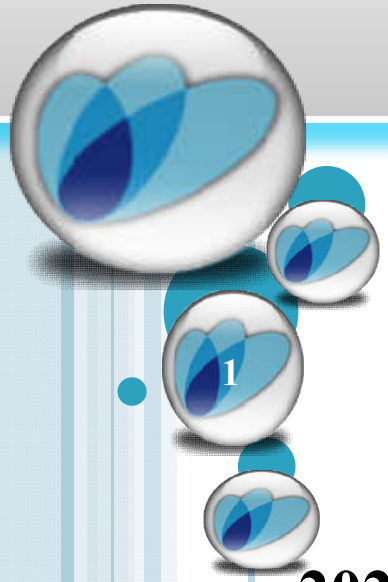
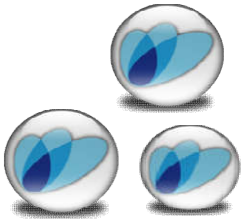
Cours Construction d'Applications Réparties (CAR)

1ère Année Master Génie Logiciel



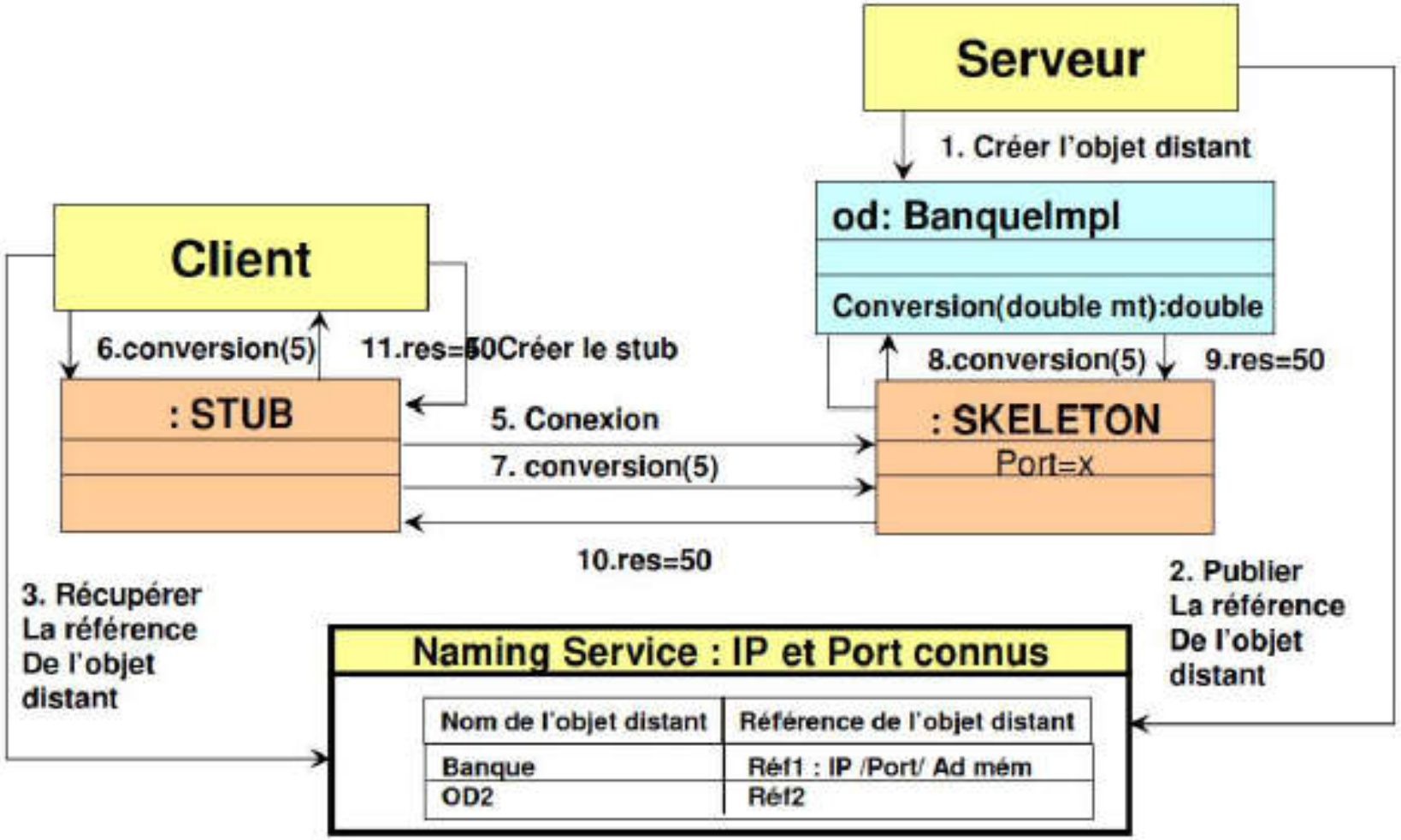


2020-2021

Mme Z.LAAREDJ

JAVA RMI

Architecture type



RMI

- RMI est apparu dés la version 1.1 du JDK dont le but est de créer un modèle objet distribué Java

- Avec RMI, les méthodes de certains objets (appelés objets distants) peuvent être invoquées depuis des JVM différents (espaces d'adressages distincts)

- En effet, RMI assure la communication entre le serveur et le client via TCP/IP et de manière transparente pour le développeur

- Il utilise des mécanismes et des protocoles définis et standardisés tels que les sockets et les RMP (Remote Method Protocol)

- Le développeur n'a donc pas à se soucier des problèmes de niveaux inférieurs spécifiques aux technologies réseaux

Rappel sur les interfaces

- Dans java, une interface est une classe abstraite qui ne contient que des méthodes abstraites

- Dans java une classe peut hériter d'une seule classe et peut implémenter plusieurs interfaces

- Implémenter une interface signifie qu'il faut redéfinir toutes les méthodes déclarées dans l'interface

Rappel sur les interfaces

➢ Exemple

Un exemple d'interface

```
public interface IBanque{
 public void verser(float mt);
}
```

Un exemple d'implémentation

```
public class BanqueImpl implements IBanque{
  private float solde ;

  public void verser(float mt){
   solde=solde+mt;
 }
}
```

Architecture de RMI

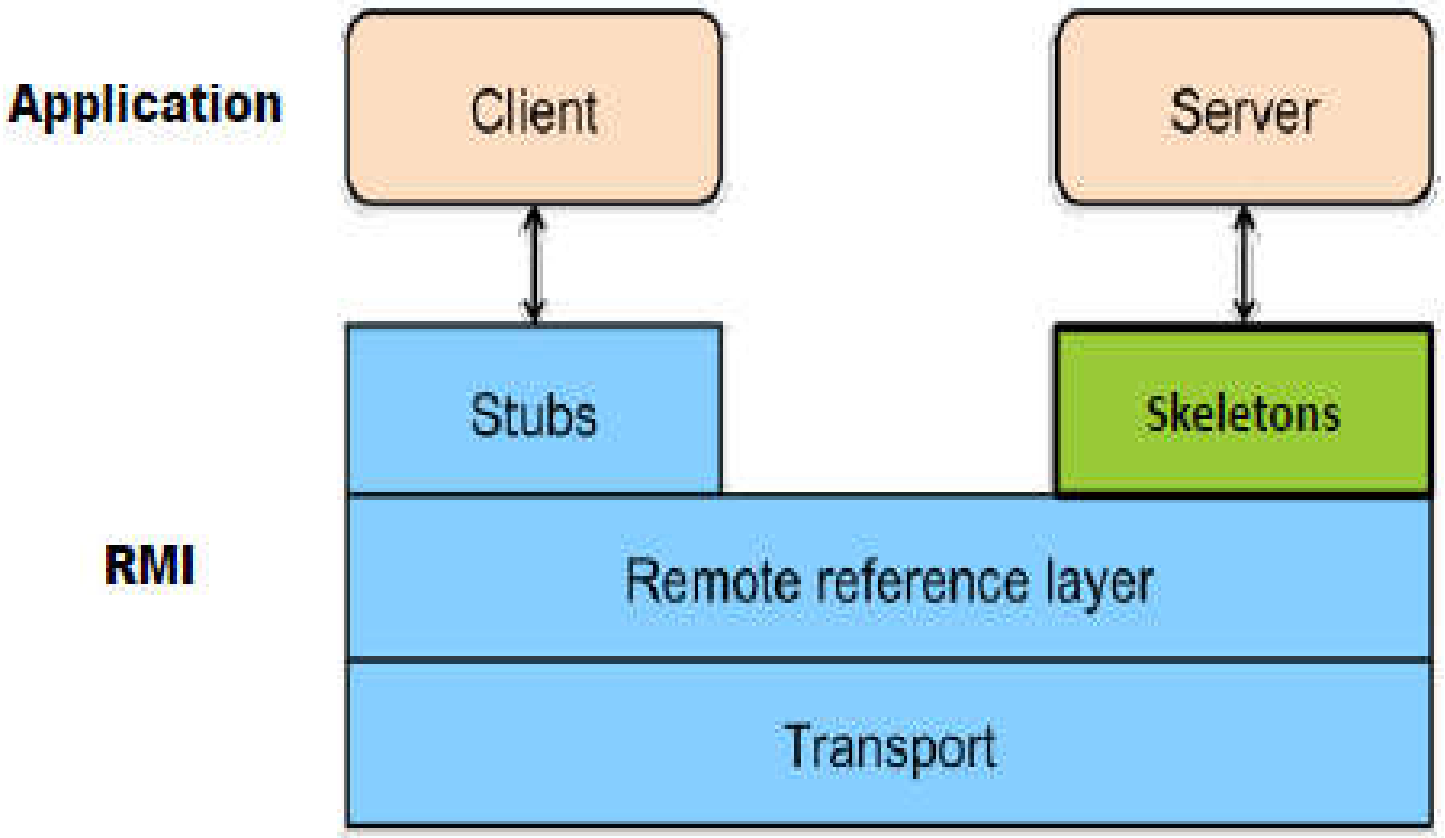
Interfaces

- Les interfaces est le cœur de RMI
- L'architecture RMI est basé sur le principe que la définition du comportement et l'exécution de ce comportement sont des concepts séparés
- La définition d'un service distant est codé en utilisant une interface Java
- L'implémentation de ce service distant est codée dans une classe

Couches RMI

- RMI est essentiellement construit sur une abstraction de trois couches
  - Stubs et Skeletons (Souche et Squlette)
  - Remote Reference Layer (Couche de référencement distante)
  - Couche Transport

Couches RMI


Application
Client
Server
Stubs
Skeletons
RMI
Remote reference layer
Transport

Couches RMI

Stubs et Skeletons

- Les stubs sont des classes placées dans la machine du client
- Lorsque notre client fera appel à une méthode distante, cet appel sera transféré au stub
- Le stub est donc un relais du coté de client. Il devra donc être placé sur la machine cliente
- Le stub est le représentant local de l'objet distant
- Il emballe les arguments de la méthode distante et les envoie dans un flux de données vers le service RMI distant
- D'autre part, il déballe la valeur ou l'objet de retour de la méthode distante

Couches RMI

Stubs et Skeletons

- Il communique avec l'objet distant par l'intermédiaire d'un skeleton
- Le skeleton est lui aussi un relais mais du coté serveur. Il devra être placé sur la machine du serveur
- Il déballe les paramètres de la méthode distante, les transmet à l'objet local et emballe les valeurs de retours à renvoyer au client
- Les stubs et les skeletons sont donc des intermédiaires entre le client et le serveur qui gèrent le transfert distant des données

Couches RMI

Stubs et Skeletons

- On utilise le compilateur rmic pou la génération des stubs et des skeletons avec JDK
- Depuis la version 2 de Java, le skeleton n'existe plus
- Seul le stub est nécessaire du coté client mais aussi du coté serveur

Couches RMI

Remote Reference Layer

- La deuxième couche de l'architecture RMI
- Ce service est assuré par le lancement du programme **rmiregistry** du coté serveur
- Le serveur doit enregistrer la référence de chaque objet distant dans le service rmiregistry en attribuant un nom à cet objet distant
- Du coté client, cette couche permet l'obtention d'une référence de l'objet distant à partir de la référence locale (le stub) en utilisant son nom rmiregistry.

Couches RMI

Couche transport

- La couche transport est basée sur les connexions TCP/IP entre les machines
- Elle fournit la connectivité de base entre les 2 JVM
- Elle fournit des stratégies pour passer les firewalls
- Elle suit les connexions en cours
- Elle construit une table des objets distants disponibles
- Elle écoute et répond aux invocations
- Cette couche utilise les classes Socket et ServerSocket et utilise un protocole propriétaire RMP (Remote Method Protocol)

Démarche RMI

- Créer les interfaces des objets distants
- Créer les implémentations des objets distant
- Générer les stubs et skeletons
- Créer le serveur RMI
- Créer le client RMI
- Déploiement Lancement
  - Lancer l'annuaire RMIREGISTRY
  - Lancer le serveur
  - Lancer le client

Démarche RMI

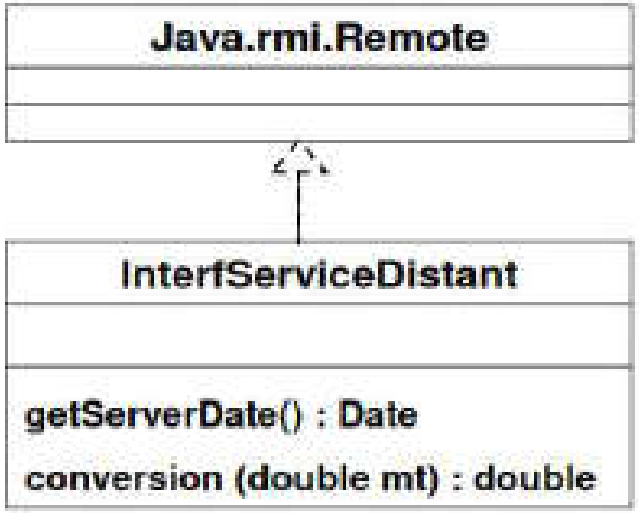
Exemple

Supposant qu'on veut créer un serveur RMI qui crée un objet qui offre les services distants suivant à un client RMI

- Convertir un montant de l'euro
- Envoyer au client la date du serveur

Démarche RMI

1. Interface de l'objet distant




```
import java.rmi.Remote;import java.rmi.RemoteException;
import java.util.Date;
public interface InterfServiceDistant extends Remote {
  public Date getServerDate() throws RemoteException;
  public double convertEuroToDH(double montant) throws
RemoteException;
}
```

Démarche RMI

Interfaces

- La première étape consiste à créer une interface distante qui décrit les méthodes que le client pourra invoquer à distante
- Pour que ses méthodes soient accessibles par le client, cette interface doit hériter de l'interface **Remote**
- Toutes les méthodes utilisables à distante doivent pouvoir lever les exceptions de type **RemoteException** qui sont spécifiques à l'appel distant
- Cette interface devra être placée sur les deux machines (serveur et client)

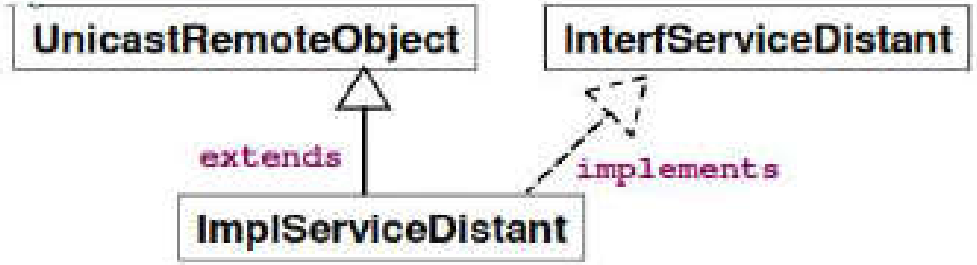
Démarche RMI

➢ Implémentation

- Il faut maintenant implémenter cette interface distante dans une classe. Par convention, le nom de cette classe aura pour suffixe Impl
- Notre classe doit hériter de la classe java.rmi.server.RemoteObject ou de l'une de ses sous-classes
- La plus facile à utiliser étant java.rmi.server.UnicicastRemoteObject
- C'est dans cette classe que nous allons définit le corps des méthodes distantes que pourront utiliser nos clients

Démarche RMI

2.Implémentation de l'objet distant




```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Date;

public class ImplServiceDistant extends UnicastRemoteObject implements
    InterfServiceDistant {
     public ImplServiceDistant()throws RemoteException{

     }
public Date getServerDate() throws RemoteException {
return new Date();
}
  public double convertEuroToDH(double montant) throws RemoteException {
return montant*11.3;
}
}
```

Démarche RMI

Stubs et Skeletons

- Si l'implémentation est crée, les stubs et skeletons peuvent être générés par l'utilitaire rmic en écrivant la commande

Rmic NOM_IMPLEMENTATION

Démarche RMI

Naming Service

- Les clients trouvent les services distants en utilisant un service d'annuaire activé sur un hôte connu avec un numéro de port connu
- RMI peut utiliser plusieurs services d'annuaire, y compris Java Naming and Directory Interface (JNDI)
- Il inclut lui-même un service simple appelé (rmicegistry).
- Le registre est exécuté sur chaque machine qui héberge des objets distants (les serveurs) et accepte les requêtes pour ces services, par défaut sur le port 1099

Utilisation de RMI

- Un serveur crée un service distant en créant d'abord un objet local qui implémente ce service
- Ensuite, il exporte cet objet vers RMI
- Quand l'objet est exporté, RMI crée un service d'écoute qui attend qu'un client se connecte et envoie des requêtes au service
- Après exportation, le serveur enregistre cet objet dans le registre de RMI sous un nom public qui devient accessible de l'extérieur
- Le client peut alors consulter le registre distant pour obtenir des références à des objets distants

Le serveur

- Notre serveur doit enregistrer auprès du registre RMI ‘objet local dont les méthodes seront disponibles à distante
- Cela se fait grâce à la méthode statique **bind()** ou **rebind()** de la classe Naming
- Cette méthode permet d’associer (enregister) l’objet local avec un synonye dans le registre RMI
- L’objet devient alors disponible par le client

```
ObjetDistantImpl od = new ObjetDistantImpl();
Naming.rebind("rmi://localhost:1099/NOM_Service",od);
```

Le serveur

➢ Code du serveur RMI

```
import java.rmi.Naming;
public class ServeurRMI {
   public static void main(String[] args) {
        try {
// Créer l'objet distant
ImplServiceDistant od=new ImplServiceDistant();
// Publier sa référence dans l'annuaire
Naming.rebind("rmi://localhost:1099/SD",od);
        } catch (Exception e) {
                e.printStackTrace();
        }
   }}
```

Le client

- Le client peut obtenir une référence à un objet distant par l'utilisation de la méthode statique **lookup()** de la classe Naming
- La méthode lookup() sert au client pour interroger un registre et récupérer un objet distant
- Elle retourne une référence à l'objet distant
- La valeur retournée est de type Remote. Il est donc nécessaire de caster cet objet en l'interface distante implémentée par l'objet distant

Le client

➢ Code du client RMI

```
import java.rmi.Naming;
public class ClientRMI {
   public static void main(String[] args) {
        try {
InterfServiceDistant stub=
   (InterfServiceDistant)Naming.lookup("rmi://localhost:1099/SD");
System.out.println("Date du serveur:"+stub.getServerDate());
System.out.println("35 euro vaut "+stub.convertEuroToDH(35));
        } catch (Exception e) {
                e.printStackTrace();
        }}}
```

Le lancement

- Il est maintenant possible de lancer l'application. Cela va nécessiter l'utilisation de trois consoles
  - La première sera utilisée pour activer le registre. Pour cela, vous devez exécuter l'utilitaire **rmiregistry**
  - Dans une deuxième console, exécuter le serveur. Celui-ci va charger l'implémentation en mémoire, enregistrer cette référence dans le registre et attendre une connexion cliente
  - Vous pouvez enfin exécuter le client dans une troisième console

Le lancement

- Même si vous exécuter le client et le serveur sur la même machine, RMI utilisera la pile réseau et TCT/IP pour communiquer entre les JVM.