Fiche n°2

Serveur en mode connecté multi-threadé

Cette fiche présente comment développer un serveur en mode connecté multi-threadé en Java, c'est-à-dire un serveur multi-clients.

1 Le programme serveur

Dans un premier temps, le serveur doit créer une socket de connexion qui permettra d'accepter des connexions de la part des clients. Le port d'écoute est spécifié ici dans un attribut `portEcoute`.

```
ServerSocket socketServeur = null;
try {
  socketServeur = new ServerSocket(portEcoute);
} catch(IOException e) {
  System.err.println("Création de la socket impossible : " + e);
  System.exit(0);
}
```

Une fois la socket créée, le serveur se met en attente de connexions à l'aide de la méthode `accept`. Pour pouvoir traiter les clients en parallèle, le serveur crée un *thread* à chaque nouvelle connexion et se remet en attente immédiatement d'une nouvelle connexion.

```
try {
  Socket socketClient;
  while(true) {
    System.out.println("En attente d'une connexion...");
    socketClient = socketServeur.accept();
    ThreadConnexion t = new ThreadConnexion(socketClient);
    t.start();
  }
} catch(IOException e) {
  System.err.println("Erreur lors de l'attente d'une connexion : " + e);
  System.exit(0);
}
```

Les échanges entre le client et le serveur sont gérés dans la méthode principale du *thread*.

2 Le *thread* de communication

La socket de communication propre à chaque client est créée dans le serveur lors de l'appel à la méthode `accept`. Pour gérer les échanges dans le *thread*, nous avons choisi de passer la socket dans le constructeur. Nous en profitons pour créer les flux. Le code suivant présente les attributs et le constructeur.

```
// Attributs
private BufferedReader input;
private PrintWriter output;
private Socket socketClient;

// Constructeur
public ThreadConnexion(Socket socketClient) {
  this.socketClient = socketClient;
  try {
    input = new BufferedReader(
        new InputStreamReader(socketClient.getInputStream()));
    output = new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(socketClient.getOutputStream())), true);
  } catch(IOException e) {
    System.err.println("Association des flux impossible : " + e);
    System.exit(0);
  }
}
```

Lorsque le *thread* est démarré dans le serveur à l'aide de la méthode `start`, la méthode `run` est ensuite exécutée. C'est donc dans cette méthode que le serveur (le *thread*) peut lire des données envoyées par le client :

```
String message = "";
try {
  message = input.readLine();
} catch(IOException e) {
  System.err.println("Erreur lors de la lecture : " + e);
  System.exit(0);
}
```

Ou envoyer des données au client :

```
String message = "Bonjour";
System.out.println("Envoi: " + message);
output.println(message);
```

L'envoi et la réception doivent respecter un dialogue d'échange entre le client et le serveur. La lecture d'un côté doit correspondre à une écriture de l'autre.

3 Le programme client

Le client doit créer une socket de communication à l'aide de la classe `Socket`. Le constructeur prend en paramètre l'adresse (ou un nom de domaine) et un port d'écoute (ici `portEcoute`, un attribut de la classe).

```
Socket socket = null;
try {
  socket = new Socket("localhost", portEcoute);
} catch(UnknownHostException e) {
  System.err.println("Erreur sur l'hôte : " + e);
  System.exit(0);
} catch(IOException e) {
  System.err.println("Création de la socket impossible : " + e);
  System.exit(0);
}
```

Comme pour le serveur, nous encapsulons les flux d'entrée/sortie de la socket dans des flux qui nous permettront de lire des données de type primitif ou des chaînes de caractères.

```
BufferedReader input = null;
PrintWriter output = null;
try {
  input = new BufferedReader(
      new InputStreamReader(socket.getInputStream()));
  output = new PrintWriter(new BufferedWriter(
      new OutputStreamWriter(socket.getOutputStream())), true);
} catch(IOException e) {
  System.err.println("Association des flux impossible : " + e);
  System.exit(0);
}
```

Comme pour le serveur, le client peut alors envoyer ou recevoir des données.

4 Exécution

Pour tester le client et le serveur, dans un premier temps, vous devez compiler les classes puis exécuter le serveur. Le client ne peut être démarré qu'une fois le serveur démarré. À noter que le serveur ne s'arrête pas une fois les échanges terminés avec un client. Il est possible d'exécuter plusieurs clients, même si le temps d'exécution du *thread* est trop court pour observer le traitement en parallèle.

Le numéro de port d'écoute du serveur est spécifié via une constante. S'il est déjà utilisé, il est possible de le modifier (dans les deux classes). Une première amélioration consiste à spécifier le numéro de port en argument du serveur (ce qui évite de recompiler la classe à chaque changement). Une autre solution consiste à laisser le système choisir le numéro port automatiquement (comme pour le client) et de l'afficher à l'écran pour pouvoir le spécifier ensuite en argument au client.