Fiche n°1

Sockets en mode non connecté

Cette fiche présente comment comment envoyer des données en utilisant des sockets en mode non connecté.

1 Le programme serveur

Dans un premier temps, le serveur doit créer une socket à l'aide de la classe `DatagramSocket`. Le port d'écoute doit être spécifié ; l'adresse IP correspondant à l'adresse IP par défaut de la machine. Ici, la variable `portEcoute` contient le port d'écoute.

```
DatagramSocket socket = null;
try {
  socket = new DatagramSocket(portEcoute);
} catch(SocketException e) {
  System.err.println("Erreur lors de la création de la socket : " + e);
  System.exit(0);
}
```

Pour recevoir des données, il faut créer un objet `DatagramPacket` qui nécessite un tampon de réception (un tableau d'octets). Le tableau est rempli par le message reçu après l'appel à la méthode `receive`. La taille spécifiée dépend de l'application (ici, 1024 est une valeur arbitraire).

```
byte[] tampon = new byte[1024];
DatagramPacket msg = new DatagramPacket(tampon, tampon.length);
```


Si un message supérieur à la taille spécifiée est reçu, le message sera tronqué sans génération d'exception.

Le serveur peut ensuite se mettre en attente de réception d'un message grâce à la méthode `receive` de la socket.

```
try {
  socket.receive(msg);
  String texte = new String(msg.getData(), 0, msg.getLength());
  System.out.println("Lu: " + texte);
} catch(IOException e) {
  System.err.println("Erreur lors de la réception du message : " + e);
  System.exit(0);
}
```

La méthode `getData` retourne le tableau d'octets rempli avec le message reçu. À noter que pour l'interpréter, il faut connaître la taille des données reçues qui est retournée par la méthode `getLength`.

2 Le programme client

Pour envoyer un message, le client doit créer une socket. Comme il ne recevra pas de message en premier et que nous n'avons pas besoin d'utiliser un numéro de port spécifique, nous ne spécifions pas de numéro de port lors de la création. Le système en attribue un libre.

```
DatagramSocket socket = null;
try {
  socket = new DatagramSocket();
} catch(SocketException e) {
  System.err.println("Erreur lors de la création de la socket : " + e);
  System.exit(0);
}
```

Le client doit ensuite créer un message qu'il enverra au serveur. Il prend en paramètres l'adresse du serveur, le numéro de port sur lequel il écoute et les données à envoyer. La classe `InetAddress` permet de récupérer une adresse à partir d'un nom de domaine ou d'une adresse IP. La valeur `null` correspond au *localhost*.

```
DatagramPacket msg = null;
try {
  InetAddress adresse = InetAddress.getByName(null);
  String message = "Bonjour";
  byte[] tampon = message.getBytes();
  msg = new DatagramPacket(tampon, tampon.length, adresse, portEcoute);
} catch(UnknownHostException e) {
  System.err.println("Erreur lors de la création du message : " + e);
  System.exit(0);
}
```

Il ne reste plus qu'à envoyer le message à l'aide de la socket créée :

```
try {
  socket.send(msg);
} catch(IOException e) {
  System.err.println("Erreur lors de l'envoi du message : " + e);
  System.exit(0);
}
```

3 Exécution

Pour tester le client et le serveur, 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 s'arrête une fois le message lu.

Le numéro de port d'écoute du serveur est spécifié via une constante. S'il est déjà utilisé, une exception est levée. 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.