Sockets en Java: exercices supplémentaires

1 Travail préliminaire

Reprendre les programmes du TP précedent (**`ClientTCP3.java`** et **`ServeurTCP3.java`**), les renommer en **`ClientTCP4.java`** et **`ServeurTCP4.java`**. Les modifier pour les faire fonctionner sur une machine distante, en partenariat avec un binôme.

Chacun d'entre vous fera tourner un serveur, et chacun d'entre vous exécutera un client qui se connectera au serveur sur la machine du voisin. Il faudra évidemment utiliser deux numéros de ports différents et remplacer dans le client la connection à "localhost" par une connection basée sur l'IP de la machine du voisin.

Chercher dans la doc comment on peut spécifier une adresse IP :
`https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html`

2 Application complète - 1

Ecrire deux programmes **`ClientTCP5.java`** et **`ServeurTCP5.java`** qui vont communiquer en TCP.

- Le serveur va recevoir des chaînes de caractères et renvoyer la chaîne "ok" au client à chaque fois, sauf s'il reçoit "end" auquel cas il doit renvoyer "fin". Il restera ensuite en attente pour une prochaine connexion.
- Le client doit demander à la console des chaînes au clavier, puis les envoyer au serveur, et vérifier que le serveur répond correctement. Dans le cas contraire, il doit afficher un message d'erreur.

3 Application complète - 2

Ecrire deux programmes **`ClientTCP6.java`** et **`ServeurTCP6.java`** qui vont communiquer en TCP.

De façon similaire au programme précédent, le client va envoyer des chaînes de caractères au serveur, et ce dernier accusera réception par la chaîne "ok" renvoyée au client.

Le serveur devra cependant concaténer la chaîne reçue avec les autres, en les séparant par une espace. Il devra ensuite renvoyer au client toute cette chaîne lors de la réception de la chaîne "end", et le client devra l'afficher. Puis le chaîne sera réinitialisée.

Il faudra cependant ajouter une limitation : si jamais la longueur de la chaîne concaténée dépasse 50 caractère, alors le serveur devra envoyer au client le message "erreur" (au lieu de "ok").

Annexe : compléments Java

1 - Négation de comparaison Pour tester l'opposé d'une comparaison, il faut utiliser l'opérateur unaire "**`!`**". Par exemple, si **`maFonction()`** est une fonction qui renvoie une valeur booléenne, alors ce test :

```
if( !maFonction() )
```

sera vrai si la fonction renvoie **`false`**, et inversement.

2 - Comparaison de chaînes Pour comparer des chaînes de caractères, il faut utiliser la methode **`equals()`** :

```
String s1,s2;
...
if( s1.equals(s2) )
{
   ...
}
```

3 - Saisie clavier Cette fonctionnalité est fournie par le package **`java.util.Scanner`**. Il faut d'abord instancier un objet (une seule fois) :

```
Scanner scan= new Scanner(System.in);
```

On pourra de façon itérative lire des chaines au clavier avec :

```
String str = scan.nextLine();
```