المدرسة الدولية الخاصة للتقنيات
POLYTECH INTL

Informatique Répartie
- RMI Remote Method Invocation -

Ichrak MEHREZ
(m_ichrak@hotmail.fr)

2018-2019

Principe des RMI

- RPC à la Java
  - invoquer de façon simple des méthodes sur des objets distribués.
- Outils
  - pour la génération des stub/skeleton, l 'enregistrement par le nom, l'activation
    - Tous les détails (connexion, transfert de données ..) sont transparents pour le développeur grâce au stub/skeleton généré
- Mono-langage et Multiplateforme.
  - Java : de JVM à JVM (les données et objets ont la même représentation qqs la JVM)

Principe des RMI

- Orienté Objet
  - Les RMIs utilisent le mécanisme standard de sérialisation de JAVA pour l'envoi d 'objets.
- Dynamique
  - Les classes des Stubs et des paramètres peuvent être chargées dynamiquement via HTTP (http://) ou NFS (file:/)
- Sécurité
  - un SecurityManager vérifie si certaines opérations sont autorisés par le serveur

Rappel : Appel local (interface et objet)

```
public interface ReverseInterface {
String reverseString(String chaine);
}
public class Reverse implements ReverseInterface
{
public String reverseString (String ChaineOrigine){
int longueur=ChaineOrigine.length();
StringBuffer temp=new StringBuffer(longueur);
    for (int i=longueur; i>0; i--) {
            temp.append(ChaineOrigine.substring(i-1, i));}
return temp.toString();
}
}
```

Rappel : Appel local (programme appelant)

```
import ReverseInterface;
public class ReverseClient
{
public static void main (String [] args)
{ Reverse rev = new Reverse();
String result = rev.reverseString (args [0]);
System.out.println ("L'inverse de "+args[0]+" est "+result);
}
}
$javac *.java
$java ReverseClient Alice
L'inverse de Alice est ecilA
$
```

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);**

Qu'attend t-on d'un objet distribué ?

- On doit pouvoir récupérer le résultat d'un appel de méthode sous la forme d'un objet distribué.

  - **monObjetDistribue=autreObjetDistribue.uneMethodeDeLOD();**
  - **monObjetDistribue=monObjetLocal.uneMethodeDeLOL();**

Java RMI : Remote Method Invocation

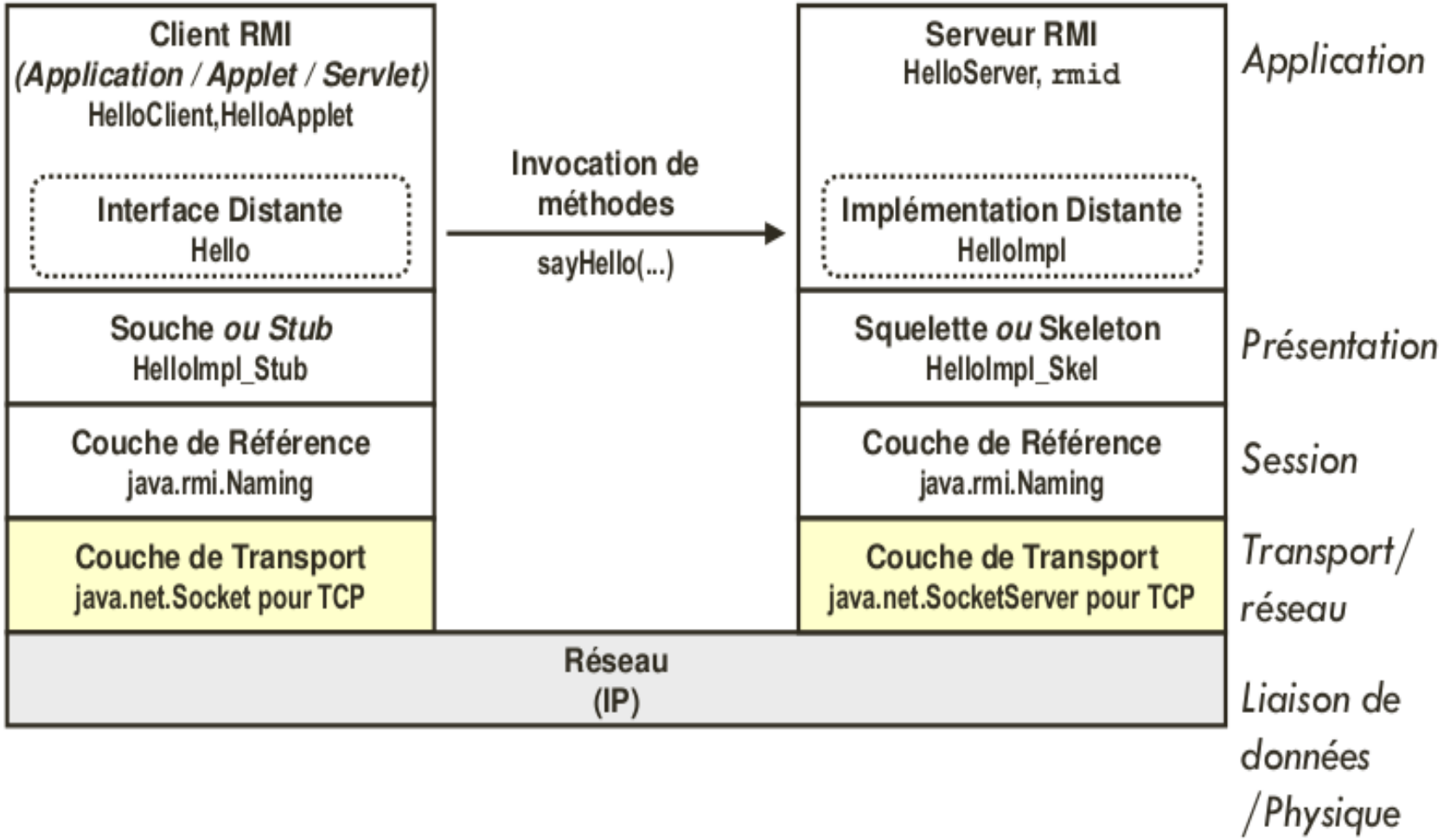
- Permet la communication entre machines virtuelles Java (JVM) qui peuvent se trouver physiquement sur la même machine ou sur deux machines distinctes.

- Présentation
  - 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 : caractéristiques

- 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.

Structure des couches RMI
Architecture logique



Structure des couches RMI

- **Souche ou Stub (sur le client)**
  - Représentant local de l'objet distant qui implémente les méthodes « exportées » de l'objet distant
  - « marshalise » les arguments de la méthode distante et les envoie en un flot de données au serveur
  - « démarshalise » la valeur ou l 'objet retournés par la méthode distante
- **Squelette ou Skeleton (sur le serveur)**
  - "démarshalise" les paramètres des méthodes
  - fait un appel à la méthode de l'objet local au serveur
  - "marshalise" la valeur ou l 'objet renvoyé par la méthode

- Le compilateur RMI (rmic) se charge de générer automatiquement les classes *stub* et *skel* à partir de la définition des classes serveur.

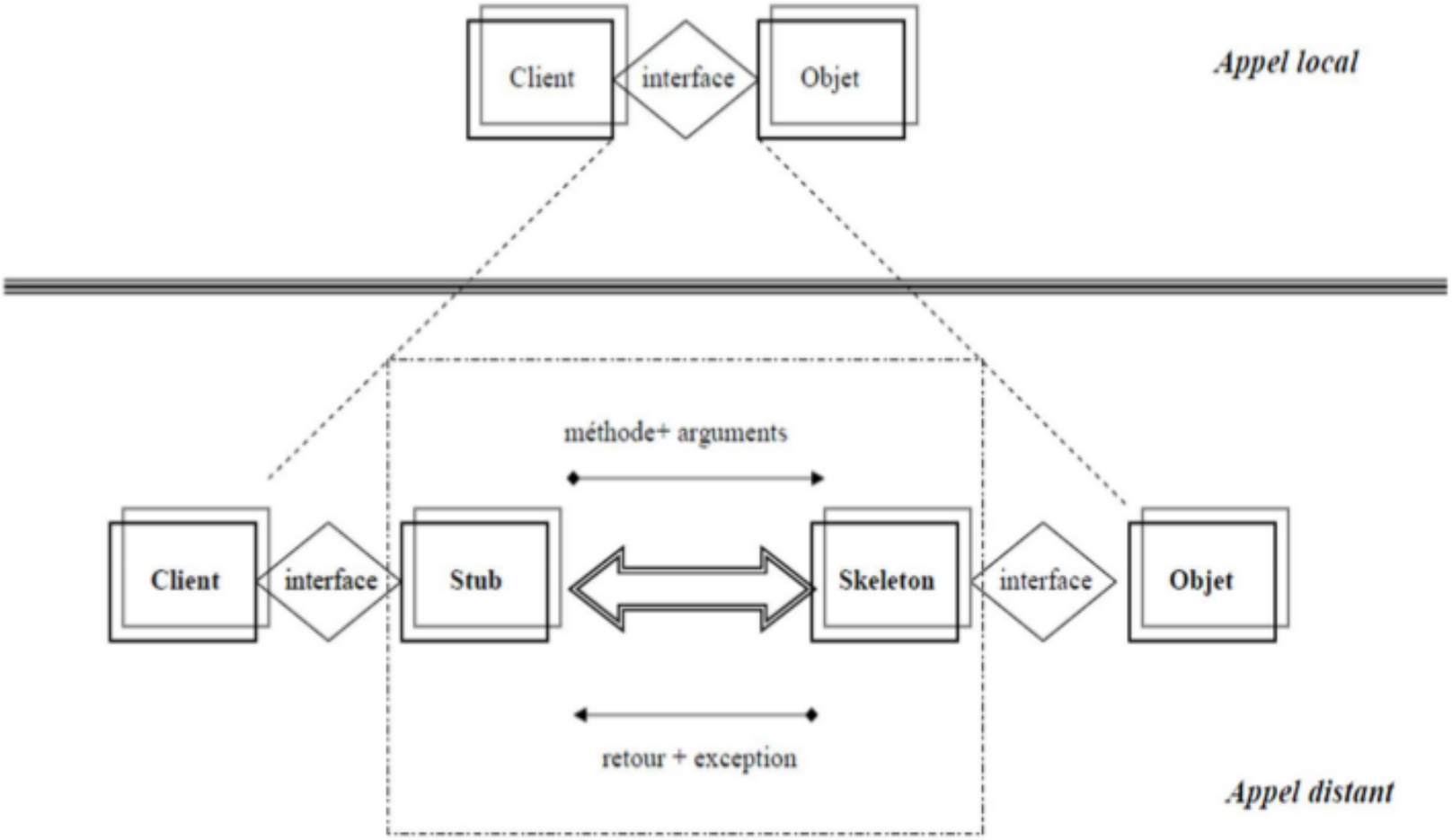
Structure des couches RMI

- Couche des références distantes (Remote Refernce 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 **rmiregistery** (qui possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants)
  - Un unique **rmiregistery** par JVM
  - **rmiregistery** s'exécute sur chaque machine hébergeant des objets distants
  - **rmiregistery** accepte des demandes de service sur le port **1099**

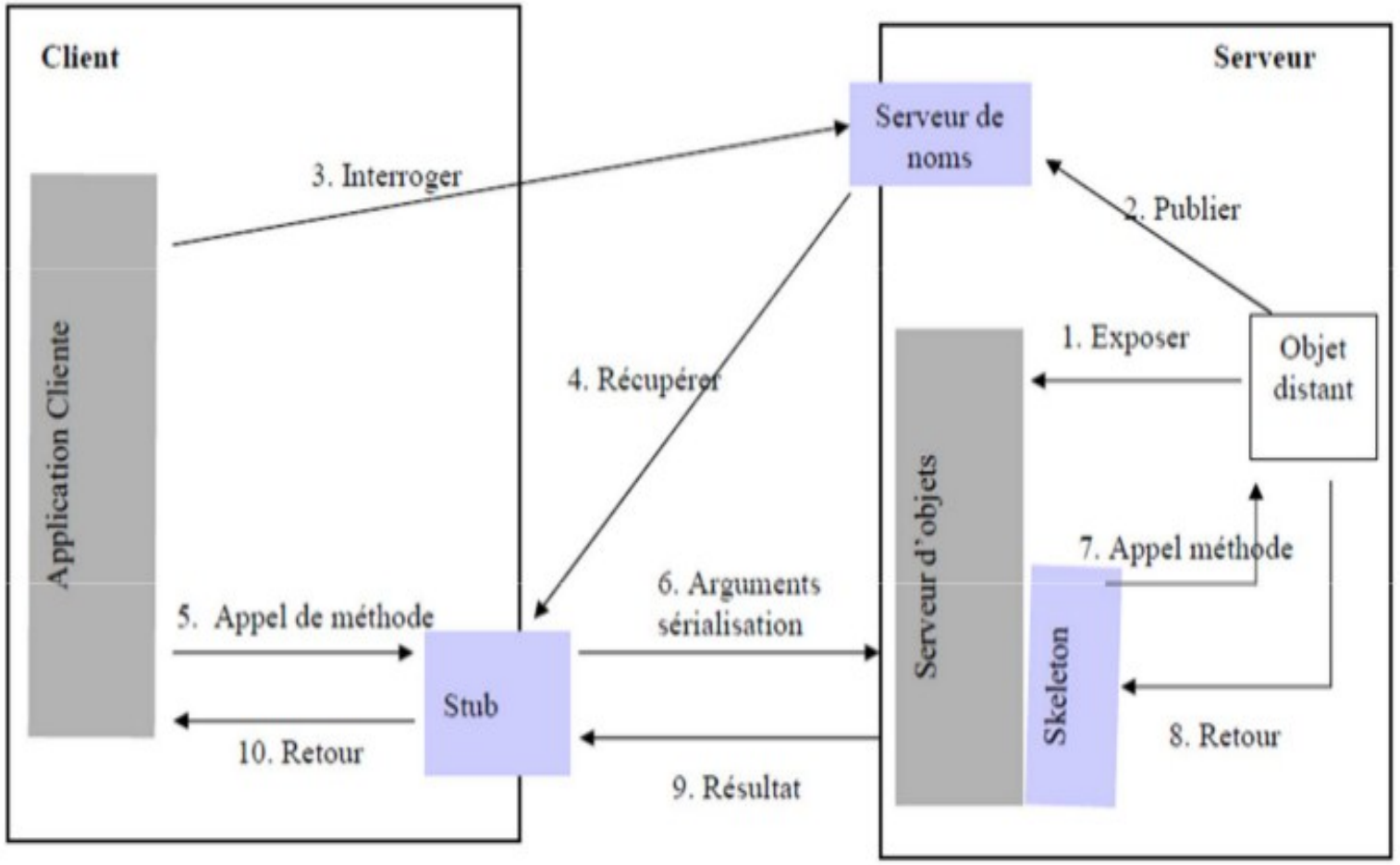
Structure des couches RMI

- **Couche de transport**
  - écoute les appels entrants
  - établit et gère les connexions avec les sites distants
  - **java.rmi.UnicastRemoteObject** utilise les classes Socket et SocketServer (TCP)
  - d'autres classes peuvent être utilisées par la couche transport

Appel Local vs Appel à distance



Etapes d'un appel de méthodes distante



IMPLÉMENTATION

Cycle de développement d'une application avec RMI

1. Définir une interface distante (Xyy.java) qui spécifie le comportement de l'objet distant
2. Créer une classe implémentant cette interface (XyyImpl.java)
3. Compiler cette classe (***javac*** XyyImpl.java)
4. Créer une application serveur (XyyServer.java)
5. Compiler l'application serveur
6. Créer à l'aide de rmic, les classes *stub* et *skeleton* (XyyImpl_Stub.java et XyyImpl_Skel.java)
7. Démarrage du registre avec **rmiregistry**
8. Lancer le serveur pour la création d'objets et leur enregistrement dans **rmiregistry**
9. Créer une classe cliente qui appelle des méthodes distantes de l'objet distribué (XyyClient.java)
10. Compiler cette classe et la lancer.

Exemple

- Inversion d'une chaîne de caractères à l'aide d'un objet distribué
  - Invocation distante de la méthode ***reverseString()*** d'un objet distribué qui inverse une chaîne de caractères fournie par l'appelant.
- On définit :
  - **ReverseInterface.java** : interface qui décrit l'objet distribué
  - **Reverse.java** : qui implémente l'objet distribué
  - **ReverseServer.java** : le serveur RMI
  - **ReverseClient.java** : le client qui utilise l'objet distribué

Exemple: Fichiers nécessaires

Coté Client

- *l'interface : ReverseInterface*
- *le client : ReverseClient*

Coté Serveur

- *l'interface : ReverseInterface*
- *l'objet : Reverse*
- *le serveur d'objets : ReverseServer*

Exemple

- Interface de l'objet distribué
  - 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.

Exemple

- Interface de la classe distante

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface ReverseInterface extends Remote {
String reverseString(String chaine) throws
    RemoteException;
}
```

Exemple

- Implémentation de l'objet distribué

  L'implémentation doit étendre la classe *RemoteServer* de java.rmi.server

- RemoteServer est une classe abstraite

- UnicastRemoteObject étend RemoteServer
  - c'est une classe concrète qui permet l'invocation de méthodes distantes
  - une instance de cette classe réside sur un serveur et est disponible via le protocole TCP/IP

Exemple

```
//Implémentation de l'objet distribué
    import java.rmi.*;
    import java.rmi.server.*;
    public class Reverse extends UnicastRemoteObject  implements ReverseInterface
    {
    public Reverse() throws RemoteException {
    super();
    }
    public String reverseString (String ChaineOrigine) throws
    RemoteException {
    int longueur=ChaineOrigine.length();
    StringBuffer temp=new StringBuffer(longueur);
      for (int i=longueur; i>0; i--)
      {
       temp.append(ChaineOrigine.substring(i-1, i));}
    return temp.toString();
    }}
```

Exemple

Le serveur

- Programme à l'écoute des clients
- Publication du service offert par le serveur dans le service annuaire (Enregistre l'objet distribué dans **rmiregistry**)
- Cet annuaire sera utiliser par les clients pour chercher les services du serveurs
- Publication à travers la méthode **rebind** de la classe Naming :
  - Naming.rebind(String nom, Remote obj)
  - nom : nom du service
  - obj : objet serveur
  - Naming.rebind("rmi://hote.isi.tn:1099/MyReverse", rev);

Exemple

```
//Le serveur
import java.rmi.*;
import java.rmi.server.*;
public class ReverseServer {
public static void main(String[] args)
{
try {
System.out.println( "Serveur : Construction de l'implémentation ");
Reverse rev= new Reverse();
System.out.println("Objet Reverse lié dans le RMIregistry");
Naming.rebind("rmi://sinus.isi.tn:1099/MyReverse", rev);
System.out.println("Attente des invocations des clients ..."); }
catch (Exception e) { System.out.println("Erreur de liaison de l'objet Reverse");
System.out.println(e.toString()); }
} // fin du main
} // fin de la classe
```

Exemple

Le Client

- On crée un client à qui on passe en paramètre l'URL du serveur : le client obtient un stub pour accéder à l'objet par une URL RMI (localisation du service)

  ReverseInterface ri = (ReverseInterface) **Naming.lookup** ("rmi://sinus.isi.tn:1099/MyReverse");

- Une URL RMI commence par **rmi://**, le nom de machine, un numéro de port optionnel et le nom de l'objet distant.
  - rmi://hote:2110/nomObjet
  - Par défaut, le numéro de port est 1099

Exemple

Le Client

- Obtient une référence d'objet distribué :

```
ReverseInterface ri = (ReverseInterface) Naming.lookup
    ("rmi://sinus.isi.tn:1099/MyReverse");
```

- Exécute une méthode de l'objet :

```
String result = ri.reverseString ("Terre");
```

Exemple

```
//Le Client
import java.rmi.*;
import ReverseInterface;
public class ReverseClient
{
public static void main (String [] args)
{
try{
ReverseInterface rev = (ReverseInterface) Naming.lookup("rmi://sinus.isi.tn:1099/MyReverse");
String result = rev.reverseString (args [0]);
System.out.println ("L'inverse de "+args[0]+" est "+result); }
catch (Exception e)
{System.out.println ("Erreur d'accès à l'objet distant.");
System.out.println (e.toString());}
}}
```