Réseaux Avancés
M1 GL 2023/2024 - TP N° 3 (Sockets UDP)

PS : pour ce TP, vous avez le choix de faire l'exo 1, 2 et 3 ou l'exo 1, 2 et 4.

Exercice 1 : Envoyer un objet en multicast avec UDP

Soit les 3 classes java suivantes :

```
import java.io.Serializable;
public class Entreprise implements Serializable {
private int id;
private String name;
public Entreprise (int id, String name) {
  this.id = id;
  this.name = name;
}
public int getId() {
 return id;
}
public String getName() {
 return name;
}
public String toString() {
 return "Id = " + getId() + " Name = " + getName();
}
}
```

```
import java.io.*;
import java.net.*;
public class UDPSocketClient {
public static void main(String[] args) throws Exception {
Entreprise entreprise = new Entreprise(10, "SOGERHWIT");
DatagramSocket Socket = new DatagramSocket();
InetAddress serveur = InetAddress.getByName("localhost");
byte[ ] buffer = new byte[1024];
// définir une structure dans laquelle les données sont écrites dans un tableau d'octets
ByteArrayOutputStream a = new ByteArrayOutputStream ( ); // classe fille de OutputStream
ObjectOutputStream b = new ObjectOutputStream(a);
b.writeObject (entreprise);
//copier le contenu de a (OutputStream) dans un tableau d'octets pour l'utiliser dans DatagramPacket
byte[ ] data = a.toByteArray();
DatagramPacket envoyer = new DatagramPacket(data, data.length, serveur, 9000);
Socket.send(envoyer);
System.out.println("Objet envoyé par le client");
DatagramPacket recu = new DatagramPacket(buffer,buffer.length);
Socket.receive(recu);
String response = new String(recu.getData());
System.out.println("Réponse du serveur : " + response);
}
}
```

```
import java.io.*;
import java.net.*;
public class UDPSocketServer {
public static void main(String[] args) throws Exception {
DatagramSocket socket = new DatagramSocket(9000);
byte[] buffer = new byte[1024];
DatagramPacket recu = new DatagramPacket(buffer, buffer.length);
socket.receive(recu);
byte[ ] data = recu.getData();
//ByteArrayInputStream classe fille de InputStream
ObjectInputStream a = new ObjectInputStream(new ByteArrayInputStream(data));
Entreprise entreprise = (Entreprise) a.readObject();
System.out.println("L'objet entreprise reçu : "+entreprise);
InetAddress IPAddress = recu.getAddress();
int port = recu.getPort();
String reponse = "J'ai bien reçu l'objet";
byte[ ] d = reponse.getBytes();
DatagramPacket envoyer =new DatagramPacket(d, d.length, IPAddress, port);
socket.send(envoyer);
}
}
```

- Lancer le serveur ainsi que le client et examiner les différentes classes et méthodes utilisées pour envoyer un objet avec UDP.

- Dans ce qui, votre objectif est **d'envoyer l'objet précèdent à un groupe multicast.**

Le premier octet d'une adresse **IP multicast** commence toujours par les 4 bits **'1110'** suivi par **28 bits** (adresse du groupe multicast).
11100000 => 224 **(valeur minimale pour le premier octet).**
11101111 => 239 **(valeur maximale pour le premier octet).**
Le protocole IP utilise les adresses IP (virtuelles) de : **224.0.0.0 à 239.255.255.255** pour le multicast.
Examiner dans ce qui suit le lien suivant :https://download.java.net/java/early_access/panama/docs/api/java.base/java/net/MulticastSocket.html

- Quel est le rôle de cette classe ? Examiner en particulier les deux méthodes joinGroup et leaveGroup.
- Soit les deux classes java suivantes :

```
import java.io.*;
import java.net.*;
public class Recepteur {
 public static void main(String argv [ ]) throws IOException{
        InetAddress group = InetAddress.getByName ("230.0.0.0");
        MulticastSocket socket = new MulticastSocket (1000) ;
        socket.joinGroup (group);
        byte [ ] buf = new byte[1024];
        DatagramPacket recv = new DatagramPacket(buf, buf.length);
        System.out.println("En attente de reception ...");
        socket.receive(recv);
        String ch= new String (recv.getData());
        System.out.println("Debut reception\n " +  ch+ " \nFin reception");
 }
}
```

```
import java.io.*;
import java.net.*;
public class Emetteur {
 public static void main(String argv [ ]) throws IOException{
        String msg = "JE SUIS UN ETUDIANT INFORMATIQUE";
        InetAddress group = InetAddress.getByName ("230.0.0.0");
        DatagramSocket socket = new DatagramSocket () ;
        DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(),group, 1000);
        socket.send(hi);
        System.out.println("Fin emission");
 }
}
```

- Lancer Recepteur.java (Examiner l'exécution), ensuite lancer Emetteur.java (Examiner à nouveau l'exécution coté Recepteur). Que fait ce code ?

  PS : Si une Exception est générée coté Recepteur, remplacer dans Receptreur.java : les 3 lignes :

```
InetAddress group = InetAddress.getByName ("230.0.0.0");
MulticastSocket socket = new MulticastSocket (1000) ;
socket.joinGroup (group);
```

  Par:

```
InetAddress groupm = InetAddress.getByName ("230.0.0.0");
InetSocketAddress group = new InetSocketAddress(groupm,0);
MulticastSocket socket = new MulticastSocket (1000) ;
socket.joinGroup (group, null);
```

  La signature **joinGroup(InetAddress mcastaddr)** est **Deprecated** depuis **JDK 14.**
  Voir le lien :
  https://download.java.net/java/early_access/panama/docs/api/java.base/java/net/MulticastSocket.html#joinGroup(java.net.InetAddress)

- Modifier l'adresse IP **"230.0.0.0"** par **"200.0.0.0".** Lancer une exécution. C'est quoi le problème ?
- Modifier les deux classes précédentes (**Emetteur.java et Recepteur.java**) afin d'envoyer l'objet **entreprise (new Entreprise (10, "SOGERHWIT"))** vers le groupe multicast **235.1.1.1** à l'écoute sur le port **4000.**
  Le **Recepteur** affiche au début : En attente de reception ...
  L'Emetteur envoie l'objet et affiche : Fin emission
  Le **Recepteur** affiche par la suite : L'objet entreprise recu : Id = 10 Name = SOGERHWIT

Exercice 2 : Manipulation de plusieurs types avec UDP

Soit les deux classes java suivantes :

```
import java.io.*;
import java.net.*;
public class ClientUDPint {
public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket();
        InetAddress serveur = InetAddress.getByName("localhost");
        ByteArrayOutputStream a = new ByteArrayOutputStream();
        DataOutputStream b = new DataOutputStream(a);
        // Ecrire : 10 true bonjour 1.2 dans le outputstream
        b.writeInt(10);
        b.writeBoolean(true);
        b.writeUTF("bonjour");
        b.writeDouble(1.2);
        b.flush();
       byte[] buffer = a.toByteArray();
       DatagramPacket packet = new DatagramPacket(buffer,buffer.length,serveur ,2000);
       socket.send(packet);
       System.out.println("Client a envoyé 10 true bonjour 1.2 au serveur");
}
}
```

```
import java.io.*;
import java.net.*;
public class ServeurUDPint {
    public static void main(String[] args) throws Exception {
         DatagramSocket socket = new DatagramSocket(2000);
        DatagramPacket packet = new DatagramPacket(new byte[1024] , 1024);
        socket.receive(packet);
        byte[] data = packet.getData();
        ByteArrayInputStream a = new ByteArrayInputStream(data);
         DataInputStream b = new DataInputStream(a);
         System.out.println("Entier recu : "+b.readInt());
         System.out.println("Boolean recu : "+b.readBoolean());
         System.out.println("String recu : "+b.readUTF());
         System.out.println("Double recu : "+b.readDouble());
}
}
```

- Lancer l'exécution des deux classes précédentes. Quel est l'intérêt de DataOutputStream et DataInputStream pour UDP ?

- Déclarer maintenant un tableau d'entier **(ex. int [ ] tab = {1, 6, 8, 9, 13, 10};)** coté client. Ce dernier doit l'envoyer par UDP au serveur. Le serveur affichera par la suite son contenu (les entiers avec des sauts de lignes). Proposer **deux solutions différentes** (la première considère **tab** comme un objet (même code que pour envoyer **entreprise :** faire un seul writeObject), la seconde envoie les éléments du tableau un par un : writeInt de tous les éléments du tableau).

Exercice 3 : Application client/serveur avec UDP

Dans ce qui suit, nous allons créer les objets suivants coté serveur (qui seront insérés dans un ArrayList) :

```
Joueur a= new  Joueur(9, "BENZEMA", "REAL");

Joueur b= new  Joueur(10, "MODRIC", "REAL");

Joueur c= new  Joueur(26, "MAHREZ","City");

Joueur d= new  Joueur(30, "MESSI", "PSG");

Joueur e= new  Joueur(10, "NEYMAR", "PSG");
```

Vous allez commencer par la création de la classe Joueur qui comporte les champs suivants : **numero (int), nom (String) et equipe (String).**

Vous devrez par la suite réaliser les traitements suivants avec UDP :

- Le client envoie le nom d'une équipe au serveur (ex. REAL), le serveur doit répondre par un objet de type ArrayList contenant les joueurs de cette équipe. Si vous utilisez un buffer de taille supérieur à la taille du mot reçu (coté serveur), utiliser la méthode **trim()** après la conversion du buffer d'octets en **String** pour supprimer les espaces liés aux cases non remplies dans le buffer de réception.
  Le résultat attendu est comme suit : **[numero = 9 nom = BENZEMA equipe = REAL, numero = 10 nom = MODRIC equipe = REAL].**

- Le client envoie un caractère au serveur (ex. M), le serveur doit répondre par un objet de type ArrayList contenant les joueurs dont le nom commence par ce caractère.
  Le résultat attendu dans le cas de l'envoi de 'M' sera comme suit : **[numero = 30 nom = MESSI equipe = PSG, numero = 26 nom = MAHREZ equipe = City, numero = 10 nom = MODRIC equipe = REAL].**

Exercice 4 : Opérations mathématiques avec UDP

Dans cet exercice, **ADD, MUL, DIV, SOUS et PUIS** correspondent respectivement aux opérations **d'addition, de multiplication, de division, de soustraction et de l'élévation à une puissance.**

Votre objectif est de réaliser une application client/serveur en utilisant les sockets UDP.

Le client envoie au serveur deux opérandes ainsi que l'opération à effectuer (les 3 sous forme d'une seule chaîne de caractères), le serveur répond par le résultat de l'opération.

Par exemple, si le client envoie au serveur :

- **ADD 6 1**, ce dernier répondra par **7.**
- **MUL 10 88**, ce dernier répondra par **880.**
- **SOUS 5 9**, ce dernier répondra par **-4.**
- **DIV 9 8**, ce dernier répondra par **1.125. (Division réelle).**
- **PUIS 3 2**, ce dernier répondra par **9 (3^2).**
- **RSD 6 7**, ce dernier répondra par **Erreur.**

PS : Vous pouvez utiliser la méthode **split** de la classe **String** https://docs.oracle.com/javase/7/docs/api/java/lang/String.html afin de convertir coté serveur une chaîne de caractères **(ex. ADD 6 1)** en tableau de chaîne de caractères **(ex. String [ ] tab = "ADD 6 1".split(" ");).**

Ou passer par la classe **StringTokenizer** du package **java.util :** https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html.

```
java.util.StringTokenizer st = new java.util.StringTokenizer("ADD 6 1");

String operation = st.nextToken();

int oprnd1 = Integer.parseInt(st.nextToken());

int oprnd2 = Integer.parseInt(st.nextToken());
```