Informatique Repartie

Chapitre 3 : RMI

Cecilia Zanni-Merk

cecilia.zanni-merk@insa-rouen.fr

Bureau BO B R1 04

Basé sur le cours de M Alexandre Pauchet, INSA Rouen Normandie, 2016

Références

- Le cours de M Pauchet sur Moodle
- Architectures réparties en Java de Annick Fron ISBN 9782100738700. Ed Dunod
- The JAVA tutorials by Oracle https://docs.oracle.com/javase/tutorial/rmi/

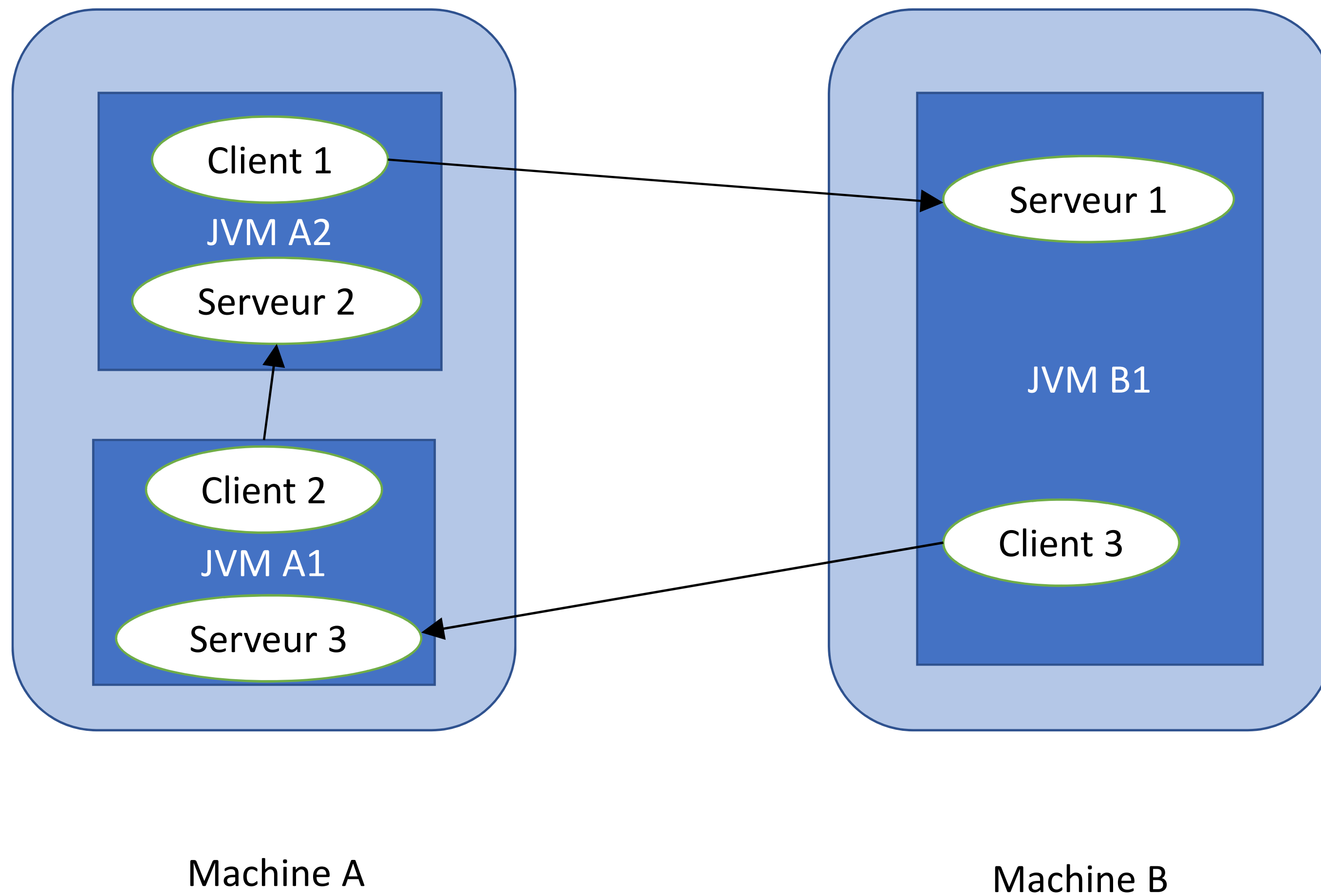
Présentation

- RMI (*Remote Method Invocation*) est un moyen simple pour développer des programmes client-serveur.
- RMI s'appuie sur un protocole de type RPC (*Remote Procedure Call*)

Présentation

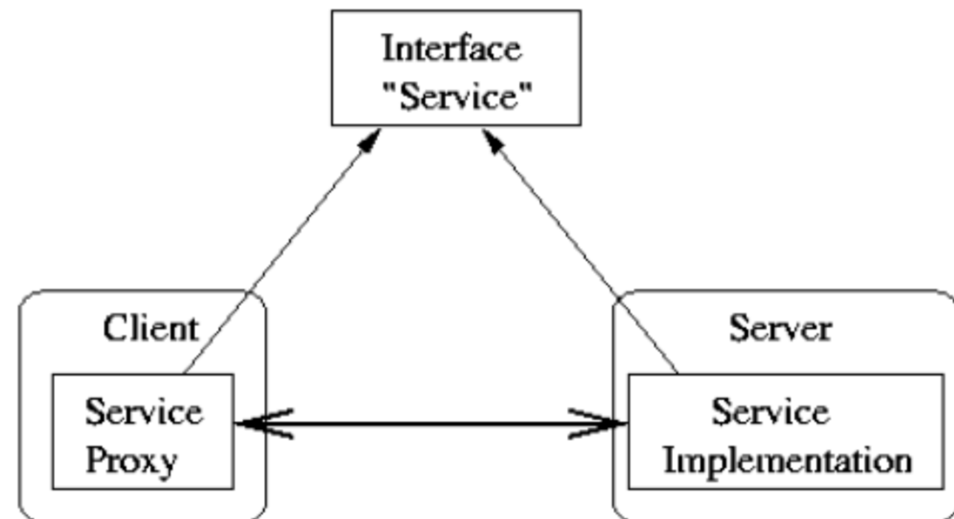
- RMI (*Remote Method Invocation*) est un moyen simple pour développer des programmes client-serveur.
- RMI s’appuie sur un protocole de type RPC (*Remote Procedure Call*)
- L'objectif est de permettre à des applications clientes (s'exécutant localement) d'invoquer des méthodes sur des objets distants, c'est-à-dire localisés dans une autre application (dans une autre JVM de la même machine physique ou sur une autre machine accessible via Internet) communément appelée *serveur*.
- *Les serveurs et les clients sont des objets*

Présentation



Principe et Architecture

- Séparation de la définition d'un service et de son interface.
  - interface du service
  - implémentation du service
  - proxy vers le service



Principe et Architecture

- Ainsi, une application s'exécutant sur une machine M1 peut créer un objet et le rendre accessible à d'autres applications : cette application (et la machine M1) joue donc le rôle de serveur. Les autres applications manipulant un tel objet sont des clients.

Principe et Architecture

- Ainsi, une application s'exécutant sur une machine M1 peut créer un objet et le rendre accessible à d'autres applications : cette application (et la machine M1) joue donc le rôle de serveur. Les autres applications manipulant un tel objet sont des clients.
- Pour manipuler un objet distant, un client récupère sur sa machine une représentation de l'objet appelé **proxy / talon / souche / stub**.

Principe et Architecture

- Le proxy (ou objet de communication) est un objet qui va faire le lien entre une interface locale et l'objet distant : c'est via ce talon que le client pourra invoquer des méthodes sur l'objet distant. Une telle invocation sera transmise au serveur (le protocole TCP est utilisé) afin d'en réaliser l'exécution.

Principe et Architecture

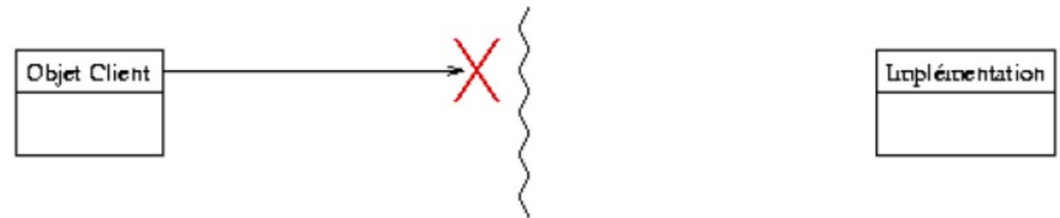
- Le proxy (ou objet de communication) est un objet qui va faire le lien entre une interface locale et l'objet distant : c'est via ce talon que le client pourra invoquer des méthodes sur l'objet distant. Une telle invocation sera transmise au serveur (le protocole TCP est utilisé) afin d'en réaliser l'exécution.
- Du côté du serveur un **skeleton / squelette** a en charge la réception des invocations distantes, de leur réalisation et de l'envoi des résultats.

Souche et Squelette

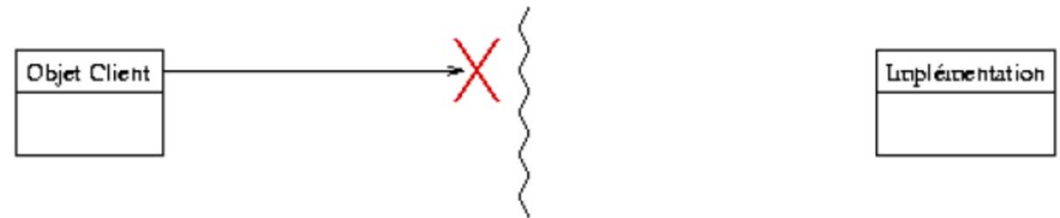
- Le client ne peut pas invoquer directement une méthode de l'implémentation par le réseau

Souche et Squelette

- Le client ne peut pas invoquer directement une méthode de l'implémentation par le réseau



Souche et Squelette



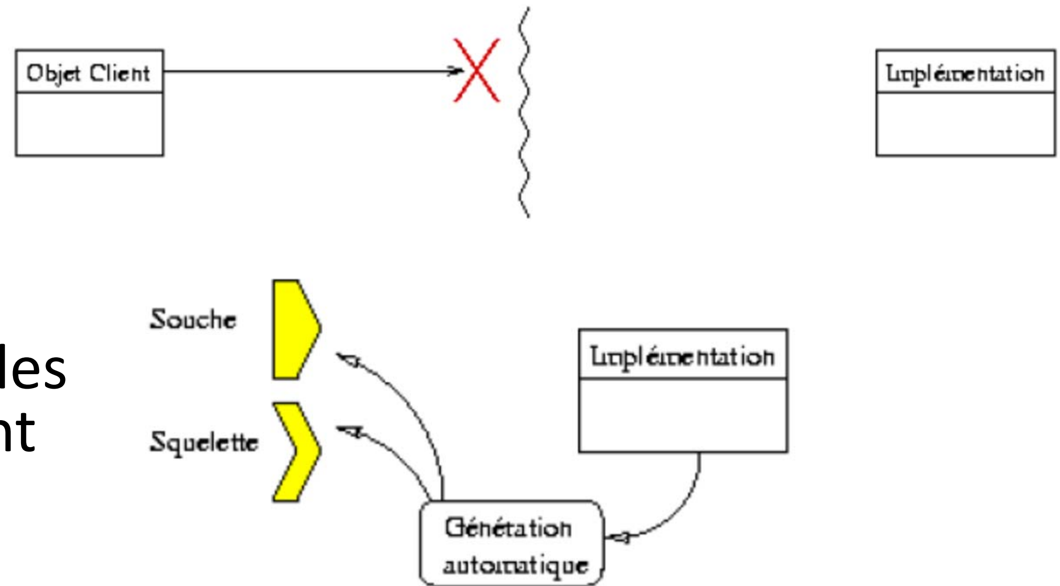
- Le client ne peut pas invoquer directement une méthode de l'implémentation par le réseau

- Une souche et un squelette capables de communiquer par le réseau sont automatiquement générés

Souche et Squelette

- Le client ne peut pas invoquer directement une méthode de l'implémentation par le réseau

- Une souche et un squelette capables de communiquer par le réseau sont automatiquement générés

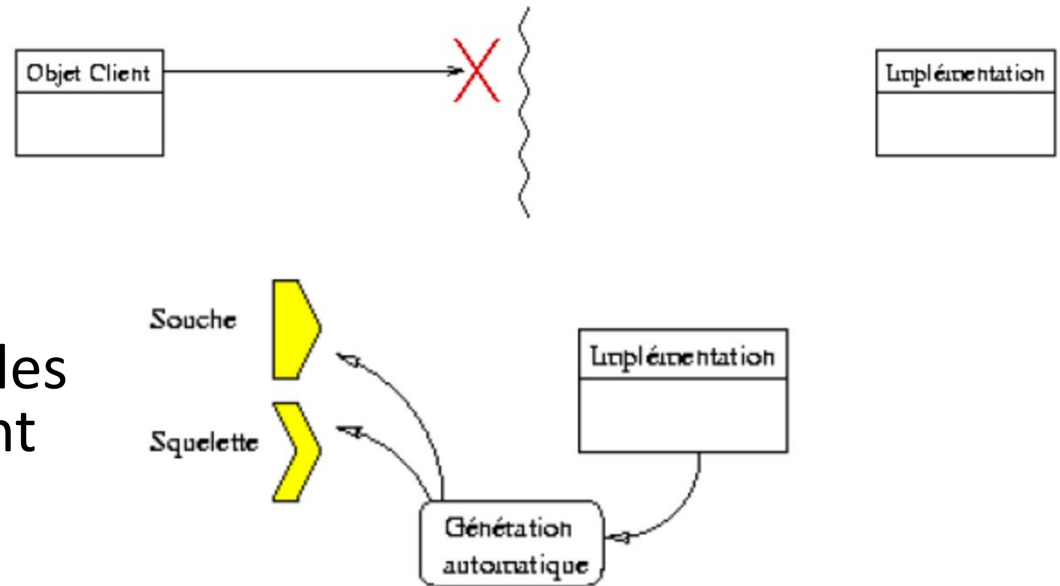


Souche et Squelette

- Le client ne peut pas invoquer directement une méthode de l'implémentation par le réseau

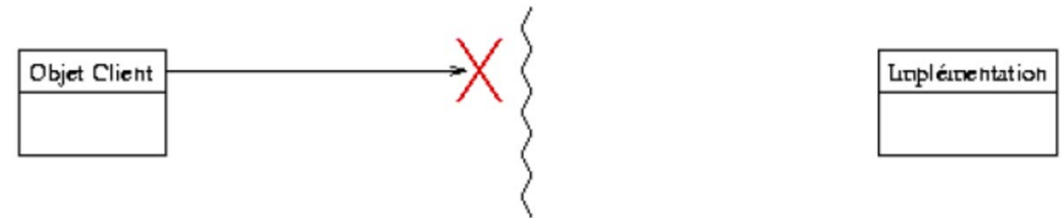
- Une souche et un squelette capables de communiquer par le réseau sont automatiquement générés

- La souche est déployée chez le client et le squelette chez le serveur

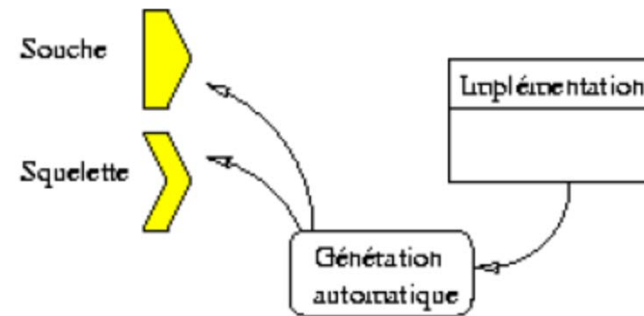


Souche et Squelette

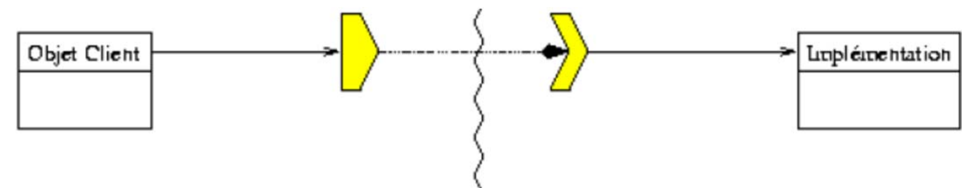
- Le client ne peut pas invoquer directement une méthode de l'implémentation par le réseau



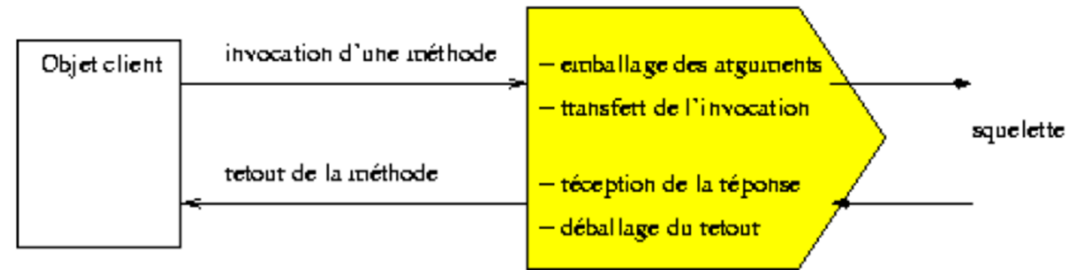
- Une souche et un squelette capables de communiquer par le réseau sont automatiquement générés



- La souche est déployée chez le client et le squelette chez le serveur

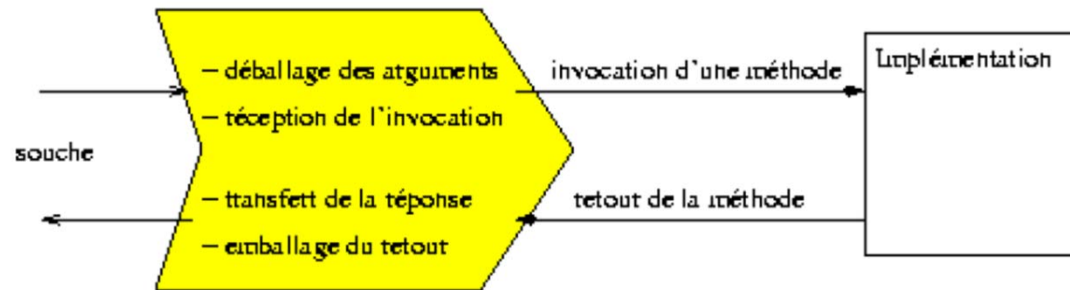


La souche



- Présente du côté client
- Représente un objet distant
- Convertit les arguments en un format transmissible via le réseau (marshalling)
- Reconstruit les valeurs de retour a partir de données reçues par le réseau (unmarshalling)
- Générée automatiquement

Le squelette

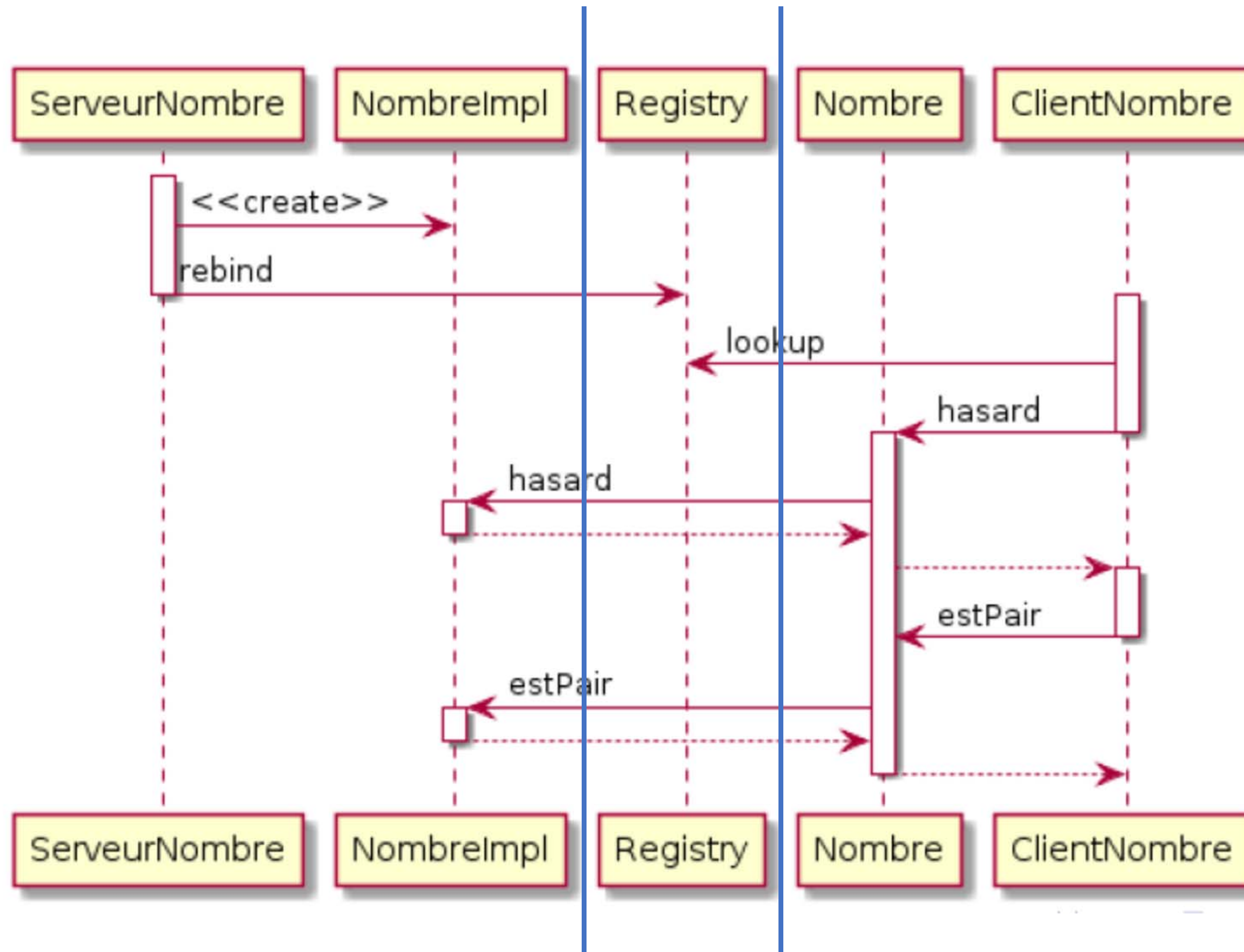


- Présent du côté serveur
- Invoque des méthodes sur l'objet local référence pour le compte d'une souche
- Convertit les valeurs de retour en un format transmissible via le réseau (unmarshalling)
- Reconstruit les arguments a partir de données reçues par le réseau (marshalling)
- Généré automatiquement

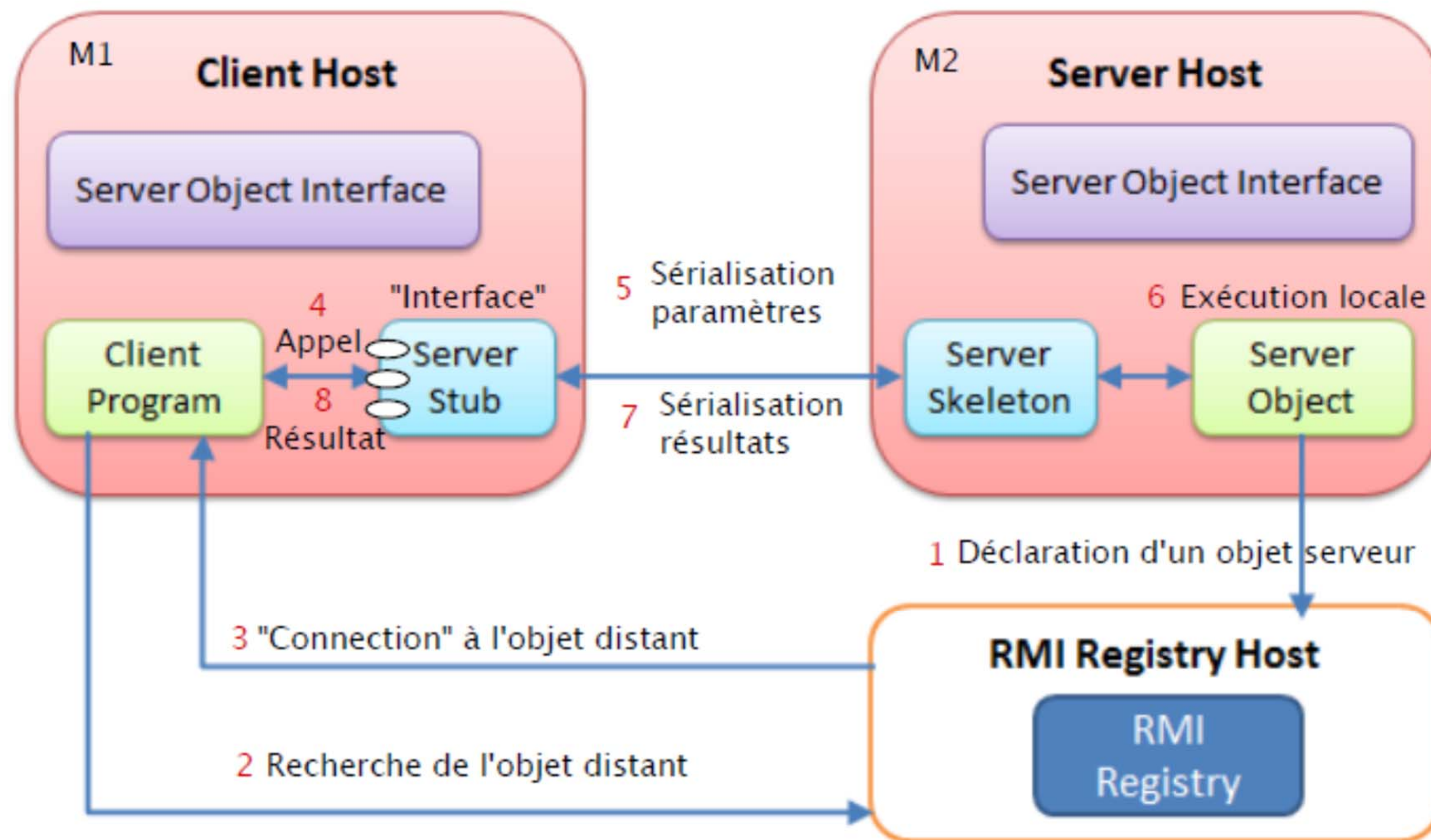
Impact sur le Génie Logiciel

- L'écriture d'un programme en RMI nécessite une conception préliminaire pour déterminer où se situent les frontières entre le programme distant et le programme local, et quels sont les objets serveurs de part et d'autre.
- Tous les échanges entre objets clients et serveurs doivent être analysés pour déterminer les méthodes qui sont échangées, et les arguments qui transitent entre les deux → diagramme de séquence UML pour analyser le flux entre les objets

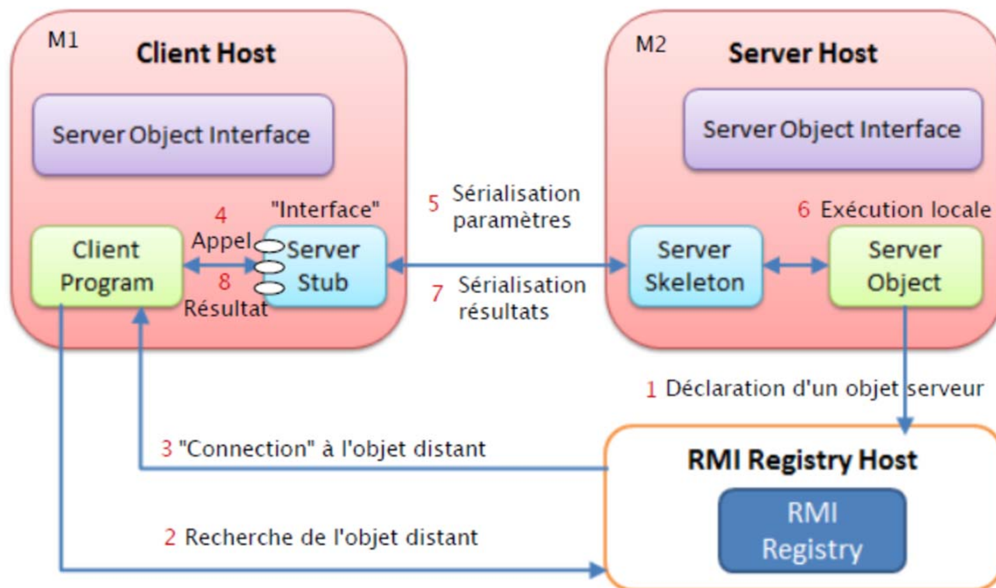
Impact sur le Génie Logiciel



Mise en œuvre

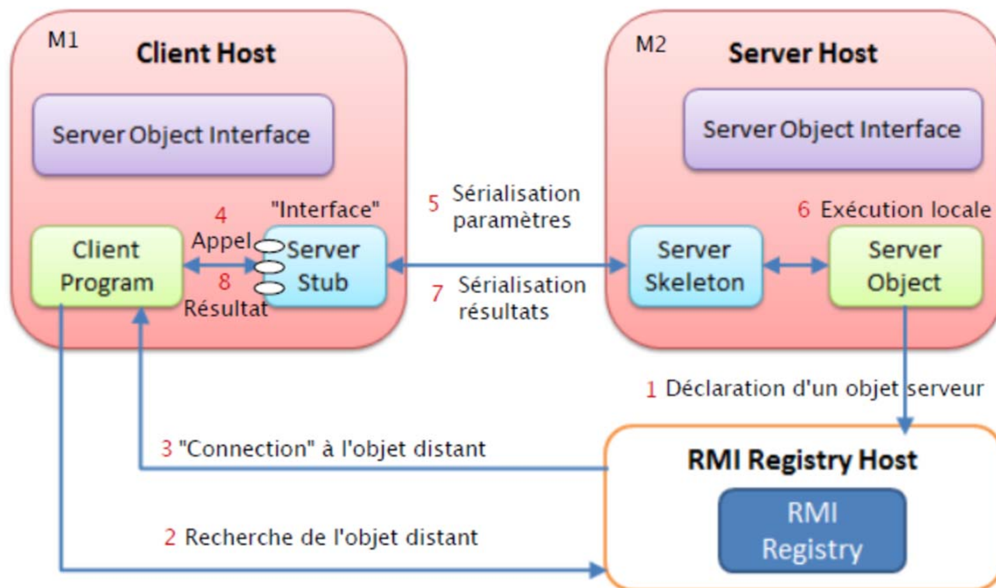


1



Mise en œuvre

1. Dans un premier temps, le serveur sur la machine M2 va déclarer le service qu'il est prêt à rendre → Serveur de noms : `rmiregistry`
2. Le client sur la machine M1 va demander au serveur de noms de résoudre le nom de la machine M2
3. Création d'une interface connectée sur l'objet distant (proxy)
4. Le client sur la machine M1 appelle une méthode sur cette interface



Mise en œuvre

5. Le proxy (stub) sur la machine M1
   a) emballe l'identifiant de la méthode et ses arguments (sérialisation) ;
   b) la requête est transmise sur le réseau ;
6. Le squelette (skeleton) sur la machine M2
   a) reçoit et déballe le message (désérialisation) ;
   b) appelle la méthode demandée ;
   c) reçoit le retour de la méthode ;
7. Le squelette
   a) emballe ce résultat ;
   b) transmet le résultat vers le proxy (stub) sur la machine M1;
8. Le proxy sur la machine M1
   a) reçoit et déballe le message ;
   b) retourne le résultat comme une méthode ordinaire.

JAVA RMI

- Protocole et outil d’invocation de méthodes à distance proposés par Sun depuis 1996
- À partir du JDK 1.1, RMI fait partie intégrante de Java :
  - Un ensemble de classes dans les packages :
    - `java.rmi`
    - `java.rmi.server`
    - `java.rmi.registry`
    - `java.rmi.dgc`
    - `java.rmi.activation`
  - Un ”serveur” :
    - `rmiregistry`
  - Protocole de transport utilisé : JRMP (Java Remote Method Protocol)

JAVA RMI

- Produit gratuit
- Mono-langage mais multi-plateformes
- Orienté-objet
- Invocation de méthodes synchrones
- Passage de paramètres des invocations distantes :
  - Type simple : passage par valeur
  - Instance d’une classe qui implémente `Serializable` : passage par valeur
  - Instance d’une classe qui implémente `Remote` et référencée comme objet distribué: c’est un stub qui est envoyé
  - Sinon une erreur est produite
- Service d’annuaire (`rmiregistry`)
- Port par défaut : 1099

Cycle de développement

- Le développement coté serveur se compose de :
  - La définition d'une interface qui contient les méthodes qui peuvent être appelées à distance (hérite de `java.rmi.Remote`)
  - L'écriture d'une classe qui implémente cette interface (hérite de `java.rmi.server.RemoteObject`, par exemple `java.rmi.server.UnicastRemoteObject`)
  - L'écriture d'une classe qui instanciera l'objet et l'enregistrera en lui affectant un nom dans le registre de noms RMI ou `rmiregistry`

Cycle de développement

- La création de la souche et du squelette anciennement par `rmic`, maintenant transparente

- Le développement côté client se compose de :
  - L'obtention d'une référence sur l'objet distant à partir de son nom (par `java.rmi.registry.Registry`)
  - L'appel à la méthode à partir de cette référence

Exemple côte serveur

- Définition de l'interface d'accès aux objets adressables à distance

Client.java | Serveur.java | Message.java | MessageImpl.java

```
1  import java.rmi.Remote ;
2  import java.rmi.RemoteException ;
3
4  public interface Message extends Remote {
5      public String messageDistant ( ) throws RemoteException ;
6  }
7
8
```

Exemple côte serveur

- Définition de la classe implantant le code qui réalisera réellement les opérations définies dans l'interface

Exemple côte serveur

- Écriture du serveur

Client.java Serveur.java Message.java MessageImpl.java

```java
import java.rmi.registry.LocateRegistry ;
import java.rmi.registry.Registry ;
import java.rmi.server.UnicastRemoteObject ;
import java.util.Arrays ;



public class Serveur {

    public static void main ( String [ ] args ) {
        try {

                int port = 1099;
                Message skeleton = (Message) UnicastRemoteObject.exportObject(new MessageImpl () , 0);
                System.out.println ( " Serveur pret " ) ;
                Registry registry = LocateRegistry.getRegistry (port);
                System.out.println ( " Service Message enregistre " ) ;
                if (! Arrays.asList(registry.list()).contains (" Message "))
                    registry.bind(" Message ", skeleton );
                else
                    registry.rebind (" Message ", skeleton );
        } catch ( Exception ex) {
            ex.printStackTrace ();
            }
    }
}
```

Service de nommage

- Sur le serveur, le registre de noms RMI doit s'exécuter avant de pouvoir enregistrer un objet ou obtenir une référence.

Service de nommage

- Sur le serveur, le registre de noms RMI doit s'exécuter avant de pouvoir enregistrer un objet ou obtenir une référence.
- Ce registre peut être lancé en tant qu'application fournie dans le JDK (`rmiregistry` sur la ligne de commande) ou être lancé dynamiquement dans la classe qui enregistre l'objet.
    - Le service de nommage est un objet RMI → `rmiregistry` est un serveur

Service de nommage

- Sur le serveur, le registre de noms RMI doit s'exécuter avant de pouvoir enregistrer un objet ou obtenir une référence.
- Ce registre peut être lancé en tant qu'application fournie dans le JDK (`rmiregistry` sur la ligne de commande) ou être lancé dynamiquement dans la classe qui enregistre l'objet.
  - Le service de nommage est un objet RMI → `rmiregistry` est un serveur
- **Ce lancement ne doit avoir lieu qu'une seule et unique fois.**
  - Le code pour exécuter le registre est la méthode `createRegistry()` de la classe `java.rmi.registry.LocateRegistry`.
  - Cette méthode attend en paramètre un numéro de port.

Service de nommage

- Le service de nommage offre plusieurs services :
  - `bind(), rebind(), unbind(), list(), lookup()`

Service de nommage

- Le service de nommage offre plusieurs services :
  - `bind(), rebind(), unbind(), list(), lookup()`
- Il peut être récupéré par un appel à la méthode statique :
  `Registry LocateRegistry.getRegistry([machine], [port]);`
  - Le port par défaut est 1099

Service de nommage

- Le service de nommage offre plusieurs services :
  - `bind(), rebind(), unbind(), list(), lookup()`
- Il peut être récupéré par un appel à la méthode statique :
  `Registry LocateRegistry.getRegistry([machine], [port]);`
  - Le port par défaut est 1099
- Pour des raisons de sécurité, les méthodes `bind(), rebind()` et `unbind()` ne sont accessibles que depuis la même machine

Service de nommage

- Le service de nommage offre plusieurs services :
  - `bind(), rebind(), unbind(), list(), lookup()`
- Il peut être récupéré par un appel à la méthode statique :
  `Registry LocateRegistry.getRegistry([machine], [port]);`
  - Le port par défaut est 1099
- Pour des raisons de sécurité, les méthodes `bind(), rebind()` et `unbind()` ne sont accessibles que depuis la même machine
- Les commandes `bind()` et `rebind()` permettent l'enregistrement de l'objet dans l'annuaire
  - `rebind()` permet d'écraser le nom d'une référence existante

Lancement du service de nommage

- La commande `rmiregistry` est fournie avec le JDK. Il faut la lancer en tâche de fond :
  - Sous Unix : rmiregistry&
  - Sous Windows : start rmiregistry

Lancement du service de nommage

- La commande `rmiregistry` est fournie avec le JDK. Il faut la lancer en tâche de fond :
  - Sous Unix : rmiregistry&
  - Sous Windows : start rmiregistry
- L'annuaire doit disposer du code de la souche sur son **classpath**, et il es donc généralement lancé à la racine du code du serveur

Lancement du service de nommage

- La commande `rmiregistry` est fournie avec le JDK. Il faut la lancer en tâche de fond :
  - Sous Unix : rmiregistry&
  - Sous Windows : start rmiregistry
- L'annuaire doit disposer du code de la souche sur son **classpath**, et il es donc généralement lancé à la racine du code du serveur
- Un service inscrit est accessible par une adresse du type `rmi ://<hostname>[ :port]/<servicename>`

Lancement du service de nommage

- La commande `rmiregistry` est fournie avec le JDK. Il faut la lancer en tâche de fond :
  - Sous Unix : rmiregistry&
  - Sous Windows : start rmiregistry
- L'annuaire doit disposer du code de la souche sur son **classpath**, et il es donc généralement lancé à la racine du code du serveur
- Un service inscrit est accessible par une adresse du type `rmi ://<hostname>[ :port]/<servicename>`
  - Par exemple : `rmi ://localhost/MonObjetNombre`

Exemple côte Client

Client.java | Serveur.java | Message.java | MessageImpl.java

```
1  import java.rmi.registry.LocateRegistry ;
2  import java.rmi.registry.Registry ;
3
4
5  public class Client {
6      public static void main ( String args []) {
7      String machine = "localhost";
8      int port = 1099;
9      try {
10         Registry registry = LocateRegistry.getRegistry ( machine , port );
11         Message obj = (Message) registry.lookup (" Message ");
12         System.out.println ( "Le client recoit : "+ obj.messageDistant ( ) ) ;
13
14     } catch ( Exception e) {
15         System.out.println (" Client exception : " +e);
16         }
17     }
18 }
19
```

Exemple côte Client

Client.java | Serveur.java | Message.java | MessageImpl.java

```java
import java.rmi.registry.LocateRegistry ;
import java.rmi.registry.Registry ;


public class Client {
    public static void main ( String args []) {
    String machine = "localhost";
    int port = 1099;
    try {
        Registry registry = LocateRegistry.getRegistry ( machine , port );
        Message obj = (Message) registry.lookup (" Message ");
        System.out.println ( "Le client recoit : "+ obj.messageDistant ( ) ) ;

    } catch ( Exception e) {
        System.out.println (" Client exception : " +e);
        }
    }
}
```

Exécution du tout

- Lancer le service de nommage
  - L’interface doit être accessible au serveur de nommage via le classpath ou le codebase (http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/codebase.html)
  - “You can think of your CLASSPATH as a "local codebase", because it is the list of places on disk from which you load local classes.”
- Lancer le serveur
  - L’interface et l’implémentation doivent être accessibles via le classpath ou le codebase
- Lancer le client
  - Accès à l’interface via le classpath ou le codebase

Invite de commandes - java Serveur

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleBase\serveur>java Serveur
 Serveur pret
 Service Message enregistre
```

Invite de commandes

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleBase\client>java Client
Le client recoit : Message : Salut !

C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleBase\client>
```

Les Exceptions

- La gestion des exceptions se fait de manière transparente
  - Les exceptions existantes sont levées et transmises identiquement à un traitement local
  - La création d’exceptions “spécifiques" se fait identiquement à une création locale

Les Exceptions

- La gestion des exceptions se fait de manière transparente
  - Les exceptions existantes sont levées et transmises identiquement à un traitement local
  - La création d'exceptions “spécifiques" se fait identiquement à une création locale

- Toute exception créée doit être accessible au serveur, au client et au rmiregistry (via le classpath ou le codebase).

Client.java | Serveur.java | ChaineVide.java | Hello.java | HelloImpl.java

```java
public class ChaineVide extends Exception {
   public ChaineVide(){
      super("String vide interdit !");
   }
}
```

Client.java | Serveur.java | ChaineVide.java | Hello.java | HelloImpl.java

```java

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.io.Serializable;

public interface Hello extends Remote {
   public String sayHello(String nom) throws RemoteException, ChaineVide;
   public void divisionParZero() throws RemoteException;
}
```

Client.java Serveur.java ChaineVide.java Hello.java HelloImpl.java

```
public class HelloImpl implements Hello {

  public String sayHello(String nom) throws ChaineVide {
    if(nom.length()==0)
      throw new ChaineVide();
    System.out.println("Request from " + nom);
    return "Bonjour " + nom + " !";
  }

  public void divisionParZero() {
    int tmp = 1/0;
  }
}
```

Client.java Serveur.java ChaineVide.java Hello.java HelloImpl.java

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Arrays;


public class Serveur {
  public static void main(String args[]) {
    int port  = 1099;
    if(args.length==1)
      port = Integer.parseInt(args[0]);
    try {
      Hello skeleton = (Hello)UnicastRemoteObject.exportObject(new HelloImpl(), 0);
      Registry registry = LocateRegistry.getRegistry(port);
      if(!Arrays.asList(registry.list()).contains("HelloExceptions"))
        registry.bind("HelloExceptions", skeleton);
      else
        registry.rebind("HelloExceptions", skeleton);
      System.out.println("Service HelloExceptions lie au registre");
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

Client.java | Serveur.java | ChaineVide.java | Hello.java | HelloImpl.java

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;


public class Client {
  public static void main(String args[]) {
    String machine = "localhost";
    int port = 1099;
      try {
      Registry registry = LocateRegistry.getRegistry(machine, port);
      Hello obj = (Hello)registry.lookup("HelloExceptions");
      if (args.length == 1)
        if (args[0].equals("division"))
            obj.divisionParZero();
        else    System.out.println(obj.sayHello(args[0]));
     else   System.out.println(obj.sayHello(""));

    } catch (Exception e) {
       System.out.println("Client exception: " +e);
    }
  }
}

```

Invite de commandes - java Serveur

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\serveur>java Serveur
Service HelloExceptions lie au registre
Request from pepe
```

Invite de commandes

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\client>java Client pepe
Bonjour pepe !

C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\client>
```

Invite de commandes - java Serveur

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\serveur>java Serveur
Service HelloExceptions lie au registre
Request from pepe
```

Invite de commandes

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\client>java Client division
Client exception: java.lang.ArithmeticException: / by zero

C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\client>
```

Invite de commandes - java Serveur

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\serveur>java Serveur
Service HelloExceptions lie au registre
Request from pepe
```

Invite de commandes

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\client>java Client
Client exception: ChaineVide: String vide interdit !

C:\Users\chzm\workspace\RMI\soucesNoPackages\exampleExceptions\client>
```

Passage des arguments

- En RMI, il est possible de passer les arguments des deux manières classiques (par valeur ou par référence)
- Tous les arguments de fonctions distantes doivent être
  - soit **des objets distants,**
  - soit **sérialisés**

Passage des arguments

- En RMI, il est possible de passer les arguments des deux manières classiques (par valeur ou par référence)
- Tous les arguments de fonctions distantes doivent être
  - soit **des objets distants,**
  - soit **sérialisés**

- La but de la sérialisation est de permettre de transférer un objet vers un support binaire.
  - Pour les types primitifs, ceci se fait de manière canonique (entiers, chaînes de caractères …)
  - Pour les types structurés, Java définit un mécanisme appelé sérialisation permettant de lier et d’écrire un objet de et vers une représentation en binaire de forme récursive.

Passage d’un objet `Serializable`

- Le passage d’un objet `Serializable` se fait de manière transparente
  - Identiquement à un passage de type de base
  - Il s’agit d’un passage par valeur pour un objet

- Remarques
  - Tout objet non `Serializable` ne pourra pas faire l’objet d’un passage de paramètre en RMI
  - À l’exécution, le rmiregistry (et évidemment le serveur et le client) doit avoir accès à la classe à sérialiser via le classpath ou le codebase

Guy.java | HelloImpl.java | Hello.java | Serveur.java | Client.java

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

public interface Hello extends Remote {
    public String sayHello(Guy aGuy) throws RemoteException;
}

```

Guy.java | HelloImpl.java | Hello.java | Serveur.java | Client.java

```java

import java.io.Serializable;

public class HelloImpl implements Hello {
    public String sayHello(Guy aGuy) {
        System.out.println("Request from a guy: " + aGuy.getName());
        return "Bonjour " + aGuy.getName() + " !";
    }
}

```

Guy.java | HelloImpl.java | Hello.java | Serveur.java | Client.java

```java

import java.io.Serializable;

public class Guy implements Serializable {
    private String name;
    public Guy(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
}

```

Guy.java HelloImpl.java Hello.java Serveur.java Client.java

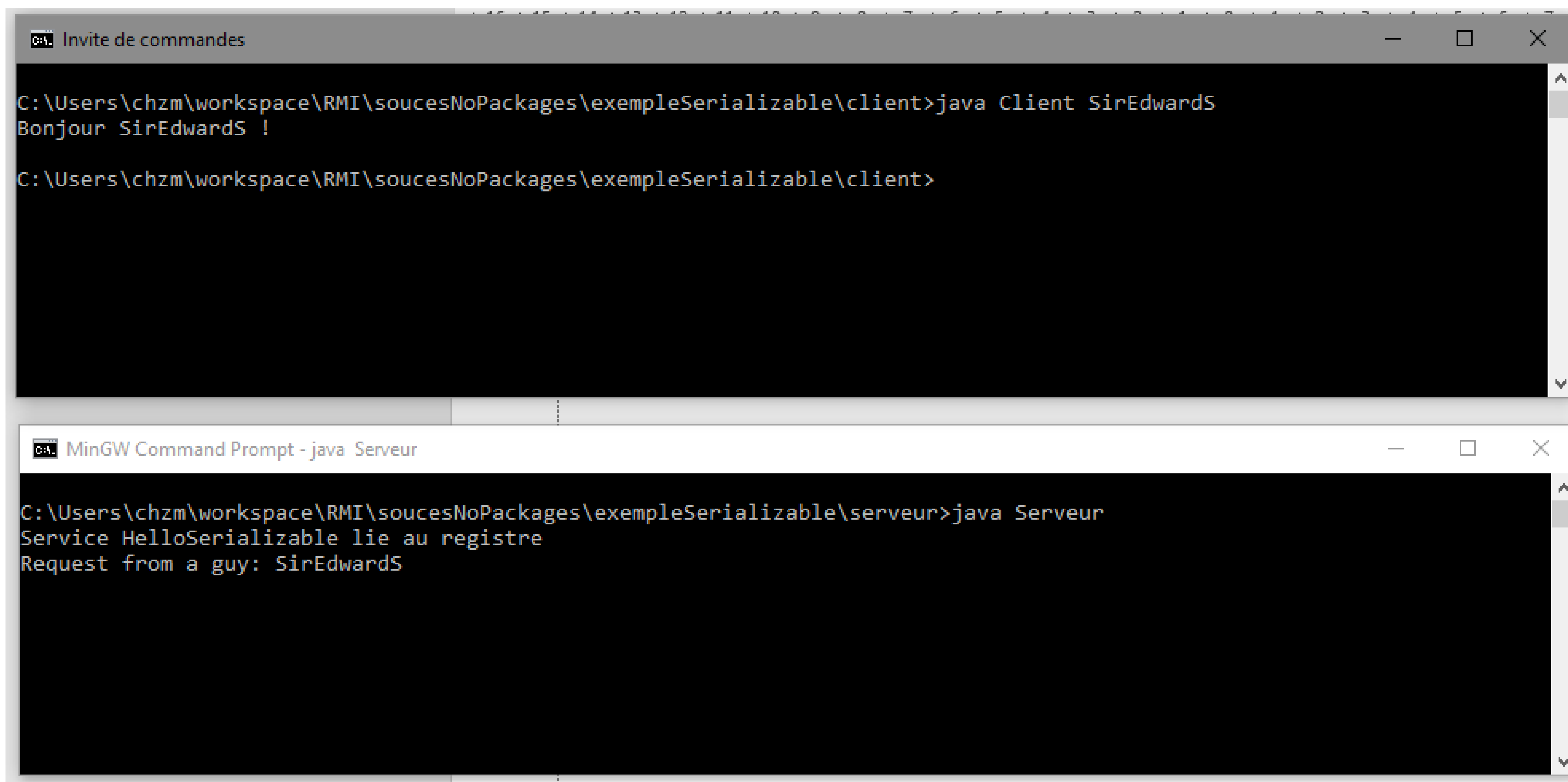
```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Arrays;

public class Serveur {
  public static void main(String args[]) {
    int port  = 1099;
    if(args.length==1)
      port = Integer.parseInt(args[0]);
    try {
      Hello skeleton = (Hello)UnicastRemoteObject.exportObject(new HelloImpl(), 0);
      Registry registry = LocateRegistry.getRegistry(port);
      if(!Arrays.asList(registry.list()).contains("HelloSerializable"))
        registry.bind("HelloSerializable", skeleton);
      else
        registry.rebind("HelloSerializable", skeleton);
      System.out.println("Service HelloSerializable lie au registre");
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

Guy.java | HelloImpl.java | Hello.java | Serveur.java | Client.java

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Client {
  public static void main(String args[]) {
    String machine = "localhost";
    Guy aGuy;
    int port = 1099;
    if (args.length ==0)
        aGuy = new Guy("personne");
    else aGuy = new Guy(args[args.length-1]);
    try {
      Registry registry = LocateRegistry.getRegistry(machine, port);
      Hello obj = (Hello)registry.lookup("HelloSerializable");
      System.out.println(obj.sayHello(aGuy));
    } catch (Exception e) {
       System.out.println("Client exception: " +e);
    }
  }
}
```


Invite de commandes
C:\Users\chzm\workspace\RMI\soucesNoPackages\exempleSerializable\client>java Client SirEdwardS
Bonjour SirEdwardS !
C:\Users\chzm\workspace\RMI\soucesNoPackages\exempleSerializable\client>
MinGW Command Prompt - java Serveur
C:\Users\chzm\workspace\RMI\soucesNoPackages\exempleSerializable\serveur>java Serveur
Service HelloSerializable lie au registre
Request from a guy: SirEdwardS

Invite de commandes

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exempleSerializable\client>java Client
Bonjour personne !

C:\Users\chzm\workspace\RMI\soucesNoPackages\exempleSerializable\client>
```

MinGW Command Prompt - java Serveur

```
C:\Users\chzm\workspace\RMI\soucesNoPackages\exempleSerializable\serveur>java Serveur
Service HelloSerializable lie au registre
Request from a guy: SirEdwardS
Request from a guy: personne
```