ADRESSE IPV4 :

Un objet de la classe **InetAddress** représente une adresse Internet de type IPV4. Elle contient des méthodes pour lire une adresse, la comparer avec une autre ou la convertir en chaîne de caractères.

Elle ne possède pas de constructeur : il faut utiliser certaines méthodes statiques de la classe pour obtenir une instance de cette classe :

Méthode	Rôle
InetAddress getByName(String)	Renvoie l'adresse internet associée au nom d'hôte fourni en paramètre
InetAddress[] getAllByName(String)	Renvoie un tableau des adresses internet associées au nom d'hôte fourni en paramètre
InetAddress getLocalHost()	Renvoie l'adresse internet de la machine locale
byte[] getAddress()	Renvoie un tableau contenant les 4 octets de l'adresse internet
String getHostAddress()	Renvoie l'adresse internet sous la forme d'une chaîne de caractères
String getHostName()	Renvoie le nom du serveur

```
InetAddress adrresseServeurAstier = InetAddress.getByName("astier.elycee.rhonealpes.fr");
System.out.println("Adresse LPO Astier = "+adrresseServeurAstier.getHostAddress());
```

→ `Adresse LPO Astier = 80.247.224.245`

PROTOCOLE TCP :

TCP est un protocole (avec connexion) qui permet une connexion de type point à point entre deux applications.

C'est un protocole fiable qui garantit la réception dans l'ordre d'envoi des données.

En contrepartie, ce protocole offre de moins bonnes performances que l'UDP.

TCP utilise la notion de port pour permettre à plusieurs applications d'utiliser TCP.

Dans une liaison entre deux ordinateurs, l'un des deux joue le rôle de serveur et l'autre celui de client.

TCP, comme UDP, utilise le numéro de port pour identifier les applications. À chaque extrémité (client/serveur) de la connexion TCP est associé un numéro de port sur 16 bits (de 1 à 65535) assigné à l'application émettrice ou réceptrice.

1. Le server : classe « ServerSocket » :

La classe **ServerSocket** est utilisée côté serveur : elle attend simplement les appels du ou des clients.

Constructeur	Rôle
ServerSocket(int)	Créer une socket sur le port fourni en paramètre

Méthode	Rôle
Socket accept()	Attendre une nouvelle connexion
void close()	Fermer la socket

1-1 Le Serveur TCP mono-client (mono-Thread) :

Le mise en œuvre de la classe **ServerSocket** suit toujours la même logique :

- Créer une instance de la classe **SocketServer** en précisant le port en paramètre,
- définir une boucle sans fin contenant les actions ci-dessous,
- appelle de la méthode **accept()** qui renvoie une socket lors d'une nouvelle connexion,
- obtenir un **flux en entrée et en sortie** à partir de la socket,
- écrire les **traitements** à réaliser.

2. Le client : classe « Socket »:

La classe **Socket** est utilisée côté client : Elle gère la connexion, l'envoi de données, la réception de données et la déconnexion..

Constructeur	Rôle
Server(InetAddress, int)	Créer une socket sur la machine dont l'adresse IP et le port sont fournis en paramètres

Méthode	Rôle
InputStream getInputStream()	Renvoie un flux en entrée pour recevoir les données de la socket
OutputStream getOutputStream()	Renvoie un flux en sortie pour émettre les données de la socket la socket
void close()	Fermer la socket

2-1 Le client TCP:

- Créer une instance de la classe **Socket** en précisant la machine et le port en paramètres,
- obtenir un **flux en entrée et en sortie,**
- écrire les **traitements** à réaliser.

PROTOCOLE UDP :

UDP est un protocole sans connexion. Il a été développé pour permettre une transmission des données très simple sans détection d'erreur d'aucune sorte. Cependant, il est très bien adapté pour les applications de type requête/réponse, comme par exemple DNS, etc. En contrepartie, ce protocole offre plus de rapidité que TCP .

1. Principe de la communication « mode datagramme » :

- La partie serveur crée une socket et la lie à un port UDP particulier
- La partie client crée une socket pour accéder à la couche UDP et la lie sur un port quelconque
- Le serveur se met en attente de réception de paquet sur sa socket
- Le client envoie un paquet via sa socket en précisant l'adresse du destinataire

- Couple @IP/port
- Destinataire = partie serveur
  - @IP de la machine sur laquelle tourne la partie serveur et numéro de port sur lequel est liée la socket de la partie serveur
- Il est reçu par le serveur ou pas (pb réseau…)

- Si le client envoie un paquet avant que le serveur ne soit prêt à recevoir : le paquet est perdu.

2. Sockets UDP en Java : « DatagramSocket » et « DatagramPacket »

2-1 « DatagramSocket » crée un Socket qui utilise le protocole UDPpour émettre ou recevoir des données.

Constructeur	Rôle
DatagramSocket()	Créer une socket attachée à toutes les adresses IP de la machine et à un des ports libres sur la machine ***(coté client)***
DatagramSocket(int)	Créer une socket attachée à toutes les adresses IP de la machine et au port précisé en paramètre ***(coté serveur)***

Méthode	Rôle
receive(DatagramPacket)	Recevoir des données
send(DatagramPacket)	Envoyer des données
close()	Fermer la socket
void setSoTimeout(int)	Préciser un timeout d'attente pour la réception d'un message.

Par défaut la méthode receive() est bloquante jusqu'à la réception d'un packet de données. La méthode setSoTimeout() permet de préciser un timeout en millisecondes. Une fois ce délai écoulé sans réception d'un paquet de données, la méthode lève une exception du type SocketTimeoutException.

2-2 « DatagramPacket» crée un Socket qui utilise le protocole UDPpour émettre ou recevoir des données.

Constructeur	Rôle
DatagramPacket(byte tampon[], int taille)	Encapsule des paquets en réception dans un tampon ***(coté serveur)***
DatagramPacket(byte port[], int taille, InetAddress adresse, int port)	Encapsule des paquets en émission à destination d'une machine ***(coté client)***

Méthode	Rôle
byte[] getData()	Renvoyer les données contenues dans le paquet
setData(byte[])	Mettre à jour les données contenues dans le paquet

Format des données à transmettre :

- Très limité : ***tableaux de byte***
  - Pour tenir dans un seul ***datagramme IP***, le datagramme UDP ne doit pas contenir plus de ***65467*** octets de données.
  - Pour être certain de ne pas perdre de données : ***512 octets max***
  - Si datagramme UDP trop grand : les données sont tronquées
  - Si tableau d'octets en réception est plus petit que les données envoyées, les données reçues sont tronquées.
- Doit donc pouvoir convertir
  - Un objet quelconque en byte[] pour l'envoyer
  - Un byte[] en un objet d'un certain type après réception
- Solutions
  - Utiliser les flux Java pour conversion automatique

En écriture : conversion de Object en byte[]

```
ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
ObjectOutputStream objectStream = new ObjectOutputStream(byteStream);
objectStream.writeObject(object);
byte[] byteArray = byteStream.toByteArray();
```

En lecture : conversion de byte[] en Object

```
ByteArrayInputStream byteStream =new ByteArrayInputStream(byteArray);
ObjectInputStream objectStream = new ObjectInputStream(byteStream);
object = objectStream.readObject();
```