Université d'Évry
Val d'Essonne

Master Informatique / MIAGE

Systèmes et Applications Répartis (SAR)

TD 3 : RPC (Remote Procedure Call)

Exercice 1 (RPC avec Java)

Le but de cet exercice est de mettre en place des objets distants en utilisant les sockets. Nous allons programmer une version simplifiée des coulisses de RMI (Remote Methode Invocation). Nous allons implémenter les Stubs et les skeletons.

Question 1. Rappelez brièvement les rôles des Stubs et des Skeletons dans le RPC (Remote Procedure Call).

Soit la classe *ObjetDistantImpl* (figure 1) qui implémente les deux méthodes $M1$ et $M2$ de l'interface *ObjetDistant*.
Les deux méthodes prennent respectivement en paramètre un objet de type *ObjetParam*1 et *ObjetParam*2 et renvoient respectivement *ObjetParam*2 et *ObjetParam*1. La figure 1 contient la définition des quatre types.

Listing 1 – classe ObjetDistantImpl et son interface ObjetDistant.

```
1  public class ObjetParam1{}
2
3  public class ObjetParam2{}
4
5  public interface ObjetDistant{
6
7      public ObjetParam2 M1(ObjetParam1 arg);
8
9      public ObjetParam1 M2(ObjetParam2 arg);
10 }
11
12 public class ObjetDistantImpl implements ObjetDistant{
13
14     public ObjetParam2 M1(ObjetParam1 arg){
15             System.out.println("La méthode M1 vient d'être invoquée.");
16             return new ObjetParam2();
17     }
18
19     public ObjetParam1 M2(ObjetParam2 arg){
20             System.out.println("La méthode M2 vient d'être invoquée.");
```

```
21              return new ObjetParam1();
22      }
23  }
```

Question 2. Quelle est la différence entre le passage de paramètres par valeur et le passage par référence ? Dans le cadre des appels/invocations à distance, seules les valeurs peuvent transiter sur le réseau, comment simule-t-on le passage par référence ?

Question 3. Dans le cas de la classe *ObjetDistantImpl*, on suppose que les passages des paramètres pour les deux méthodes de notre objet distant se fait par valeur. Quelles modifications faut-il apporter aux classes *ObjetParam*1 et *ObjetParam*2 pour que cela soit possible ?

1 Développement des Stub et des Skeleton

On désigne par *StubObjetDistant* la classe Stub de notre objet distant. Le Stub est une classe qui a la même signature que l'objet distant (implémente la même interface, voir figure 2) et qui est située côté client. Au lieu du code des méthodes, les méthodes du Stub contiennent le code nécessaire pour transformer les appels locaux à des requêtes vers le Skeleton responsable de l'objet : à chaque appel/invocation de l'une de ses méthodes, le Stub ouvre une connexion avec le Skeleton et envoi une sérialisation de l'appel. Chaque appel est sérialisé comme suit :

- L'envoi du nom de la méthode qu'on veut invoquer ; string
- L'envoi de l'ensemble des paramètres (dans l'ordre de la signature) ;
- L'attente de la réponse. param1 ou param2

La réponse ainsi obtenue sera le retour de la méthode invoquée. Côté serveur, chaque objet distant (ici instance de ObjetDistantImpl) est confié à un objet "Skeleton", Skeleton ObjetDistant, sur une adresse particulière et un port particulier. Le "skeleton" est un processus (programme) qui a pour charge de répondre aux demandes de connexion des clients. Pour chaque demande de connexion, la socket résultante est confiée à la méthode *traiterRequete* (voir figure 2) qui écoute de manière symétrique ce que le "stub" envoi, ensuite elle invoque la méthode correspondante de l'instance de l'objet géré par le "skeleton" et renvoie la réponse à travers la socket au client (en vérité au stub).

Listing 2 – code à compléter.

```
1
2  public class StubObjetDistant implements ObjetDistant{
3      private int port;
4      private String adresse;
5
6      public StubObjetDistant(String ad, int p){ port=p; adresse=ad; }
7
8      public ObjetParam2 M1(ObjetParam1 arg){ /* à compléter */ }
9
10     public ObjetParam1 M2(ObjetParam2 arg){ /* à compléter */ }
11 }
12
```

```
13 public class SkeletonObjetDistant{
14     private int port;
15     private ObjetDistant od;
16
17     public SkeletonObjetDistant(ObjetDistant o, int p){ od=o; port=p; }
18
19     public void run(){ /* à compléter */ }
20
21     private void traiterRequete(Socket s){ /*à compléter */ }
22 }
```

Question 4. Complétez le code des deux classes *SkeletonObjetDistant* et *StubObjetDistant* de la figure 2 afin de simuler l'invocation de méthodes à distance des méthodes $M1$ et $M2$.

Rappel Pour envoyer sur une socket s un objet "Serializable" obj, on connecte la sortie de la socket (s.getOutputStream()) avec celle d'un objet de type ObjectOutputStream o comme suit :

$ObjectOutputStreamo = newObjectOutputStream(ss.getOutputStream());$
$oo.writeObject(obj);$

pour lire :

$ObjectInputStreamoo = newObjectInputStream(ss.getInputStream());$
$obj = o.readObject();$