

## M.3.3.2 Programmation Objet Avancée

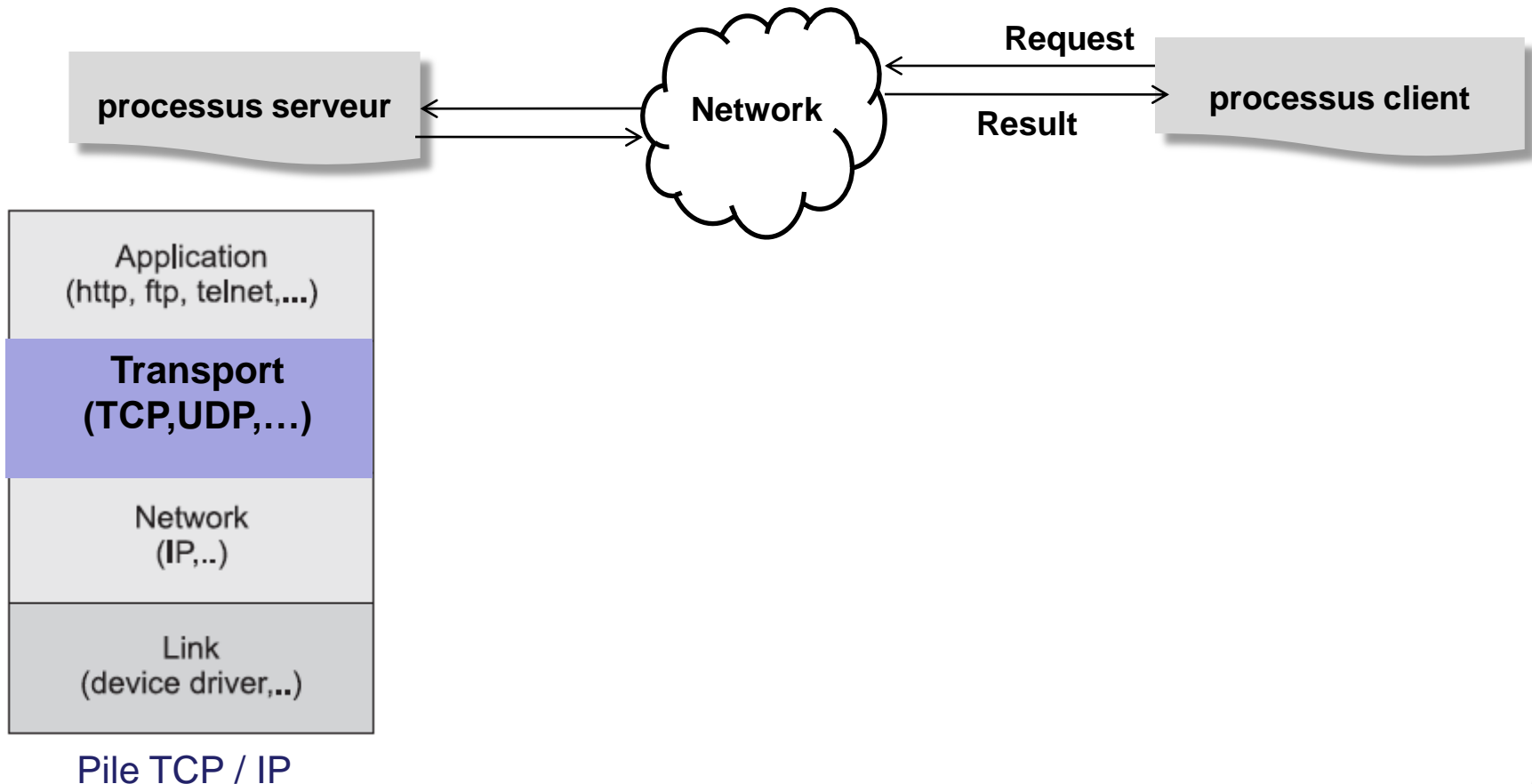
# Sockets Java et réseau





- **Paradigme client/serveur**
- **Sockets TCP**
- **Servir plusieurs clients**
- **Adressage**

**Clients** et **serveurs** impliquent des services réseau fournis par la **couche de transport**



**Clients** et **serveurs** impliquent des services réseau fournis par la **couche de transport**

→ **TCP : Transmission Control Protocol**

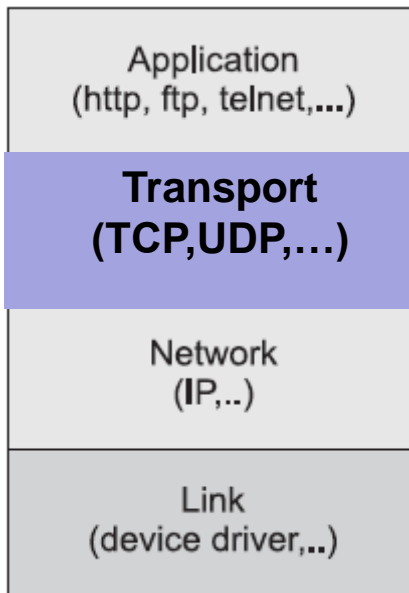
⊕ **Protocole sûr** puisqu'il garantit l'ordre d'arrivée des données envoyées

- Téléphonie

⊖ **Vitesse de connexion lente**

- Un nombre important d'A/R

— **HTTP** est basé sur le protocole TCP



Pile TCP / IP

**Clients** et **serveurs** impliquent des services réseau fournis par la couche de transport

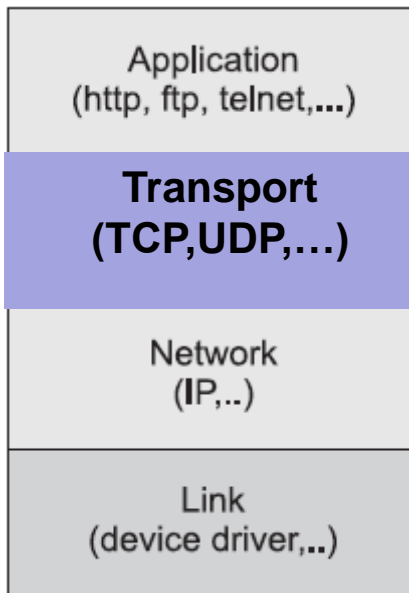
→ **UDP : User Datagram Protocol**

⊖ **Peu fiable** (poste)

- *Il n'est pas sûr que les données arrivent à destination ni dans le bon ordre*

⊕ **Rapide** (peu d'A/R)

- *Utilisé pour les échanges dont la perte de paquets n'est pas important (vidéocast, VoIP ...)*

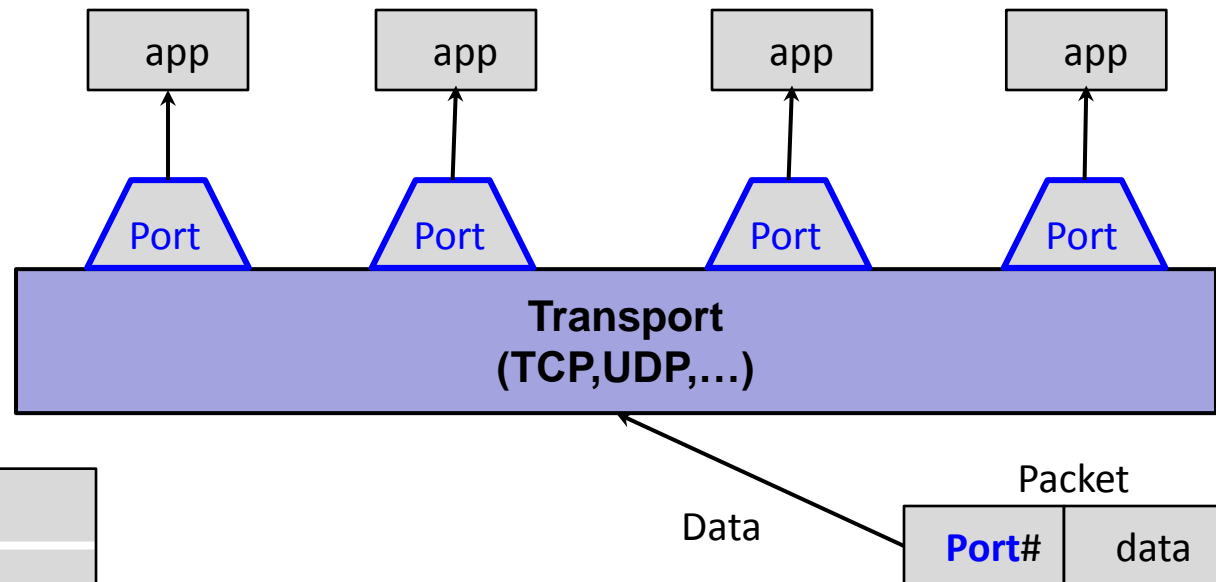


Pile TCP / IP

**TCP** ou **UDP** utilisent des numéros de **ports** pour mapper les données entrantes à un **service** particulier

Ports logiciels  
**réservés :**

ftp	21	tcp
telnet	23	tcp
smtp	25	tcp
http	80	tcp,udp
https	443	tcp,udp



Sous *UNIX*, les ports logiciels **< 1024** ne peuvent être attribués que si le programme est exécuté en **root**. (Services implémentés par le **SE**)



La couche de transport utilise le mécanisme de **sockets**

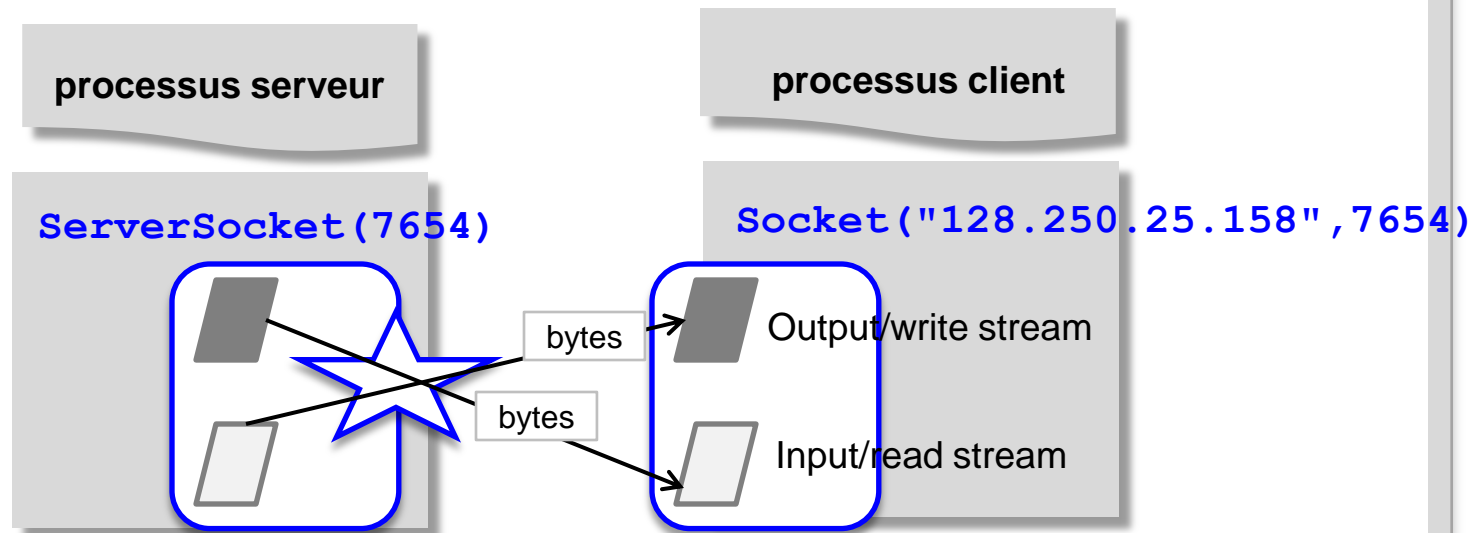
```
import java.net.*; import java.io.*;
public class PortScanner {
    public static void main(String[] args) {
        String host = "localhost";
        try {
            InetAddress adr = InetAddress.getByName(host);
            for (int i = 1; i < 1024; i++) {
                try {
                    Socket laSocket = new Socket(adr, i);
                    System.out.println("Il y a un serveur sur le port " + i + " de " + host);
                }
                catch (IOException ex) { // ne doit pas y avoir de serveur sur ce port
                }
            } // end for
        } // end try
        catch (UnknownHostException e) {
            System.err.println(e);
        } // end main
    } // end PortScanner
```

Ports sur  
écoute par des  
serveurs tcp

Il y a un serveur sur le port 21 de localhost  
Il y a un serveur sur le port 22 de localhost  
Il y a un serveur sur le port 23 de localhost  
Il y a un serveur sur le port 25 de localhost  
Il y a un serveur sur le port 37 de localhost  
...



Une **socket** décrit une **extrémité** de la connexion entre deux programmes en cours d'exécution sur le réseau

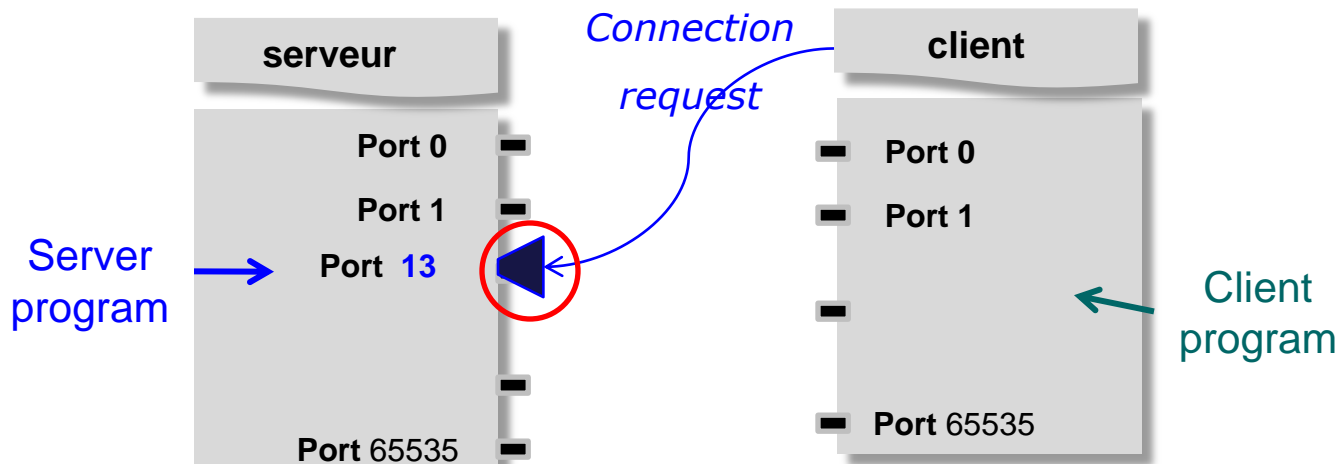


1

Le **serveur** attend une connexion en écoutant sur un port spécifique

2

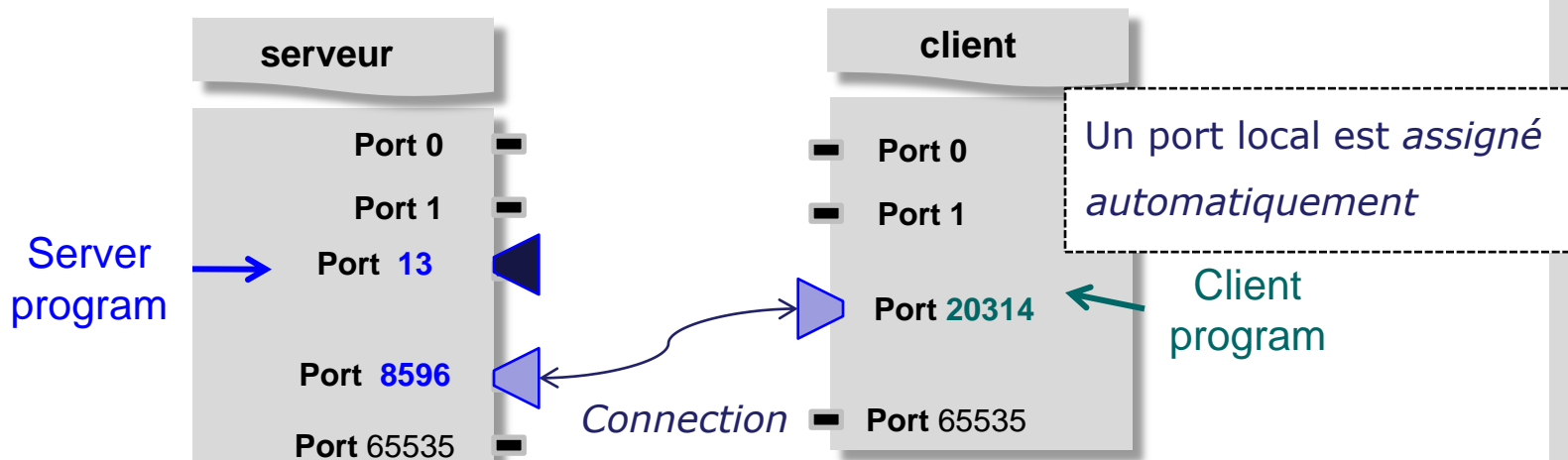
Le **client** émet une requête de connexion au serveur sur ce port



Le serveur accepte la connexion.

*Une session est établie.*

Le serveur utilise un seul port pour établir les connexion, mais réserve dynamiquement *un port temporaire* (local au serveur) à chaque client .

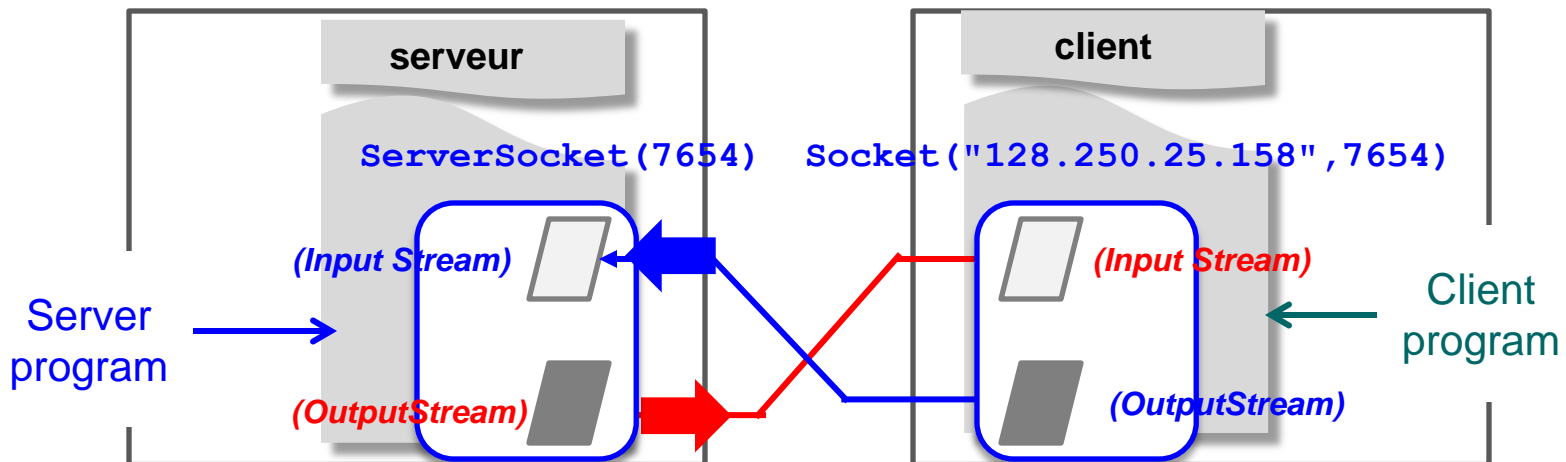


Le serveur accepte la connexion.

*2 sockets sont créées.*

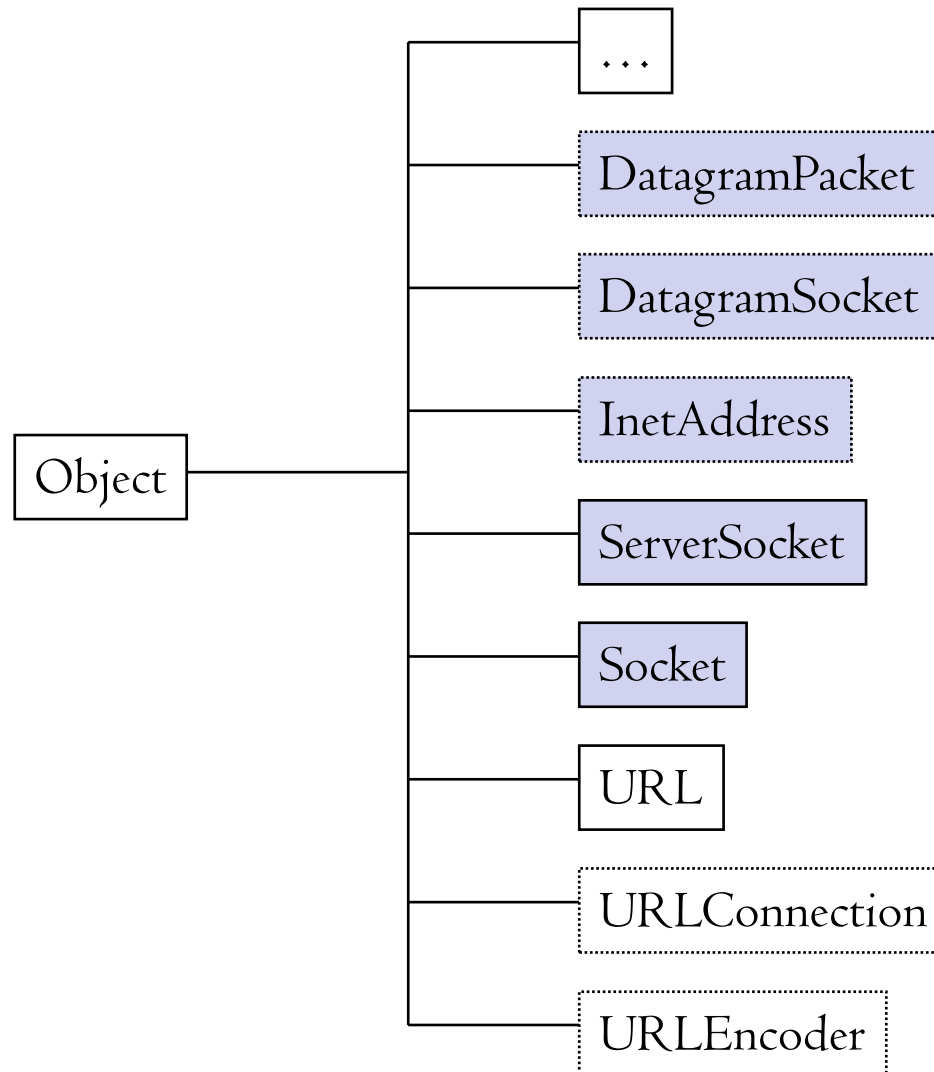
Les données peuvent être transmises.

Les flux d'entrée et de sortie sont connectés aux sockets



# Sockets TCP





**java.net.\***

## API

## java.net.Socket 1.0

- `Socket(String host, int port)`  
*constructs a socket to connect to the given host and port.*
- `InputStream getInputStream()`
- `OutputStream getOutputStream()`  
*get streams to read data from the socket and write data to the socket*

→ Récupérer la page d'index du serveur de l'ENSIAS

## Client HTTP

```
String g = "GET / HTTP/1.1\n" + "Host: www.ensias.ma\n\n";
```

**g** : pour récupérer la page d'index du serveur. Utilise le protocole HTTP

```
Socket socket = new Socket("www.ensias.ma", 80);
```

```
OutputStream out = socket.getOutputStream();
```

```
out.write(g.getBytes());
```



→ Traiter les données renvoyées par le serveur

## Client HTTP

```
InputStream in = socket.getInputStream();  
byte[] b = new byte[1000];  
//pour les données renvoyées  
int nbBitsRecus = in.read(b); //effectivement recues  
if(nbBitsRecus>0) {  
    System.out.println(nbBitsRecus + " bits recus.");  
    System.out.println("Recu: "+  
                        new String(b,0,nbBitsRecus));  
}
```

## → Résultat

```
1000 bits recus.  
Recu: HTTP/1.1 200 OK  
Date: Wed, 09 Oct 2013 21:40:05 GMT  
Server: Apache/2.2.3 (CentOS)  
X-Powered-By: PHP/5.1.6  
...  
<meta http-equiv="Content-Type"  
content="text/html;> charset=utf-8" />  
<meta content="text/html; Charset=UTF-8" http-  
equiv="Content-Type" />  
<title>Ecole Nationale SupÃ©rieure d&#039;Informatique ...
```

## 1. Créer une socket

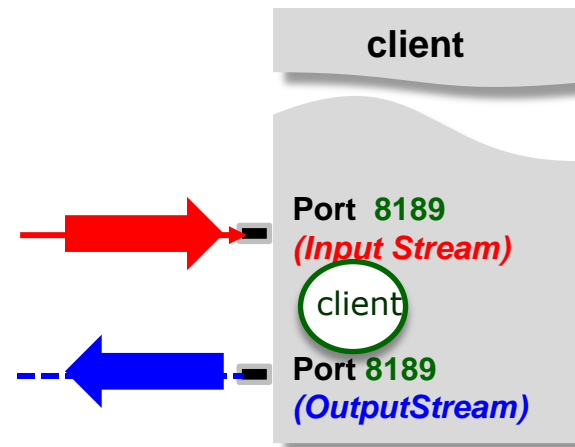
```
Socket client = new Socket(server, port_id);
```

## 2. Créer les flots d'E/S pour la communication

```
Scanner in = new Scanner(client.getInputStream());
```

```
PrintWriter out = new
```

```
PrintWriter(client.getOutputStream(), true);
```



## 3. Communiquer avec le serveur

Recevoir les données à partir du serveur :

```
String line = in.readLine();
```

Envoyer les données au serveur :

```
out.writeBytes("Hello\n");
```

## 4. Fermer la socket

```
client.close();
```

Le client peut fonctionner sur tout ordinateur dans le réseau (LAN, WAN ou Internet) tant qu'il n'y a pas de pare-feu qui bloque la communication.

## API

### java.net.ServerSocket 1.0

- `ServerSocket(int port)`  
*creates a server socket that monitors a port.*
- `Socket accept()`  
*Waits for a connection. This method blocks (that is, idles) the current thread until the connection is made. The method returns a `Socket` object through which the program can communicate with the connecting client.*
- `void close()`  
*closes the server socket*

## DateServer.java

```
public class DateServer {  
    public static void main(String[] args) throws IOException  
    {  
        ServerSocket listener = new ServerSocket(8189);  
        try {  
            while (true) {  
                Socket socket = listener.accept();  
                try {  
                    PrintWriter out =  
                        new PrintWriter(socket.getOutputStream(), true);  
                    out.println(new Date().toString());  
                } finally {  
                    socket.close();  
                }  
            }  
        }  
        finally { listener.close(); }  
    }  
}
```

Il est possible de tester le serveur avec telnet.

## En général : côté serveur

```
int port_d_ecoute = 8189;  
ServerSocket listener = new ServerSocket(port_d_ecoute);  
  
while(true)  
{  
    Socket socket_de_travail = listener.accept();  
    new ClasseDuTraitement(socket_travail);  
}
```

## 1. Créer une socket serveur

```
ServerSocket listener= new ServerSocket( 8189);
```

## 2. Attendre une connexion

```
Socket s = listener.accept();
```

## 3. Créer les flots d'E/S pour la communication

```
Scanner in = new Scanner(s.getInputStream());
```

```
Printwriter out=
```

```
new PrintWriter(s.getOutputStream(),true /*autoFlush*/);
```



## 4. Communiquer avec le client

Recevoir à partir du client:

```
String line = in.readLine();
```

Envoyer au client:

```
out.writeBytes("Hello\n");
```

## 5. Fermer la socket

```
client.close();
```

**Attention** : si on ne ferme pas la socket et un autre programme utilise le même port, il sera impossible de la fermer car le port sera occupé !

# Servir plusieurs clients



- Le **serveur** est **bloqué** jusqu'à ce qu'un client se connecte au serveur :

```
Socket s = listener.accept();
```

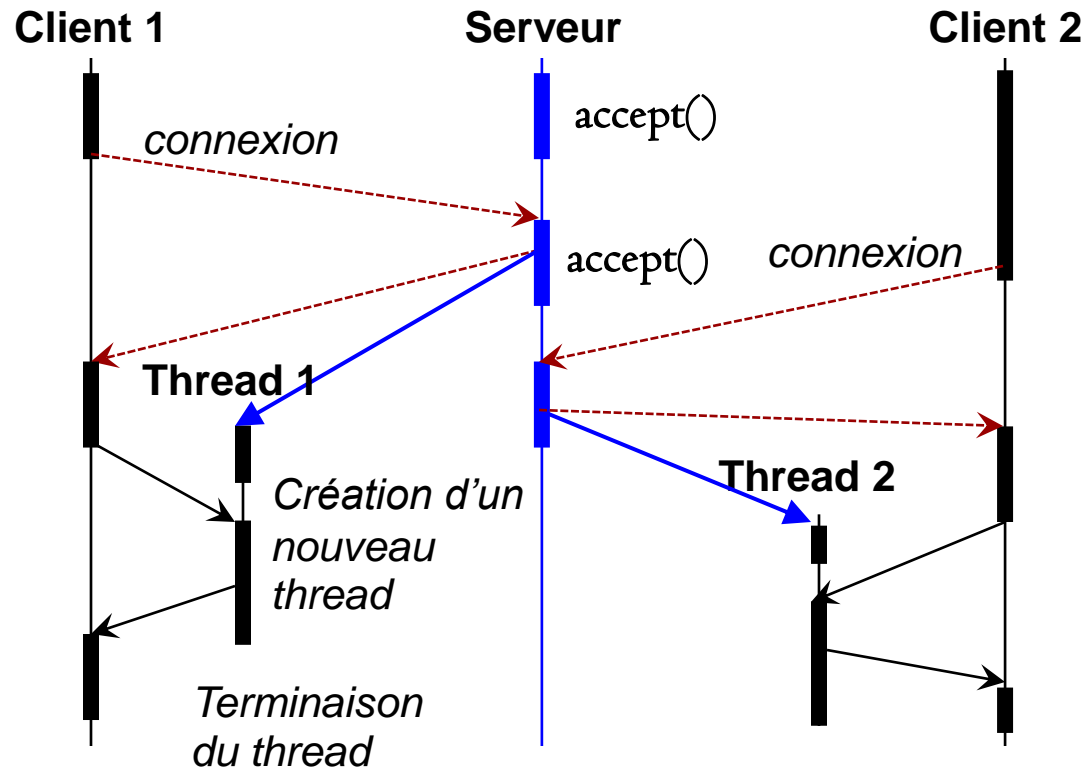
- Le **serveur** et le **client** seront **bloqués** si les données à partir de la socket ne sont pas disponibles :

```
String line = in.readLine();
```

- Il est difficile de servir plusieurs clients
  - Connexion avec le 1<sup>e</sup> client
  - Attente d'un 2<sup>e</sup> client ...  
et devient non répondant aux requêtes du 1<sup>e</sup>

- Placer l'appel à **accept()** dans une boucle
- Un nouveau thread pour traiter chaque connexion

```
public static void main(String[] args) {  
    ServerSocket listener = new ServerSocket(8189);  
    while(true){  
        Socket socket = listener.accept();  
        Runnable client= new ClientHandler(socket);  
        Thread thread = new Thread(client);  
        thread.start();  
    }  
}
```



```
class ClientHandler implements Runnable{
```

```
    private Socket socket;
```

```
    public ClientHandler(Socket socket) { ... }
```

```
    public void run() {
```

```
        try {
```

```
            InputStream inS = socket.getInputStream();
```

```
            OutputStream outS = socket.getOutputStream();
```

```
            Scanner in = new Scanner(inS);
```

```
            PrintWriter out = new PrintWriter(outS, true);
```

```
            echo(in, out); //echo sortie client par exemple
```

```
        } finally { socket.close(); }
```

```
    }
```

Le constructeur maintient  
une référence sur la socket

# Adressage



→ Chercher l'adresse IP de "www.ensias.ma"

## API

java.net.InetAddress 1.0

- static InetAddress **getByName**(String host)
- static InetAddress[] **getAllByName**(String host)  
*constructs an InetAddress, or an array of all Internet addresses, for the given host name.*

```
System.out.println("Hôte/IP : " +  
    InetAddress.getByName("www.ensias.ma")) ;
```

Hôte/IP : www.ensias.ma/196.200.135.4



→ Chercher l'adresse IP de "www.ensias.ma"

- ◆ Se fait automatiquement à l'aide d'un serveur DNS
- ◆ Ne signifie pas que l'IP est accessible

→ Savoir si un nom d'hôte est accessible

```
boolean isReachable(int timeout)
```

## → Récupérer son adresse IP

**API**

**java.net.InetAddress 1.0**

- `static InetAddress getLocalHost ()`  
*constructs an InetAddress for the local host.*

```
import java.net.InetAddress; ...

public class TestInetAddress {

    public static void main(String[] args) throws
                                UnknownHostException {

        System.out.println("Hôte/IP : " +
                                InetAddress.getLocalHost());

    }
}
```

Hôte/IP : Inspiron-DELL/192.168.2.53

→ Mais aussi

## API

### java.net.InetAddress 1.0

- `byte[] getAddress()`  
*returns an array of bytes that contains the numerical address.*
- `String getHostAddress()`  
*returns a String with decimal numbers, separated by periods, for example, "132.163.4.102"*
- `String getHostName()`  
*returns a the host name.*

- Chaque interface (carte réseau) peut disposer de plusieurs adresses IP
- adresse IP local (127.0.0.1) ➔ localhost
  - adresse IP réseau (192.168.x.x réseaux maisons)
  - adresse IP internet

API

java.net. **NetworkInterface** 1.0

- Enumeration<NetworkInterface> **getNetworkInterfaces()**  
*returns all the network interfaces.*

```
import java.net.NetworkInterface;...

public class TestNetworkInterface {

    public static void main(String[] args) throws Exception {

        Enumeration<NetworkInterface> lesInterfaces=

            NetworkInterface.getNetworkInterfaces();

        while (lesInterfaces.hasMoreElements()) {

            NetworkInterface interface = lesInterfaces.nextElement();

            Enumeration<InetAddress> lesAddresses =


                interface.getInetAddresses();

            while (lesAddresses.hasMoreElements()) {

                InetAddress address = lesAddresses.nextElement();

                System.out.println(address.getHostAddress());

            } } } }
```



## M.3.3.2 Programmation Objet Avancée