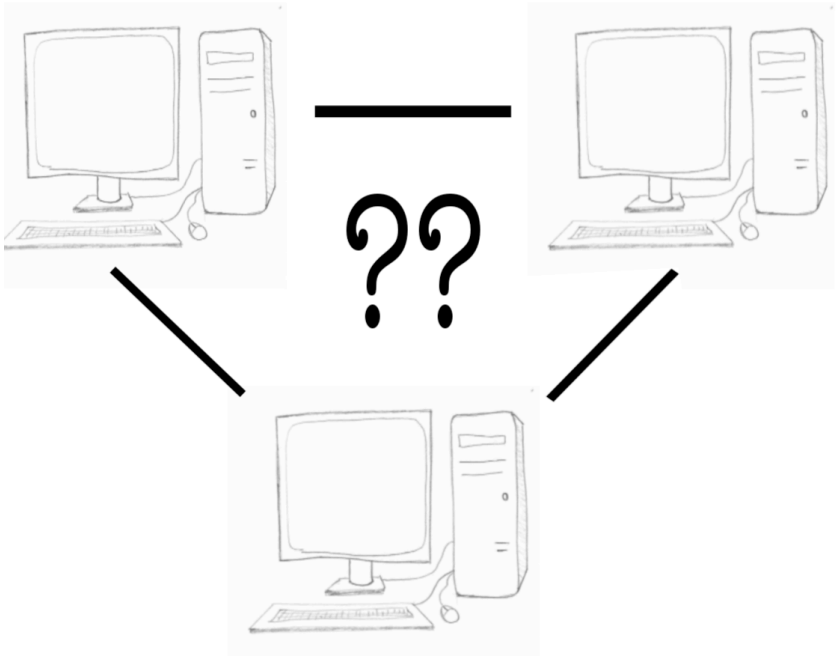
PROGRAMMATION RÉSEAU

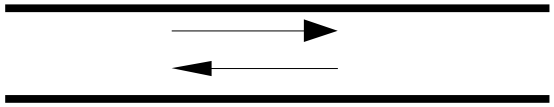
Arnaud Sangnier
sangnier@irif.fr

API TCP Java - II



Retour sur notre problème

- **TCP -> mode connecté**

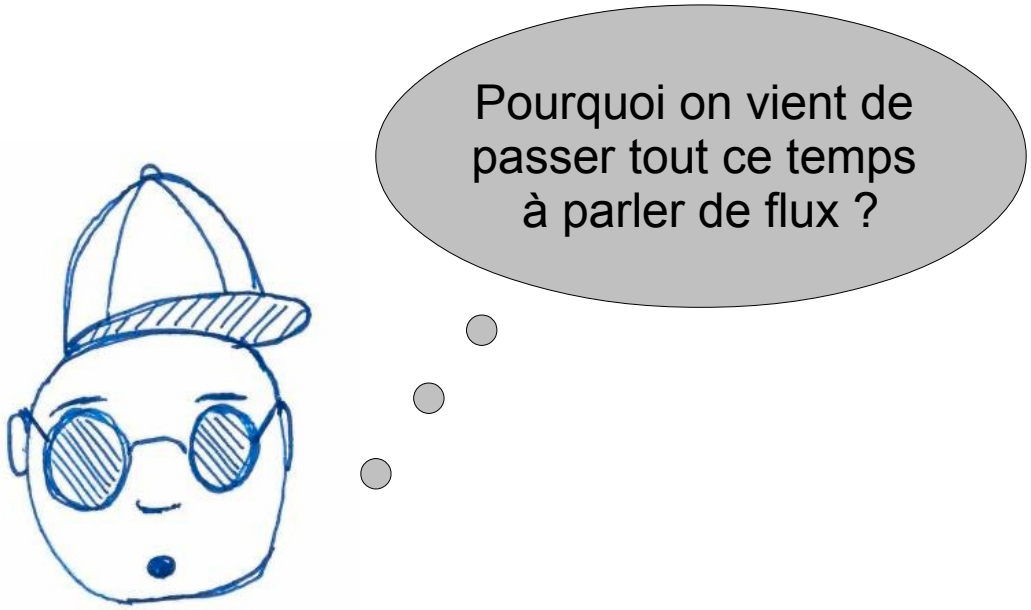


Client

Serveur

- Comment se connecter à un service ?
- Comment envoyer nos données vers un service ?
- Comment recevoir les données envoyées ?

Liens avec les flux et le réseau



- Il existe un outil, les **sockets** qui permet de se connecter à des machines et de faire passer toutes les communications avec des flux
- Donc c'est sur des flux que vous enverrez et recevrez des données
- Les fonctions vues précédemment vont donc être très utiles

Les sockets en bref

- Une socket est une connexion entre deux machines
- Elle passe donc par deux ports
- Opérations basiques pouvant être réalisées par une socket
  - Se connecter à une machine à distance
  - Envoyer des données
  - Recevoir des données
  - Fermer une connexion
  - Se connecter à un port
  - Attendre des données
  - Accepter des connexions de machines sur le port auquel elle est liée
- Les trois dernières opérations sont nécessaires pour les serveurs (cf la classe **ServerSocket**)

Client pour le service echo tcp

- Questions à se poser avant de programmer le client :
  - Sur quelle machine tourne le service avec lequel on souhaite communiquer
    - **lampe.informatique.univ-paris-diderot.fr**
  - Sur quel port écoute ce service
    - **port 7**
    - on peut retrouver cette information dans **/etc/services**
  - Quelle forme a la communication
    - On envoie des chaînes de caractères
    - Le client commence à 'parler'
    - Le serveur renvoie la même chaîne

Création de socket TCP

- On va créer une socket pour se connecter au port 7 de monjetas
- Utilisation de la classe **java.net.Socket**
  - Plusieurs constructeurs possibles (à explorer)
  - On va utiliser le suivant :
    - **public Socket(String host, int port) throws UnknownHostException, IOException**
    - **host** est le nom de la machine distante
    - **port** est le numéro du port

```
Socket socket=new Socket("lampe.informatique.univ-paris-
diderot.fr",7);
```

Création de socket TCP(2)

- Que se passe-t-il quand on fait :

```
Socket socket=new Socket("lampe.informatique.univ-paris-diderot.fr",7);
```

- Une socket est créé entre la machine locale et **lampe**
- La socket est attachée localement à un port éphémère (choisi par java)
- À la création de la socket, une demande de connexion est lancée vers **lampe** sur le **port 7**
- Si la connexion est acceptée, le constructeur termine normalement
- On peut ensuite utiliser la socket pour communiquer

Traiter correctement les exceptions

- La signature de la création de socket est :
  - **public Socket(String host, int port) throws UnknownHostException, IOException**
- En java, il est important de *catcher* les exceptions afin d'éviter qu'un programme crashe brutalement
- En programmation réseaux, c'est d'autant plus important car les problèmes peuvent être **multiples** :
  - mauvais numéro de port, hôte inconnu, problème de lecture ou d'écriture sur un flux

```
try{...}
catch(Exception e){
    System.out.println(e);
    e.printStackTrace();
}
```

Exemple fonctionnant sur lulu

```
import java.net.*;
import java.io.*;
public class ClientEcho{
    public static void main(String[] args){
      try{
       Socket socket=new Socket("lampe",7);
       socket.close();
      }
      catch(Exception e){
        System.out.println(e);
        e.printStackTrace();
      }
    }
}
```

Programme levant une exception sur lulu

```
import java.net.*;
import java.io.*;
public class ClientPb{
    public static void main(String[] args){
        try{
            Socket socket=new Socket("lulu",10);
            socket.close();
        }
        catch(Exception e){
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Question

- Il existe différents constructeurs dans la classe **Socket**
- Encore, plus simple, l'adresse de lampe est `192.168.70.237`
-

```
Socket socket=new Socket("192.168.70.237",7);
```

Utilisation de la socket créée

- Pour mettre fin à la connexion (c-à-d pour raccrocher) :
  - Utilisation de la méthode **void close()**
- Comment communique-t-on par la socket
  - la méthode **public InputStream getInputStream() throws IOException**
    - elle permet de récupérer un flux d'entrée sur lequel on va lire
  - la méthode **public OutputStream getOutputStream() throws IOException**
    - elle permet de récupérer un flux de sortie sur lequel on va écrire
- **Remarques :**
  - Les communications sur la socket sont **bi-directionnelles**
  - On utilise la même socket pour recevoir et envoyer des messages
  - **Par contre deux flux différents**

Récupération des flux

- Partie de code montrant comment on récupère les informations

```
Socket socket=new Socket("lampe",7);
InputStream is=socket.getInputStream();
OutputStream os=socket.getOutputStream();
```

- On a vu que l'on pouvait installer des filtres sur les objets des classes InputStream et OutputStream pour manipuler plus facilement les entrées-sorties

```
Socket socket=new Socket("lampe",7);
BufferedReader br=new BufferedReader(
    new InputStreamReader(socket.getInputStream()));
PrintWriter pw=new PrintWriter(
   new OutputStreamWriter(socket.getOutputStream()));
```

Création d'un client TCP (1)

- On va créer un client pour le service echo tcp (port 7) tournant sur lampe
- Voilà les étapes que notre client va faire
  1) Créer une socket pour se connecter au service
  2) Récupérer les flux d'entrée et sortie et les filtrer
  3) Envoyer une chaîne de caractères `"Hello"`
  4) Attendre et recevoir une chaîne de caractères du service
  5) Afficher la chaîne de caractères reçue
  6) Fermer les flux et la connexion

Création d'un client TCP (2)

```
import java.net.*;
import java.io.*;
public class ClientEcho{
    public static void main(String[] args){
        try{
            Socket socket=new Socket("lampe",7);
            BufferedReader br=new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter pw=new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
            pw.print("HELLO\n");
            pw.flush();
            String mess=br.readLine();
            System.out.println("Message recu :"+mess);
            pw.close();
            br.close();
            socket.close();
        }
        catch(Exception e){
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Les points importants

- Ne pas oublier de *catcher* les exceptions
- Utiliser la méthode flush() pour vider le buffer et envoyer le message
- La lecture avec readLine() bloque tant que l'on ne reçoit pas de message
- Exemple de code ne fonctionnant pas (sans flush le message n'est pas envoyé)

```
Socket socket=new Socket("lampe",7);
BufferedReader br=new BufferedReader(new
                  InputStreamReader(socket.getInputStream()));
PrintWriter pw=new PrintWriter(new
                  OutputStreamWriter(socket.getOutputStream()));
pw.print("HELLO\n");
String mess=br.readLine();
System.out.println("Message recu :"+mess);
```