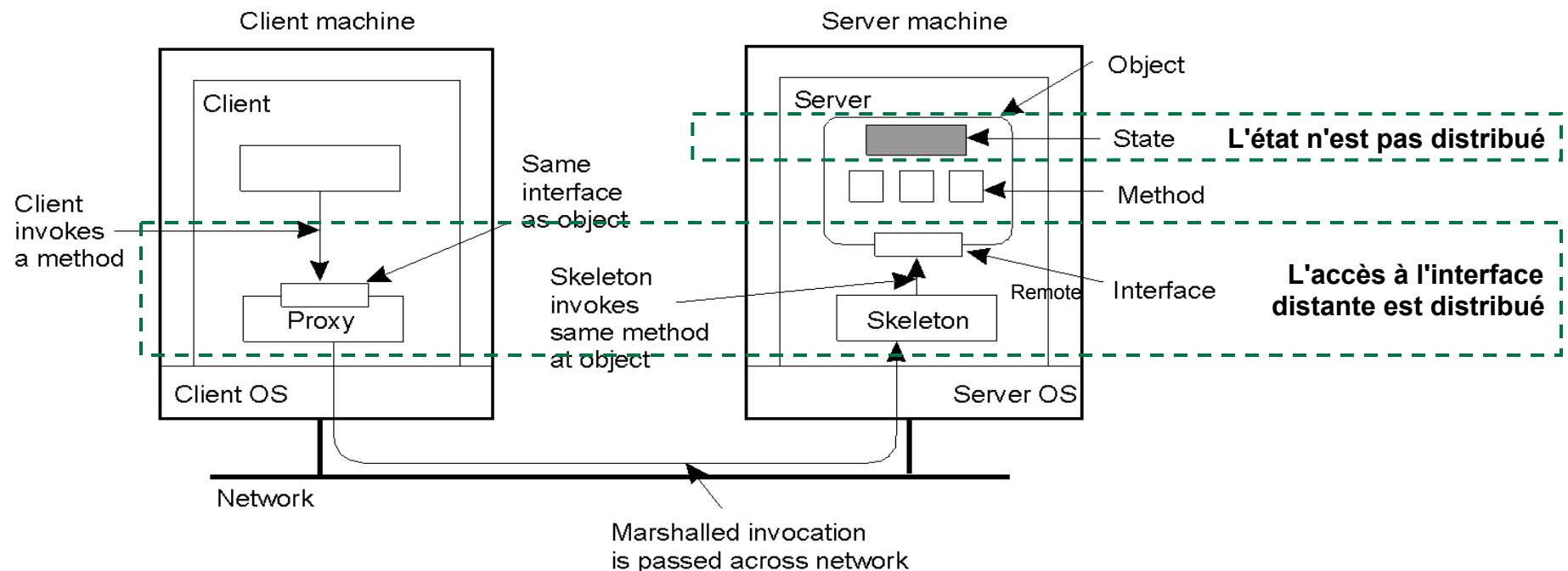
- Communication par objet distribué :
  - Principes
  - Référenciation des objets distants
  - Passage des paramètres
  - Java *RMI*
    - Principes
    - Implémentation
    - API Java RMI

# Objet distribué – Principes (1/2)

- Principes :
  - Donner la possibilité à une machine cliente d'invoquer une méthode sur un objet distant se trouvant sur une machine serveur
    - ➔ *RMI : Remote Method Invocation*
- Philosophie : faire en sorte qu'un client puisse utiliser un objet distant comme s'il s'agit d'un objet local
  - ➔ Transparence de la distribution des objets et de la communication
- Impact :
  - RMI est le second *middleware* à avoir été défini après RPC (années 1995)
  - Il est très répandu car il est associé à Java
- Rappel :
  - Un objet est constitué d'attributs et de méthodes
  - Les valeurs des attributs forment l'état de l'objet
  - Sérialiser un objet : extraire l'état de l'objet et en faire une suite d'informations
  - Dé-sérialiser un objet : créer un objet à partir de la suite d'informations **et de sa classe**

# Objet distribué – Principes (2/2)

- La transparence s'effectue via la distribution
  - de l'interface distante de l'objet distant sur le client = le *proxy*
  - et de l'exportation de l'interface distante sur le serveur = le *skeleton*
- Le *proxy* et le *skeleton* jouent le rôle d'emballage et de déballage de l'invocation de méthode en message et gestion de l'envoi de ce message

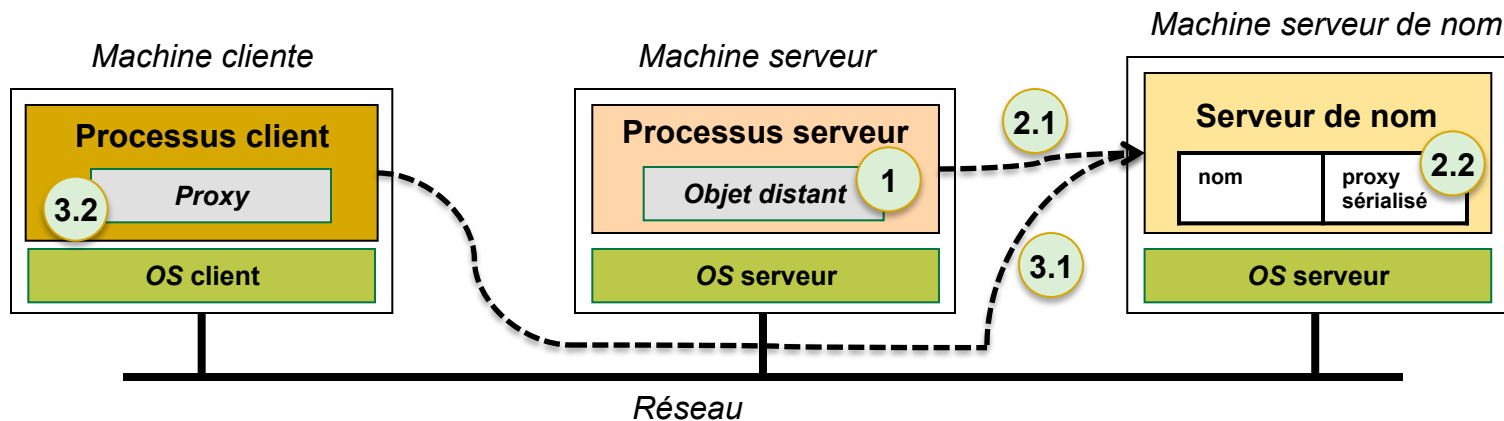


# Plan du cours

- Communication par objet distribué :
  - ✓ Principes
  - Référenciation des objets distants
  - Passage des paramètres
  - Java *RMI*
    - Principes
    - Implémentation
    - API Java RMI

# Objet distribué – Référenciation (1/2)

- Comment connaître et utiliser un objet distant ?
- Le processus de référencement :
  - ① Exportation : le processus serveur crée un objet puis l'exporte ; un objet exporté a une URL + un nom + un port de communication
  - ② Publication :
    - ① Le processus serveur rend public l'objet distant via un serveur de nom par une requête d'enregistrement
    - ② Le serveur de nom maintiens actif une table d'association entre un nom d'objet distant et un représentant de l'objet distant (proxy)
  - ③ Utilisation :
    - ① Le processus client fait une requête de localisation au serveur de nom; le serveur répond en sérialisant le proxy au client
    - ② Le client dé-séréalise le proxy puis invoque les méthodes du proxy



# Objet distribué – Référenciation (2/2)

- Il y a deux modes de référencement :
  - Dans les deux modes, le serveur et le client doivent charger les classes nécessaires à l'exécution de l'objet distant et du *skeleton* ou du *proxy*
  - Référenciation par chargement local :
    - le serveur charge localement les classes de l'objet distant et du *skeleton*
    - le client charge localement la classe du *proxy*
    - ainsi que toutes les autres classes nécessaires (paramètres ou valeur de retour)
  - Référenciation par téléchargement dynamique :
    - Pour le serveur
      - Le code = tout ce qui est nécessaire pour exécuter l'objet distant et le *skeleton*
      - Le code est téléchargé sur le serveur au moment de la création de l'objet distant ou au moment de l'exécution du code de la méthode distante
    - Pour le client
      - Le code = tout ce qui est nécessaire pour exécuter le *proxy* sur le client
      - Le code est téléchargé sur le client au moment de la première référencement de l'objet distant (c'est-à-dire à la désérialisation du proxy)
- Cela nécessite de paramétrer le client et le serveur
- Cela nécessite **une politique de chargement et d'exécution** sécurisés (afin de s'assurer que le code chargé n'effectue pas d'opérations interdites !)

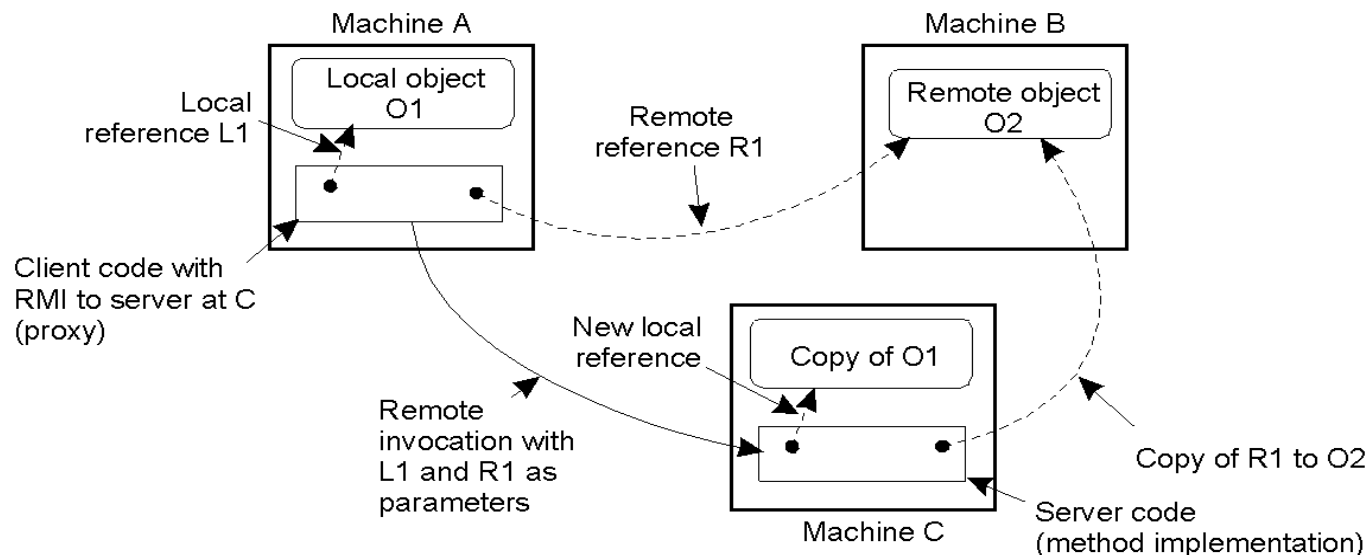


# Plan du cours

- Communication par objet distribué :
  - ✓ Principes
  - ✓ Référenciation des objets distants
  - Passage des paramètres
  - Java *RMI*
    - Principes
    - Implémentation
    - API Java RMI

# Objet distribué – Passage des paramètres

- Les paramètres et les valeurs retournées doivent être sérialisables
- Si la méthode distante a des paramètres ou retourne une valeur :
  - ❑ Les types primitifs sont passés/retournés par valeur
  - ❑ Les objets locaux sont passés/retournés par valeur
  - ❑ Les objets distants non exportés sont passés/retournés par valeur
  - ❑ Les objets distants exportés sont passés/retournés « par référence » c'est-à-dire par copie du proxy



# Plan du cours

- Communication par objet distribué :

- ✓ Principes
- ✓ Référenciation des objets distants
- ✓ Passage des paramètres

- Java *RMI*

- Principes
  - Implémentation
  - API Java RMI

# Objet distribué – Java RMI – Principes (1/4)

## ■ L'offre :

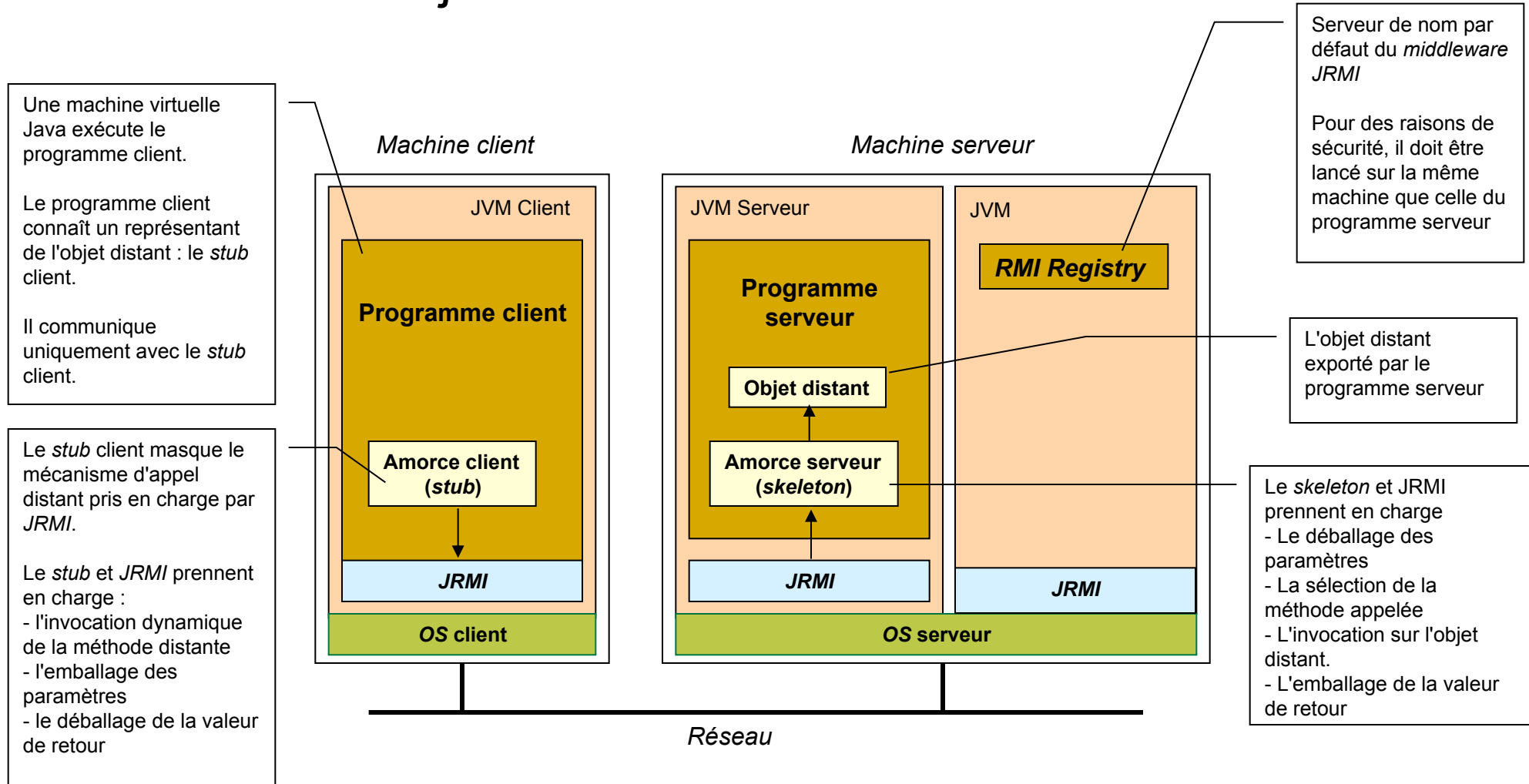
- ❑ Java RMI est une spécification d'un *middleware* RMI (Oracle)
- ❑ Java RMI existe depuis le JDK1.1 de 1997 (mais a été simplifié depuis)
- ❑ Java RMI marche avec des objets Java (ou d'autres langages : cf. CORBA) !

## ■ Sous le capot :

- ❑ L'API permet de faire interagir des objets créés dans des machines virtuelles différentes lancées sur une même machine ou sur des machines différentes
- ❑ Il utilise les *sockets* (par défaut : TCP/IP non sécurisé)
- ❑ Il utilise un protocole spécifique (*Java Remote Method Protocol*) ou CORBA
- ❑ Il préserve la sécurité inhérente au langage Java avec notamment :
  - *RMISecurityManager* : un gestionnaire de sécurité par défaut
  - *Distributed Garbage Collector (DGC)* : le ramasse-miettes distribué
- ❑ Téléchargement dynamique possible (non abordé dans ce cours)
  - Positionner la propriété *java.rmi.server.codebase* (côté client et / ou côté serveur)
  - Définir une politique de sécurité soit en écrivant un gestionnaire de sécurité soit en positionnant la propriété *java.security.policy* pointant sur un fichier *policy*

# Objet distribué – Java RMI – Principes (2/4)

## Le modèle d'objet distribué de Java RMI



# Objet distribué – Java RMI – Principes (3/4)

## Fonctionnement côté serveur : démarrage du programme serveur

### 1. Exportation de l'objet distant

- Un objet distant doit être exporté pour que les clients puissent invoquer des méthodes sur lui. L'exportation consiste à :
  - créer un *stub* de l'objet distant (qui sera publié via le service de nom) + un *skeleton*
  - et ouvrir un port d'écoute sur la machine hôte pour recevoir le flux d'invocation envoyé par le client une fois la référencement établie

### 2. Publication de l'objet distant

- La publication rend accessible l'objet distant en utilisant un service de nom
- Un service de nom est un processus serveur qui maintient à jour une table de liaison entre le nom et le *stub* d'un objet distant
- En pratique on ne donne donc pas accès à l'objet distant mais à son *stub*
- Les services de nom utilisables sont :
  - Le service *rmiregistry* de *JRMI* : service de nom simple (sans persistance) de la machine serveur commun à tous les processus serveurs de la machine hôte
  - Un registre propre au processus : registre privé inaccessible aux autres processus serveurs de la machine hôte
  - JNDI (*Java Naming and Directory Interface*) : API fournissant une interface unique d'accès à différents services de nommage ou d'annuaires (LDAP, RMI Registry, DNS, NIS, COS Naming...)

# Objet distribué – Java RMI – Principes (4/4)

## Fonctionnement côté client : accès et utilisation de l'objet distant

### 1. Localisation de l'objet distant

- ❑ Le client doit trouver l'objet distant pour l'utiliser
- ❑ Il fait une requête au service de nom en indiquant le nom de l'objet distant recherché (il doit connaître la machine serveur de nom)
- ❑ Le service de nom lui répond en lui envoyant une copie du *stub* de l'objet distant

### 2. Utilisation de l'objet distant

- ❑ Une fois que le client possède un *stub* sur l'objet distant, il peut invoquer des méthodes sur ce *stub* qui implémente l'interface distante
- ❑ Le *stub* n'exécute pas la méthode lui-même (il ne la possède pas) mais fait une invocation dynamique sur l'objet distant qu'il représente
- ❑ Le *stub* communique avec le *skeleton*

# Plan du cours

## ■ Communication par objet distribué :

- ✓ Principes
- ✓ Référenciation des objets distants
- ✓ Passage des paramètres

## □ Java *RMI*

- ✓ Principes
- Implémentation
  - API Java RMI



# Objet distribué – Java RMI – Implémentation (1/6)

## ■ Démarche d'implémentation

1. Définir l'interface distante (= une classe interface Java)
2. Ecrire une classe d'implémentation de l'interface distante pour instancier l'objet distant
3. Implémenter un serveur qui crée, exporte et publie l'objet distant
4. Implémenter un client qui localise et utilise l'objet distant
5. Compiler les fichiers .java

## ■ Démarche « classique » de déploiement manuel

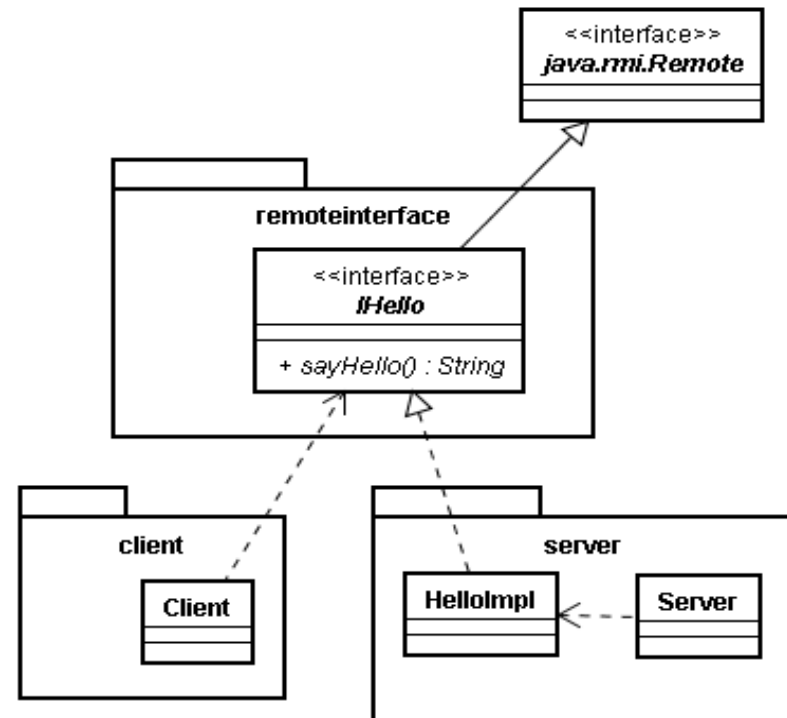
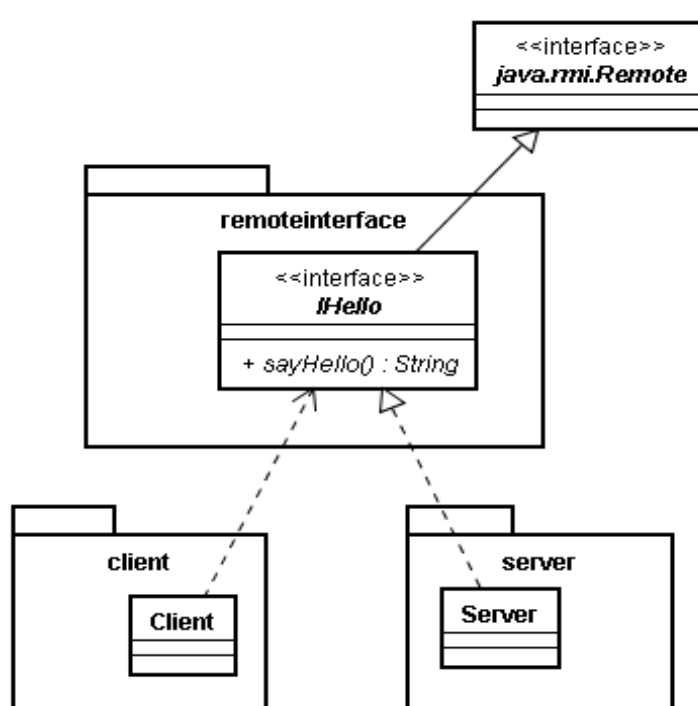
1. Générer les classes *stub* et *skeleton* avec l'outil *rmic* du jdk
2. Copier le fichier *stub* sur le client et le fichier *skeleton* sur le serveur
3. Démarrer un service de nom (optionnel quand on utilise un registre privé)
  - Exemple avec *rmiregistry* : par défaut ce service écoute sur le port 1099 du serveur  
Sous Windows : `start rmiregistry [num_port]`  
Sous Unix : `rmiregistry [num_port] &`
4. Lancer le serveur
5. Lancer le client

## ■ Remarques :

- Le déploiement est automatique par référencement par téléchargement
- A partir du jdk 1.2 : plus besoin de générer le *skeleton* si le serveur est  $\geq 1.2$  !
- A partir du jdk 1.5 : plus besoin de générer le *stub* si le client est  $\geq 1.5$  !

# Objet distribué – Java RMI – Implémentation (2/6)

- Un exemple simple : le « hello world » distribué
  - Deux implémentations possibles du serveur
    - Soit le serveur implémente lui-même l'interface distante (à gauche)
    - Soit le serveur utilise une classe d'implémentation (à droite)



# Objet distribué – Java RMI – Implémentation (3/6)

- Définition de l'interface distante *!Hello* :
  - Une interface distante étend l'interface *java.rmi.Remote*
  - Elle déclare le service rendu par le serveur en déclarant les méthodes distantes
  - Les méthodes distantes lèvent une *java.rmi.RemoteException*

```
package remoteinterface;

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

public interface !Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

# Objet distribué – Java RMI – Implémentation (4/6)

- Implémentation de l'interface distante
  - Il y a 2 méthodes possibles d'implémentation
    - Soit l'objet distant étend *java.rmi.server.UnicastRemoteObject*
      - ➔ *L'exportation se fait automatiquement à la création de l'objet distant*
    - Soit l'objet distant n'étend pas *UnicastRemoteObject*
      - ➔ *L'exportation doit se faire manuellement*
        - Soit dans le constructeur de l'objet distant
        - Soit au démarrage du serveur
- Exemple d'implémentation dans une classe séparée avec exportation automatique

```
package server;

import remoteinterface.IHello;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public HelloImpl extends UnicastRemoteObject implements IHello {
    public HelloImpl() throws RemoteException {};
    String sayHello() throws RemoteException { return "Hello World"; }
}
```

# Objet distribué – Java RMI – Implémentation (5/6)

## ■ Exemples d'implémentation d'un serveur

- ❑ A gauche : le serveur utilise un objet distant qu'il crée (exportation automatique) puis publie
- ❑ A droite : le serveur implémente l'interface (c'est l'objet distant), s'exporte manuellement et se publie

```
...
public class Server {
    public static void main(String args[]) {
        try {
            IHello od = (IHello)new HelloImpl();
            Naming.rebind("Hello", od);
            System.out.println("Server ready");
        } catch (Exception e) {...}
    }
}
```

```
...
public class Server implements IHello {
    public String sayHello() throws RemoteException { return "Hello World"; }
    public static void main(String args[]) {
        try {
            Server obj = new Server();
            IHello od = (IHello)UnicastRemoteObject.exportObject(obj, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", od);
            System.out.println("Server ready");
        } catch (Exception e) {...}
    }
}
```

## ■ Remarques :

- ❑ rmiregistry peut être accédé soit par la classe *java.rmi.registry.LocateRegistry* soit par la classe *java.rmi.Naming* (cf. fin du cours)
- ❑ *JRMI* maintient le serveur actif tant que l'objet distant est référencé (par le client et/ou le registre)
- ❑ Les invocations sur l'objet distant sont « threadées » pas *JRMI* (un thread est créé pour exécuter la méthode invoquée) : l'objet distant doit gérer l'accès parallèle (*thread-safe*)

# Objet distribué – Java RMI – Implémentation (6/6)

## ■ Implémentation d'un client

```
package client;

import remoteinterface.IHello;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    private Client() {}

    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            IHello stub = (IHello) registry.lookup("Hello");
            System.out.println("response: " + stub.sayHello());
        } catch (Exception e) {...}
    }
}
```

# Plan du cours

## ■ Communication par objet distribué :

- ✓ Principes
- ✓ Référenciation des objets distants
- ✓ Passage des paramètres

## □ Java *RMI*

- ✓ Principes
- ✓ Implémentation
- API Java RMI

# Objet distribué – Java RMI – API Java RMI

## ■ Bref tour d'horizon de l'API Java RMI

### □ Package *java.rmi*

- Surtout utile pour le client
- Contient les classes :
  - *Naming* : accès à un registre de référencement via une URL (ressemble à *LocateRegistry*)
  - *RMISecurityManager* : gestion de la police de sécurité pour *JRMI*
  - La hiérarchie des exceptions liées à *JRMI* (héritent de *RemoteException*)

### □ Package *java.rmi.server*

- Surtout utile pour le serveur
- Contient les classes :
  - *RemoteObject* : définition d'un objet distant
  - *UnicastRemoteObject* : exportation/dé-exportation d'un objet distant et création de *stub*

### □ Package *java.rmi.registry*

- Utile pour accéder directement à un registre commun ou créer un registre privé
- Contient l'interface *Registry* et la classe *LocateRegistry*

### □ Package *java.rmi.dgc* : classes pour le *DGC*

### □ Package *java.rmi.activation* : classes pour l'activation à la volée d'objets distants