RMI
Construction d'application avec Java RMI

Rappel des RPC

RPC (Remote Procedure Call)

- modèle client/serveur
- appel de procédures à distances entre un client et un serveur
- le client appelle une procédure
- le serveur exécute la procédure et renvoie le résultat
- Outil USFJHQ
- génère la souche d ’invocation et le squelette du serveur

(en C, C++, Fortran, ...)

- la souche et le squelette ouvre un socket BSD et encode/décode les

paramètres

- Couche de présentation ;'5 (eXchange Data Representation)

Limites des RPC

Limitations

- **paramêtres et valeur de retour sont des types primitifs**

programmation procédurale

dépendance à la localisation du serveur pas d ' objet

pas de « référence distante »

Introduction

RMI est un ensemble de classes permettant de manipuler des objets sur des machines distantes (objets distants) de manière similaire aux objets sur la machine locale (objet locaux).

C'est un peu du "RPC orienté objet". Un objet local demande une fonctionnalité à un objet distant.

RMI apparaît avec Java 1.1 et est complètement intégré depuis Java 1.1

Qu'attend t-on d'un objet distribué ?

Un objet distribué doit pouvoir être vu comme un objet « normal ».

Soit la déclaration suivante :

ObjetDistribue monObjetDistribue;

• On doit pouvoir invoquer une méthode de cet objet situé

sur une autre machine de la même façon qu'un objet local :

monObjetDisribue.uneMethodeDeLOD();

Qu'attend t-on d'un objet distribué ?

• On doit pouvoir utiliser cet objet distribué sans connaître sa localisation. On utilise pour cela un service sorte d'annuaire, qui doit nous renvoyer son adresse.

monObjetDistribue=

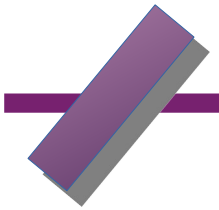
ServiceDeNoms.recherche('myDistributedObject');

• On doit pouvoir utiliser un objet distribué comme paramètre d'une méthode locale ou distante.

x=monObjetLocal.uneMethodeDeLOL(monObjetDistribue);

x= monObjetDistribue.uneMethodeDeLOD(autreObjetDistribue);

Java RMI



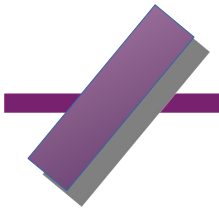
Remote Method Invocation

permet la communication entre machines virtuelles Java (JVM)

qui peuvent se trouver physiquement sur la même machine

ou sur deux machines distinctes.

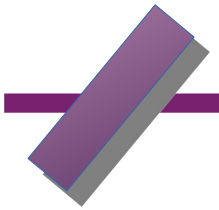
RMI



RMI est un système d'objets distribués constitué uniquement d'objets java ;

- RMI est une Application Programming Interface (intégrée au JDK 1.1 et plus) ;
- Développé par JavaSoft ;

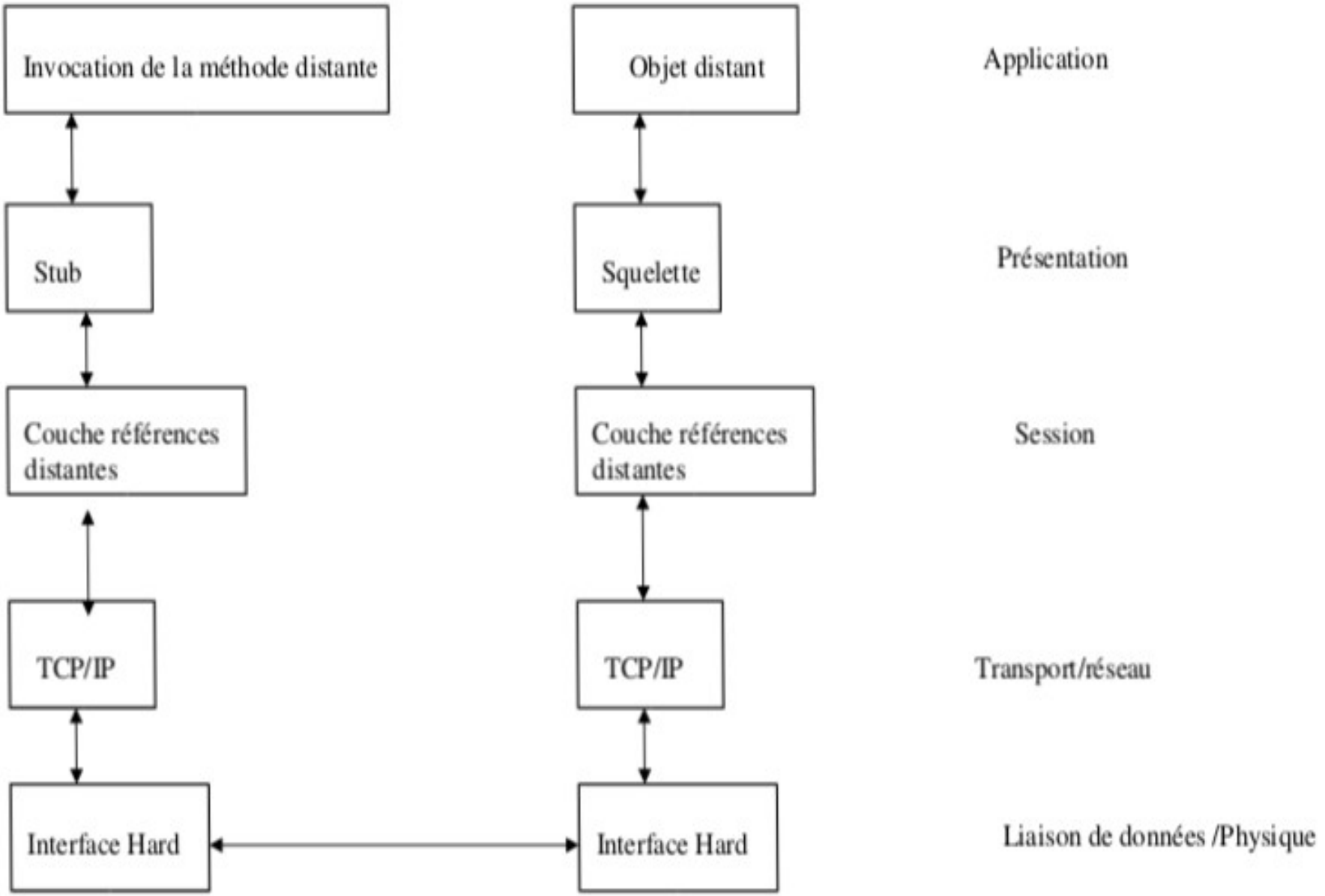
RMI



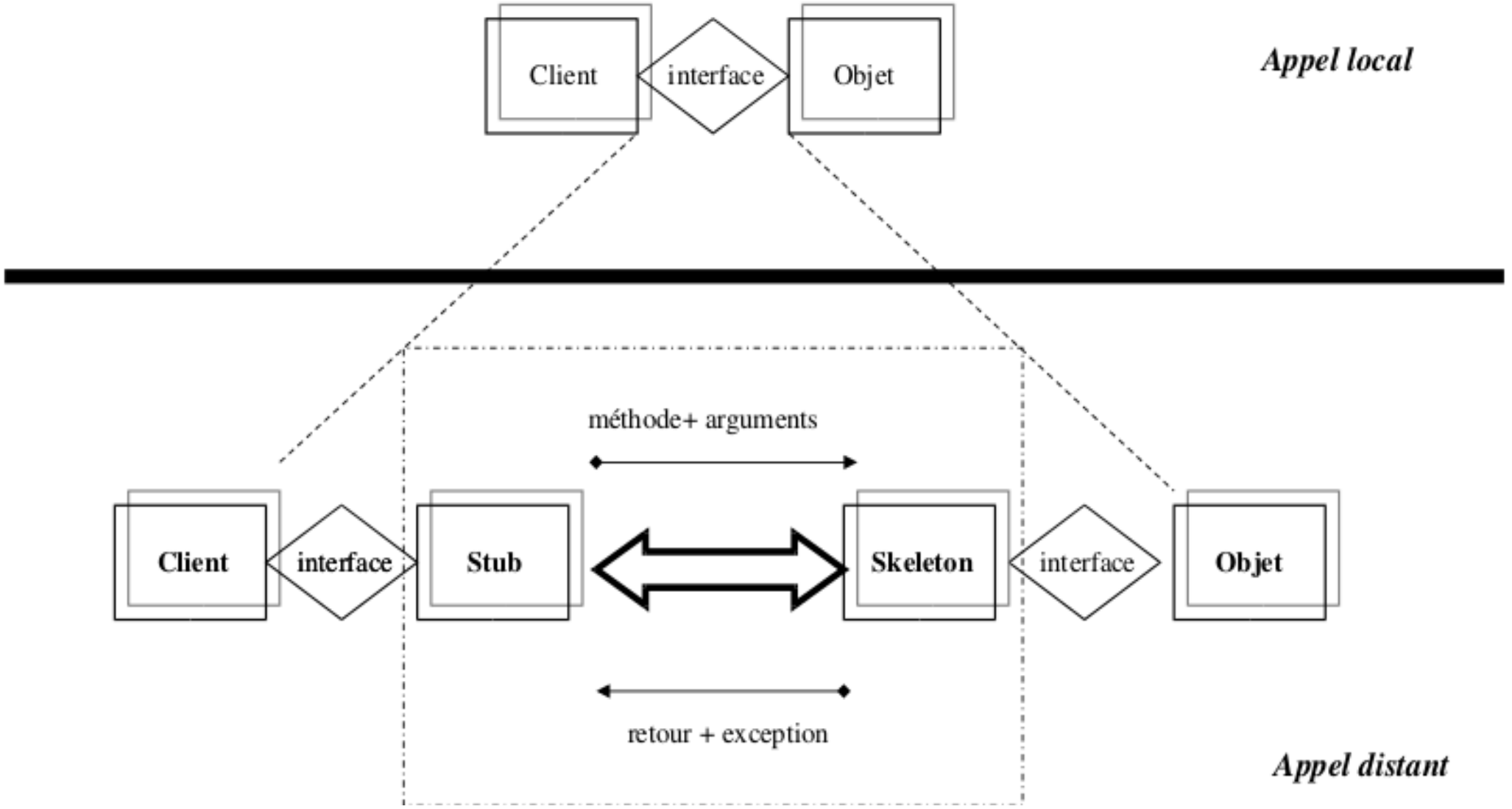
- Mécanisme qui permet l'appel de méthodes entre objets Java qui s'exécutent éventuellement sur des JVM distinctes ;
- L 'appel peut se faire sur la même machine ou bien sur des machines connectées sur un réseau ;
- Utilise les sockets ;
- Les échanges respectent un protocole propriétaire :

  Remote Method Protocol ;
- RMI repose sur les classes de sérialisation.

Architecture



Appel local versus Appel à distance



Les amorces (Stub/Skeleton)

- Elles assurent le rôle d'adaptateurs pour le transport des appels distants
- Elles réalisent les appels sur la couche réseau
- Elles réalisent l'assemblage et le désassemblage des paramètres (marshalling, unmarshalling)
- Une référence d'objets distribué correspond à une référence d'amorce
- Les amorces sont créées par le générateur rmic.

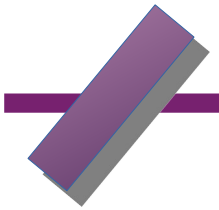
Les Stubs

- Représentants locaux de l'objet distribué ;
- Initient une connexion avec la JVM distante en transmettant

l'invocation distante à la couche des références d'objets ;

- Assemblent les paramètres pour leur transfert à la JVM distante ;
- Attendent les résultats de l'invocation distante ;
- Désassemblent la valeur ou l'exception renvoyée ;
- Renvoient la valeur à l'appelant ;
- S'appuient sur la sérialisation.

Les squelettes



- Désassemblent les paramètres pour la méthode distante ;
- Font appel à la méthode demandée ;
- Assemblage du résultat (valeur renvoyée ou exception)

à destination de l'appelant.

La couche des références d'objets
Remote Reference Layer

• Permet d'obtenir une référence d'objet distribué à partir de la référence locale au stub ;

• Cette fonction est assurée grâce à un service de noms rmiregister (qui possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants) ;

• Un unique rmiregister par JVM ;

• rmiregister s'exécute sur chaque machine hébergeant des objets distants ;

• rmiregister accepte des demandes de service sur le port 1099;

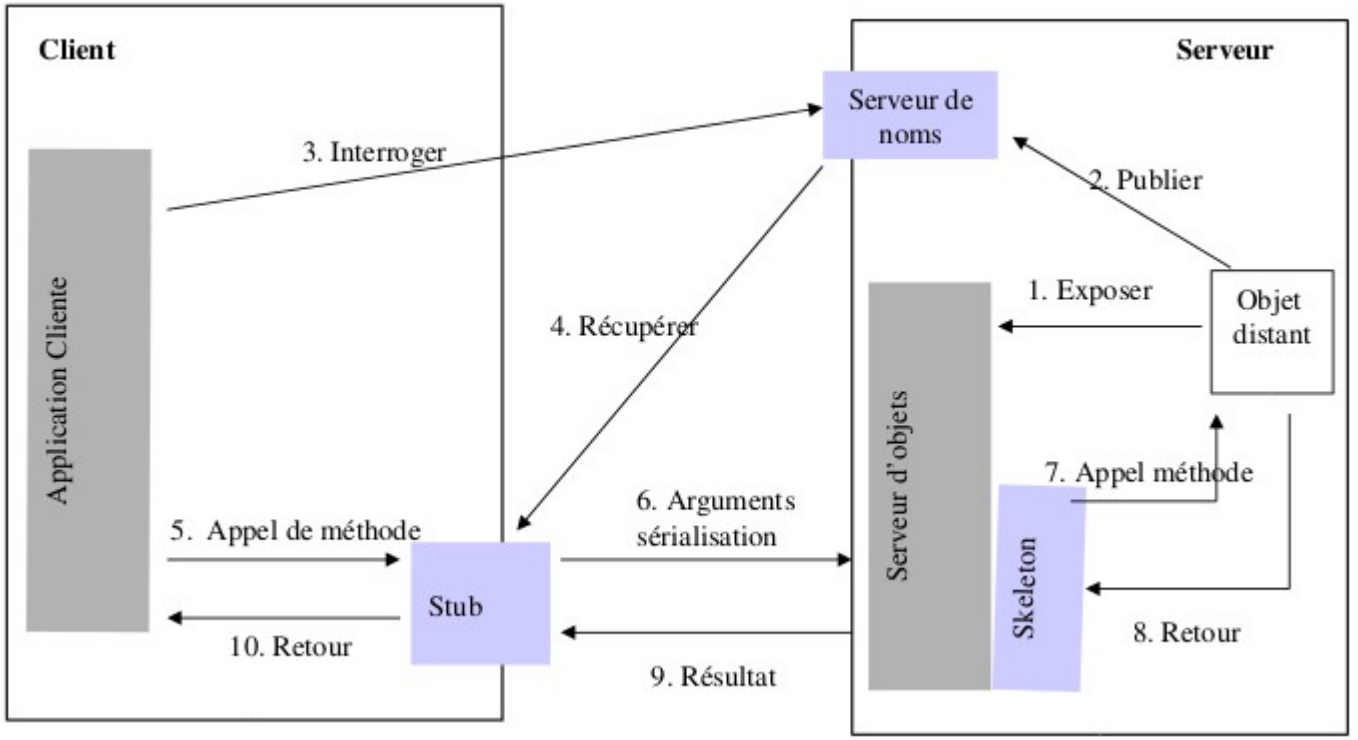
La couche transport

• réalise les connexions réseau basées sur les flux entre les JVM

•emploie un protocole de communication propriétaire (JRMP: Java Remote Method Invocation) basé sur TCP/IP

•Le protocole JRMP a été modifié afin de supprimer la nécessité des squelettes car depuis la version 1.2 de Java, une même classe skeleton générique est partagée par tous les objets distants.

Etapes d'un appel de méthode distante



Interface de l'objet distant

- Elle est partagée par le client et le serveur ;

- Elle décrit les caractéristiques de l'objet ;

- Elle étend l'interface Remote définie dans java.rmi.

Toutes les méthodes de cette interface peuvent déclencher une exception du type RemoteException.

Cette exception est levée :

- si connexion refusée à l'hôte distant

- ou bien si l'objet n'existe plus,

- ou encore s'il y a un problème lors de l'assemblage ou le désassemblage.

Interface de l'objet distant

```
import java.rmi.*;
public interface HelloInterface extends Remote
{
public String say() throws RemoteException;
}
```

Interface de l'objet distant

```
import java.rmi.*;
public interface HelloInterface extends Remote
{
public String say() throws RemoteException;
}
```

Hello World: implémentation de l'objet distant

```
import java.rmi.*;
import java.rmi.server.*;
public class Hello extends UnicastRemoteObject
implements HelloInterface {
private String message;
public Hello(String msg) throws RemoteException {
message = msg; }
```

Hello World: implémentation de l'objet distant

```
public String say() throws RemoteException {
return message;
}
}
```

Compilation

javac HelloInterface.java
javac Hello.java
(cr´ee HelloInterface.class et Hello.class)

Client

```
import java.rmi.*;
public class HelloClient {
public static void main(String[] argv) {
try {
HelloInterface hello =
(HelloInterface) Naming.lookup
("//cher.polytechnique.fr/Service");
System.out.println(hello.say());
(le serveur est suppos´e toujours ˆetre sur cher, voir plus loin pour
d'autres m´ethodes)
} catch(Exception e) {
System.out.println("HelloClient exception: "+e);
}
}
}
```

Serveur

```
import java.rmi.*;
public class HelloServer {
public static void main(String[] argv) {
try {
Naming.rebind("Service",new Hello("Hello, world!"));
System.out.println("Hello Server is ready.");
} catch(Exception e) {
System.out.println("Hello Server failed: "+e);
}
}
}
```

La classe de l'objet distant

La classe implémentant l'interface doit utiliser la classe UnicastRemoteObject

- Cette classe sert à la communication via TCP
- L'objet de la classe implémentant l'interface doit « s'exporter » pour accepter des connexions de clients

-Création et utilisation des ressources, éléments nécessaires à la communication via sockets TCP
Unicast : l'objet de la classe implémentant l'interface n'existe qu'un en seul exemplaire sur une seule machine
L'objet meurt avec la fin de l'exécution du serveur qui le lance

- Deux modes
  -La classe étend directement la classe UnicastRemoteObject
    -Le constructeur par défaut appele super();

-Pas de spécialisation et le constructeur doit exécuter l'instruction suivante :
UnicastRemoteObject.exportObject(this);

- Le constructeur par défaut doit préciser dans sa signature qu'il peut lever RemoteException

Registry

- **Une partie client doit pouvoir récupérer une référence sur l'objet distant implémentant l'interface**
  - **Utilise les services du « registry » (registre, annuaire ...)**
  - **Le registry est lancé à part à coté des applis Java**
  - **Un objet accède au registry via la classe Naming**

Registry

- **Une partie client doit pouvoir récupérer une référence sur l'objet distant implémentant l'interface**
  - **Utilise les services du « registry » (registre, annuaire ...)**
  - **Le registry est lancé à part à coté des applis Java**
  - **Un objet accède au registry via la classe Naming**

Identification d'un objet distant

Via une URL de la forme rmi://hote:port/nomObj

hote : nom de la machine distante (sur laquelle tourne un regis

port : port sur lequel écoute le registry

nomObj : nom donné à un objet offrant des opérations

Si pas précision du port : port par défaut utilisé par le registr

Si pas précision de l'hôte : registry local par défaut (localhost)

Compilation et démarrage du serveur

javac HelloClient.java

javac HelloServer.java

D´emarrer le serveur de noms:

rmiregistry &

(attendre un minimum)

D´emarrer le serveur (Hello):

java HelloServer &

(attendre un peu)

demarrage´ des clients et ex e´cution

(ici en local)

> Hello Server is ready.

> java HelloClient

Hello, world!

RMI: la pratique en résumé

-Écrire l'interface distante

-Écrire le code de l'objet distant (une seule classe ou une par item ci-après)

-Implémenter l'interface (et étendre UnicastRemoteObject)

-Ajouter le code pour le registry (en général dans le main ou le constructeur)

-Compiler

-Générer les stub et skeleton (optionnel)

-Écrire le client

-Obtenir une référence vers l'objet distant

-Utiliser ses méthodes distantes

-Compiler

-Exécuter:

-Démarrer le rmiregistry PUIS Démarrer le serveur

-Démarrer le client

Debugger :)