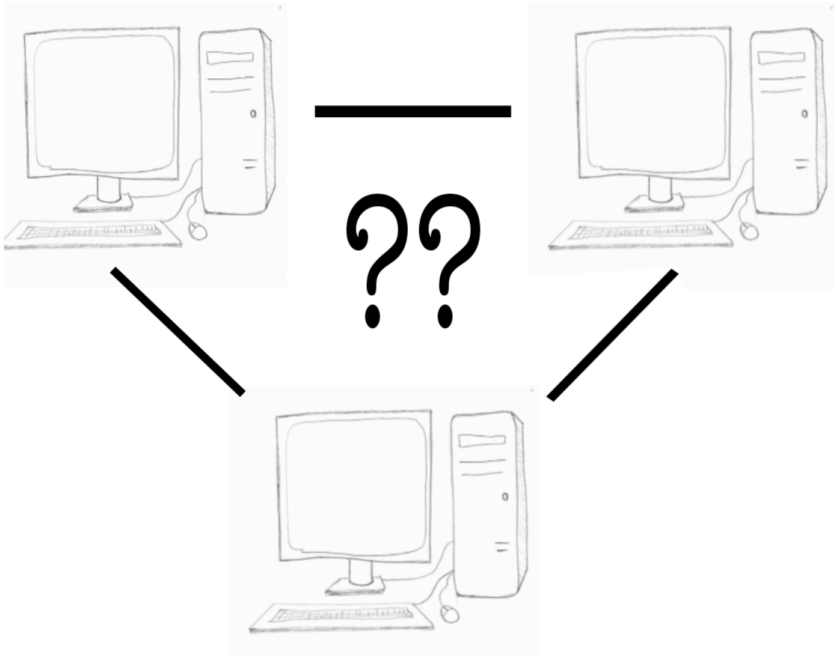
PROGRAMMATION RÉSEAU

Arnaud Sangnier
sangnier@irif.fr

API TCP Java - III



La suite ?

- C'est ce que nous allons voir maintenant avec la classe **ServerSocket**
- Ce qui est bien
  - à part l'attente d'une connexion sur un port
  - ensuite on aura des sockets que l'on manipulera comme pour les clients

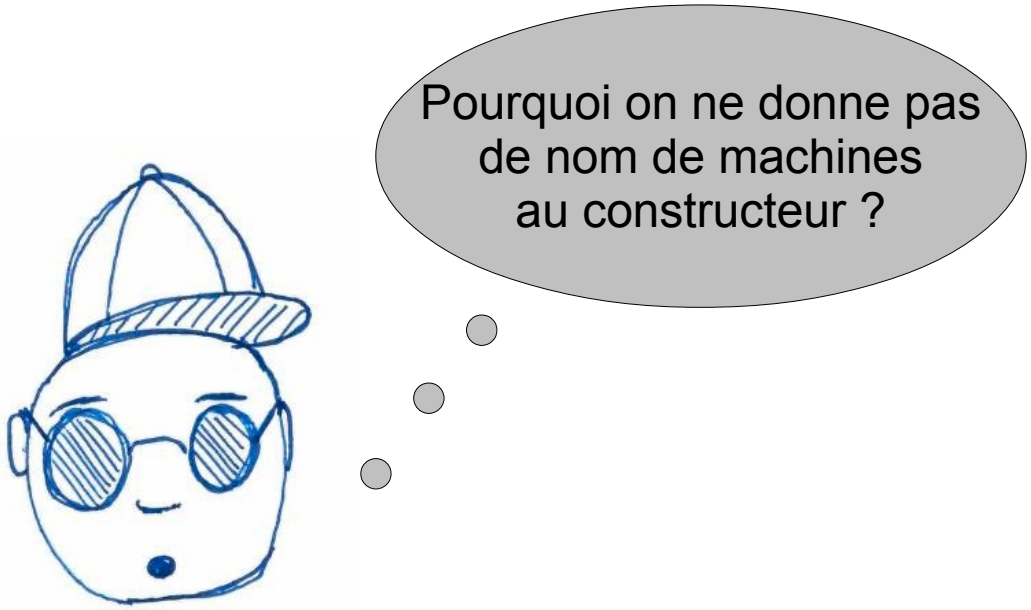
Création de un serveur

- Pour créer un serveur on va se servir de la classe **java.net.ServerSocket**
- Là encore il existe différents constructeurs et méthodes à explorer
- Deux méthodes vont être intéressantes :
  - **public ServerSocket(int port) throws IOException**
    - Crée une socket qui écoute sur un port et ne sert qu'à attendre les demandes de connexions
    - **port** est le numéro du port

```
ServerSocket server=new ServerSocket(4242);
```

  - **public Socket accept() throws IOException**
    - Attend une connexion
    - Retourne une socket qui servira (comme pour les clients à la communication

Et le nom de la machine ?




```
ServerSocket server=new ServerSocket(4242);
```

- En fait, c'est simple, le service se trouve sur la machine où vous exécutez le programme
- L'objet **server** n'a donc pas besoin de connaître le nom de la machine

Comportement d'un serveur

1) Création d'un objet **ServerSocket** qui est lié à un port
2) Attente de demande de connexion avec **accept**
3) Récupération de la Socket pour la communication
4) Récupération de flux d'entrée-sortie
5) Communication avec le client
6) Fermeture des flux et de la socket
7) Retour au point 2)

Exemple Serveur TCP

- On va programmer un serveur qui va
  - Attendre une connexion sur le port 4242
  - Envoyer un message `"Hi\n"`
  - Attendre un message du client
  - Afficher le message du client
  - Et recommencer à attendre une communication

Création d'un serveur TCP (2)

```
import java.net.*;
import java.io.*;
public class ServeurHi{
    public static void main(String[] args){
        try{
            ServerSocket server=new ServerSocket(4242);
            while(true){
                Socket socket=server.accept();
                BufferedReader br=new BufferedReader(new InputStreamReader(socket.getInputStream()));
                PrintWriter pw=new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
                pw.print("HI\n");
                pw.flush();
                String mess=br.readLine();
                System.out.println("Message recu :"+mess);
                br.close();
                pw.close();
                socket.close();
            } }
        catch(Exception e){
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Points importants

- Il y a donc différentes sockets dans notre serveur
  - Une **ServerSocket** qui sert uniquement à attendre les connexions
  - Une socket (de la classe **Socket**) par communication acceptée
- Là encore il ne faut pas oublier :
  - De *catcher* les exceptions et d'afficher les messages d'erreur correctement
  - De faire un **flush** pour les écritures afin de garantir que les messages sont bien envoyés
  - De fermer proprement les socket et les flux d'entrée-sortie
    - par exemple ne pas fermer la socket ServerSocket dans la boucle d'attente des connexions, vu que l'on en a besoin pour la prochaine connexion

Comment tester notre serveur

- Tout d'abord, rappelez vous que vous aurez au moins deux programmes
  1) Le programme du serveur
  2) Les programmes des clients
- DONC si vous voulez tester le tout il faudra **PLUSIEURS** terminaux !!!!
- Avant de programmer un programme, pourquoi ne pas faire un coup de **telnet** sur votre serveur
  - Pour le serveur que nous avons programmé, si il tourne sur **lulu**
    - **telnet lulu 4242**
    - Notre serveur enverra alors `"Hi\n"` que telnet affichera
    - Si nous tapons un message sur telnet suivi de Retour Chariot
    - Le serveur affichera notre message

Pour programmer notre client

- Cette fois-ci la machine est lulu
- Cette fois-ci le port est 4242

```
Socket socket=new Socket("lulu",4242);
```

- Qui commence par parler ?
  - Le Serveur, il envoie `"Hi\n"`
- Et ensuite ?
  - Le client doit envoyer un message se terminant par `"\n"`
  - En effet le serveur fait readLine(), il attend donc une chaîne de caractères se terminant par un retour à la ligne

Client TCP pour notre serveur

```
import java.net.*;
import java.io.*;
public class ClientHi{
    public static void main(String[] args){
        try{
            Socket socket=new Socket("lulu",4242);
            BufferedReader br=new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter pw=new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
            String mess=br.readLine();
            System.out.println("Message recu du serveur :"+mess);
            pw.print("HALLO\n");
            pw.flush();
            pw.close();
            br.close();
            socket.close();
        }
        catch(Exception e){
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Remarques

- Notre serveur n'accepte pas plusieurs connexions en même temps
  - Quand il a fait un **accept**, il communique avec un client et il doit avoir fini la communication pour parler avec un autre client
  - Nous verrons la prochaine fois comment modifier cela
- Un client qui commencerait par écrire au serveur et lire ensuite ce que le serveur fait fonctionnerait
  - MAIS mauvaise interprétation du protocole
  - On pourrait croire que le serveur répond "Hi" au "Hallo" du client

Mauvais client TCP pour notre serveur

```
import java.net.*;
import java.io.*;
public class ClientHiFalse{
    public static void main(String[] args){
        try{
            Socket socket=new Socket("lulu",4242);
            BufferedReader br=new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter pw=new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
            pw.print("HALLO\n");
            pw.flush();
            String mess=br.readLine();
            System.out.println("Message recu du serveur :"+mess);
            pw.close();
            br.close();
            socket.close();
        }
        catch(Exception e){
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Les informations liées à une socket

```
import java.net.*;
import java.io.*;
public class ClientEchoPort{
    public static void main(String[] args){
        try{
            Socket socket=new Socket("monjetas",7);
            System.out.println("La socket est connectee");
            System.out.println("Le port local est "+socket.getLocalPort());
            System.out.println("Le port distant est "+socket.getPort());
            System.out.println("La machine locale est "+socket.getLocalAddress().getHostName());
            System.out.println("La machine distante est "+socket.getInetAddress().getHostName());
            socket.close();
        }
        catch(Exception e){
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```