TP2

RMI (Remote Method Invocation)

1) Démarche pour développement RMI :

- Créer les interfaces des objets distants
- Créer les implémentations des objets distants
- 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

a) Créer les interfaces des objets distants

Cette première étape consiste à créer une interface distante qui décrit les méthodes que le client pourra invoquer à distance. Pour que ses méthodes soient accessibles par le client, cette interface doit hériter de l'interface **Remote**. Toutes les méthodes utilisables à distance doivent pouvoir lever les exceptions de type **RemoteException** qui sont spécifiques à l'appel distant.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface ADDi extends java.rmi.Remote
{
public int add(int x, int y) throws java.rmi.RemoteException;
 }
```

b) Implémentation des objets distants

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. Le plus facile à utiliser étant java rmi server UnicastRemoteObject. C'est dans cette classe que nous allons définir le corps des méthodes distantes que pourront utiliser nos clients.

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class ADDImpl extends UnicastRemoteObject implements ADDi
 {
private String name;
public ADDImpl () throws RemoteException
 {
 super();
 }
public int add(int x, int y) throws RemoteException  //  Implémentation de la méthode distante
{
return x+y;
 } }
```

c) Générer les stubs et skeletons

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

Tapez la commande : **rmic ADDImpl**

d) Créer le serveur 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. Les clients trouvent les services distants en utilisant un service d annuaire 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). 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. Cela se fait grâce à la méthode statique **bind()** ou **rebind()** de la classe **Naming**.

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

e) Créer le client RMI

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.

```
import java.rmi.Naming;
public class ClientRMI {
public static void main(String[] args)
{
try{
ADDi  obj =(ADDi)Naming.lookup("rmi://localhost:1099/bonjour");
int n = obj.add(5,4) ;
System.out.println(''addition is : ''  +n );
}
catch (Exception e)
{ e.printStackTrace(); }
 }}
```

f) Lancement

Lancer la commande: **start rmiregistry.**

2) Exercice d'application en Java RMI

Nous disposons d'un service qui offre des opérations de gestion de son compte courant. Voici le code des méthodes offertes par ce service :

```
void debiter(double montant) {
}
void crediter(double montant) {
}
double lire_solde() {
}
```

1. On souhaite rendre chacune de ces méthodes accessibles à distance de manière à ce qu'elles définissent l'interface entre le client et le serveur. Ecrire cette interface.

2. Déduire la classe qui matérialise le service qui offre les opérations debiter(), crediter() et lire_solde().

3. Implémenter la classe Serveur.java pour permettre l'enregistrement du service auprès de RMI Registry.

4. Générer tous les fichiers nécessaires au lancement du Serveur.

5. Lancer la suite de commandes ayant pour finalité le lancement du serveur.

6. Implémenter la classe Client.java qui doit être lancé à partir d'un autre répertoire ou d'une autre machine.