06/01/2020

2ème Année GI

UNIVERSITE INTERNATIONALE DE CASABLANCA

M.K. GUENNOUN

Examen Java et systèmes distribués

Durée : 2h00

PARTIE SOCKETS (8pts)

1. Ecrire un programme Client (machine locale et port serveur passé en argument) et un programme Serveur (machine locale et port passé en argument) avec les comportements suivants (4pts) :
   - Le client envoie 2 messages
     - Avant les envois, le client demande un nom à l'utilisateur (première lecture)
     - Il demande ensuite un premier message à l'utilisateur (deuxième lecture)
     - Il envoie une chaîne de caractères au serveur comprenant le nom saisi concaténé avec le message
     - Il attend la réception de l'acquittement du serveur et affiche le message reçu pour cet acquittement.
     - Il demande un deuxième message (troisième lecture), puis renvoie le nom donné initialement concaténé avec le nouveau message
   - Le serveur
     - Demande à l'utilisateur un nom pour le serveur puis réceptionne les 2 messages
     - Après chaque réception, le serveur envoie un acquittement au client en intégrant le nom du serveur saisi par l'utilisateur
     - Puis se met en attente du message suivant
2. Reprendre l'exercice précédent en considérant cette fois un nombre de messages non limité. L'arrêt de la conversation se fera quand le client saisira le message fin. (4pts)

PARTIE RMI (12pts)

Nous nous plaçons maintenant dans le cadre de la technologie RMI. Nous considérons un client C1, deux servants **Servant1** et **Servant2**, et un serveur S1.

Le servant **Servant1** offre une opération **soustraction** prenant deux entiers x et y en argument et renvoyant l'entier x-y. Le servant **Servant2** offre une opération **inferieur** prenant deux entiers x et y en argument en renvoyant un booléen (vrai si x < y et faux sinon)

1. Ecrire les différents codes correspondant: (4pts)
   a. Aux servants **Servant1 et Servant2**
   b. Au serveur S1. Ce serveur est censé créer et publier un objet de chaque servant
   c. Au Client C1. Le client C1 demande à l'utilisateur deux entiers m et n, **fait une invocation sur les deux servants** pour afficher le résultat du calcul de m-n et dire si m est inférieur à n.
2. Donner les étapes de compilation et d'exécution pour lancer l'application (2pt)

Nous considérons maintenant un troisième servant **Servant3** offrant une méthode de calcul prenant en argument deux entiers m et n et renvoyant m%n. L'algorithme implanté à ce niveau s'appuie sur le calcul récursif très simple ci-dessous.

- m%n = m si m<n, et ((m-n)%n) sinon.

Ce servant est déployé au sein d'un serveur S2 qui en crée un objet et le publie.

3. Ecrire le code du servant **Servant3** sachant qu'il **réalise ses traitements en faisant appel aux deux servants Servant1 et Servant2**. (2pts)
4. Ecrire le code de S2. (1pt)
5. Ecrire le code du client C2 demandant à l'utilisateur deux entiers m et n puis, en invoquant les deux serveurs, affichant le résultat des calculs de m<n, m-n, et m%n. (2pts)
6. Donner l'architecture de l'application (1pt)

Annexe 1

- **Fichier Client.java**

```
public class Client
{public static void main(String args[]) throws Exception{
InetAddress IA=InetAddress.getLocalHost();
Socket S=new Socket(IA,Integer.parseInt(args[0]));
BufferedReader BR = new BufferedReader(new InputStreamReader(S.getInputStream()));
PrintWriter pred = new PrintWriter( S.getOutputStream());
...
S.close();}}
```

- **Fichier Serveur.java**

```
public class Serveur
{public static void main(String args[]) throws Exception{
ServerSocket SS= new ServerSocket(Integer.parseInt(args[0]));
Socket S=SS.accept();
BufferedReader BR = new BufferedReader(new InputStreamReader(S.getInputStream()));
PrintWriter PW = new PrintWriter(S.getOutputStream());
...
S.close();}}
```

Annexe 2

- **Fichier HelloInt.java**

```
public interface HelloInt extends Remote
{public void SayIt() throws RemoteException;}
```

- **Fichier Hello.java**

```
public class Hello extends UnicastRemoteObject implements HelloInt {
                public Hello() throws RemoteException {
                super();
                }
public void SayIt()throws RemoteException {
System.out.println("bonjour");}}
```

- **Fichier Client.java**

```
public class Client{
public static void main(String[] args)
{try {
        Remote remote_h = Naming.lookup("rmi://localhost:1099/hello");
        if (remote_h instanceof HelloInt)
                {
                        ((HelloInt) remote_h).SayIt();
                }}
catch (Exception e) {}}}
```

- **Fichier Serveur.java**

```
public class Serveur{
public static void main(String[] args) throws Exception {
Hello h = new Hello();
Naming.bind("rmi://localhost:1099/hello",h);
System.out.println("C'est fait!");
}}
```