INFO721 TP3 - Programmation répartie avec JAVA RMI

L'objectif de ce TP est l'initiation à la programmation répartie. Il consiste à programmer une application typique comportant de la répartition à l'aide de RMI. Vous devez programmer les parties "client" et "serveur" de cette application à l'aide du langage JAVA et la faire exécuter dans un environnement de plusieurs machines. Ces machines pouvant être sur une seule machine physique (plusieurs JVM locales) ou sur plusieurs machines distantes.

ETAPE 1 : Familiarisation avec Java RMI

Cette première étape doit vous permettre de comprendre et d'exécuter une application "simple" mettant en œuvre une partie cliente et une partie serveur communiquant ensemble à l'aide de RMI. Dans cette première partie, vous allez déployer l'exemple très simple vu en cours : *HelloWorld*

Interface Hello

```
package example.hello;

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

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

Serveur

```
package example.hello;

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

public class Server implements Hello {

    public Server() {}

    public String sayHello() {
        return "Hello, world!";
    }

    public static void main(String args[]) {

        try {
            Server obj = new Server();
            // Declare le stub surlequel sera expose l'objet distribue
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

            // Permet de lier au Registre l'objet distribue
            Registry registry = LocateRegistry.createRegistry(1099);
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
```

```
            e.printStackTrace();
        }
    }
}
```

Client

```
package example.hello;

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

public class Client {

    private Client() {}

    public static void main(String[] args) {

        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

ETAPE 2 : Réalisation d'un FTP simplifié

Le serveur FTP simple

Vous devez programmer un serveur FTP en JAVA à l'aide de RMI. Au lancement de votre serveur, l'utilisateur doit pouvoir spécifier :

- Le port sur lequel écoute le serveur de nom (rmiregistry).
- La racine du système de fichier que gère le serveur.

Le serveur doit fournir deux méthodes :

- **byte[] get(String filename);**
  Permet de récupérer, dans le répertoire courant du serveur, un fichier identifié par son nom. Le fichier sera stocké dans le répertoire courant du client. Cette méthode prend en paramètre le nom du fichier à télécharger. Cette méthode retourne au client un tableau d'octet qui contient le contenu du fichier.

- **void put(String filename, byte[] file);**
  Permet d'écrire dans le répertoire courant du serveur un fichier identifié par son nom du côté du client. Le fichier sera stocké dans le répertoire courant du serveur.

Cette méthode prend en paramètre le nom du fichier à télécharger.

Le Client FTP simple

Ce client prend en paramètre de lancement :

- Le nom de la machine du serveur de nom
- Le port du serveur de nom
- Le fichier à télécharger

ETAPE 3 (si vous avez du temps) : Amélioration du FTP

Le serveur FTP

Modifiez votre serveur FTP pour qu'il réponde aux requêtes suivantes :

- **void cd(String dir);**
  Permet de changer le répertoire courant sur le serveur. Cette requête contient le chemin d'accès. C'est l'équivalent de la commande cd <chemin>.
- **string[] ls();**
  Permet de lister le contenu du répertoire courant sur le serveur. C'est l'équivalent de la commande ls. Le serveur retourne au client la liste du répertoire courant.
- **String pwd();**
  Permet d'afficher le répertoire courant. C'est l'équivalent de la commande pwd.

Le Client FTP

Votre client FTP évolue pour permettre de lancer des commandes FTP sous forme texte (comme dans un shell) :

- **connect <nom_serveur_ftp>**
  Le client se connecte au serveur et affiche le résultat (connexion effective ou non).
- **ls**
  Appelle la commande ls à distance sur le serveur et affiche le résultat.
- **cd <chemin>**
  Effectue le cd à distance sur le serveur et affiche le résultat.
- **get <nom_fichier>**
  Récupère le fichier à distance sur le serveur et le stocke localement.
- **put <nom_fichier>**
  Envoi et stocke le fichier, présent localement, sur le serveur