Fiche n°2

Envoi d'objets en mode non connecté

Cette fiche illustre une application client/serveur qui permet d'envoyer un objet sérialisé via une socket en mode non connecté.

Les différentes classes présentées ici correspondent à une application client/serveur en mode non connecté. Le client envoie un objet `Personne` (la classe n'est pas présentée ici) au serveur qui l'affiche une fois reçu.

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 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).

```
DatagramPacket msgRecu = null;
try {
  byte[] tampon = new byte[1024];
  msgRecu = new DatagramPacket(tampon, tampon.length);
  socket.receive(msgRecu);
} catch(IOException e) {
  System.err.println("Erreur lors de la réception du message : " + e);
  System.exit(0);
}
```

Pour convertir le tableau d'octets en objet (désérialisation), le serveur utilise un flux d'octets (`ByteArrayInputStream`) sur le tableau de données reçu qu'il encapsule dans un flux d'objets (`ObjectInputStream`). La méthode `readObject` permet de lire le prochain objet dans le flux. Un objet de type `Object` est reçu que l'on peut transtyper dans le type choisi.

```
try {
  ByteArrayInputStream bais = new ByteArrayInputStream(msgRecu.getData()
     );
  ObjectInputStream ois = new ObjectInputStream(bais);
  Personne p = (Personne) ois.readObject();

  System.out.println("Recu␣:␣" + p);
} catch(ClassNotFoundException e) {
  System.err.println("Objet␣reçu␣non␣reconnu␣:␣" + e);
  System.exit(0);
} catch(IOException e) {
  System.err.println("Erreur␣lors␣de␣la␣récupération␣de␣l'objet␣:␣" + e)
     ;
  System.exit(0);
}
```

Si l'objet reçu est transtypé dans un type invalide, une exception est levée. Il est possible de récupérer le type de l'objet à l'aide de l'introspection.

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);
}
```

Pour envoyer l'objet `Personne`, nous devons le convertir en tableau d'octets. Pour cela, nous utilisons un flux d'octets (`ByteArrayOutputStream`), qui est encapsulé dans un flux d'objets (`ObjectOutputStream`). L'objet peut donc être sérialisé avec la méthode `writeObject` et il est possible de récupérer le tableau d'octets correspondant.

```
Personne p = new Personne("Cyril", "Rabat");
ByteArrayOutputStream baos = new ByteArrayOutputStream();
try {
  ObjectOutputStream oos = new ObjectOutputStream(baos);
  oos.writeObject(p);
} catch(IOException e) {
  System.err.println("Erreur␣lors␣de␣la␣sérialisation␣:␣" + 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*. Les données sont récupérées depuis le flux d'objets à l'aide de la méthode `toByteArray`.

```
try {
  byte[] donnees = baos.toByteArray();
  InetAddress adresse = InetAddress.getByName("localhost");
  DatagramPacket msg = new DatagramPacket(donnees, donnees.length,
                                          adresse, portEcoute);
  socket.send(msg);
  System.out.println("Message envoyé.");
} catch(UnknownHostException e) {
  System.err.println("Erreur lors de la création de l'adresse : " + e);
  System.exit(0);
} 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.