Développement en JAVA

Communication TCP

Agenda

- Comment faire communiquer deux processus
- La couche TCP/IP
- La couche applicative
- La classe socket

Rappel de réseau

- Les communications à travers un réseau est normé par le modèle OSI
- Nos applications reposent donc sur la couche de transport
- Les protocoles de transport les plus courants sont TCP et UDP

Rappel de réseau

- Il existe plusieurs mode de communication à travers un réseau:
  - Client/Serveur
  - Paire à paire (p2p)
- Le protocole TCP permet d'établir une communication en mode connecté
- Le protocole UDP permet d'établir une communication en mode non connecté

Notion de socket

- Une socket est moyen pour deux applications de communiquer à travers le réseau
- Sur un OS Unix, une socket est un simple fichier texte dans lequel sont lues et écrit des données binaires. L'OS se chargeant ensuite de faire transiter les données à travers le réseau.
- Une socket permet donc d'envoyer ET de recevoir des données
- Une socket correspond à une communication établie entre deux machines du réseau sur un port donné

La classe ServerSocket

- La classe java.net.ServerSocket permet d'écouter les connexion entrantes sur un port donné

```
try{
  ServerSocket s = new ServerSocket(5000);
}catch(IOException e){
  System.err.println("Erreur de démarrage du serveur");
}
```

La classe Socket

- La classe java.net.socket permet de lire ou écrire des données à travers le réseau
- Côté serveur la méthode accept() permet d'obtenir une socket lorsque un client vient de se connecter

```
try{
  ServerSocket server = new ServerSocket(5000);
  Socket s = server.accept(); // Appel bloquant
  // Ici un utilisateur s'est connecté
}       (IOException e){
  System.err.println("Erreur de démarrage du serveur");
}
```

Fermeture des connexions

- Les classes Socket et ServerSocket doivent toujours être fermées après utilisation afin de libérer correctement les ressources

```
ServerSocket server;
Socket s;
try{
  server = new ServerSocket(5000);
  s = server.accept(); // Appel bloquant
}catch(IOException e){
  System.err.println("Erreur de démarrage du serveur");
}finally{
  if(server != null){
    server.close();
  }
  // Idem pour la socket
}
```

Utilisation d'un Socket

- Côté client on utilise directement la classe Socket
- Le constructeur de la classe prend deux paramètres
  - `String host -> l'adresse IP du destinataire`
  - `int port -> le port d'écoute du destinataire`
- Une fois l'objet construit, la connexion est établie
- Lors de l'établissement de la connexion deux erreurs peuvent survenir
  - IOException: erreur d'IO de base
  - UnknowException: l'adresse du serveur distant est inconnue

Utilisation d'une Socket

Côté Client

```
try {
  Socket s = new Socket("localhost", 5000);
  PrintWriter pr = new PrintWriter(s.getOutputStream());
  // Envoi des données à travers le réseau
  pr.write("Hello je suis un client");
  pr.flush();
}
```

Côté Serveur

```
try{
  server = new ServerSocket(5000);
  s = server.accept();
  BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
  // Affichage des données reçues
  System.out.println(br.readLine());
}
```

Connexions mutliples

- Côté serveur l'appel à accept() est **bloquant**
- Notre programme peu donc être dans deux état
  - Bloqué, en attente d'une requête
  - En train de traiter une requête
- Notre serveur ne peut donc traiter qu'une seule requête à la fois…
- Pour accepter plusieurs connexions simultanées, nous avons de parallélisme et d'utiliser des threads