systèmes distribués

Programmation réseau en Java : Socket

Master 1 RID

Plan

- **Socket**
  - Implémentation de Sun pour Java
- **Communication par socket**
  - Connecté : TCP
  - Non connecté : UDP
- **Exemple de communication par socket**
  - Echange de messages (String, int, …)
- **Lecture et écriture de plus haut niveau**
  - Lecture ligne par ligne de texte d'un socket
  - Ecriture de texte dans un socket
  - Lecture de données Java à partir d'un socket
  - Ecriture de données Java dans un socket
  - Transmission de données en passant par une chaîne de caractères
- **Transmission d'objet par les sockets**

Socket réseaux

- Socket réseaux est un modèle permettant **la communication et la synchronisation** interprocessus afin de permettre à divers processus de communiquer aussi bien sur une même machine qu'à travers un réseau.

- Les sockets sont mis au point par l'université de Berkeley (University of California) en 1986.
  - L'interface originelle de Berkeley était pour le langage C, mais depuis les sockets ont été implémentées dans de nombreux langages.

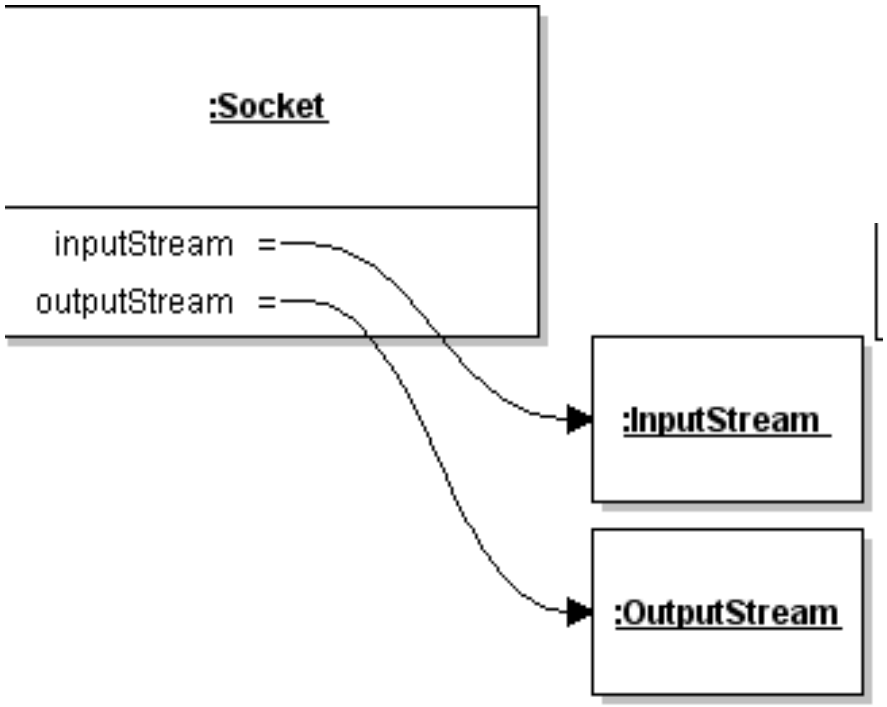
- Package **java.net** à partir de **J2SE 1.4 (6 février 2002).**

Socket

- **Deux modes de communication :**
  - **Mode connecté (comparable à une communication téléphonique),** utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l'adresse de destination n'est pas nécessaire à chaque envoi de données.
  - **Mode non connecté (analogue à une communication par courrier),** utilisant le protocole UDP. Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.
- Les sockets utilisent directement les services de la couche transport du modèle OSI (protocoles UDP ou TCP), elle-même utilisant les services de la couche réseau (protocole IP).
  - Les sockets permettent à deux processus de communiquer entre eux à travers d'une liaison identifiée par **une adresse IP et un numéro de port.**

Implémentation Socket de Sun pour Java

- La communication nécessite 2 sockets : 1 pour chacun des 2 programmes communicants via le réseau.
  - **1 socket pour le client.**
  - **1 socket pour le serveur.**
- En Java, chaque instance de la classe Socket est associé à un objet de la classe **InputStream** (pour **lire** sur le socket) et à un objet de la classe **OutputStream** (pour **écrire** sur le socket).

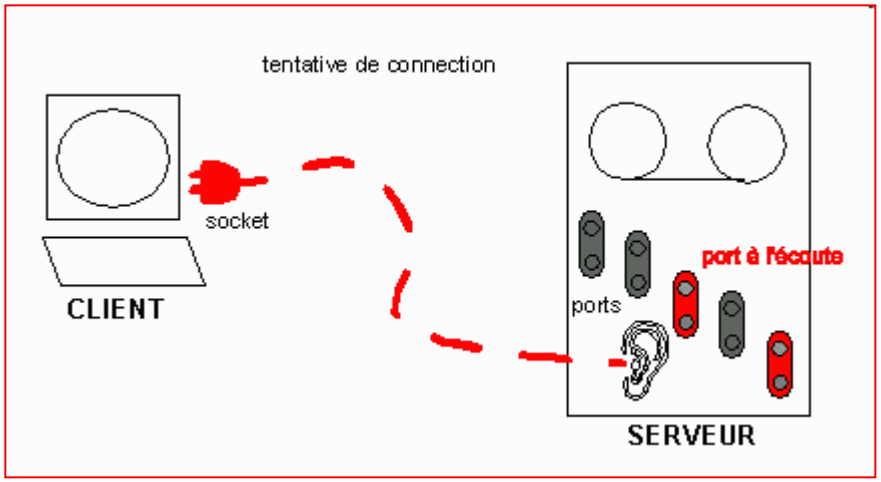


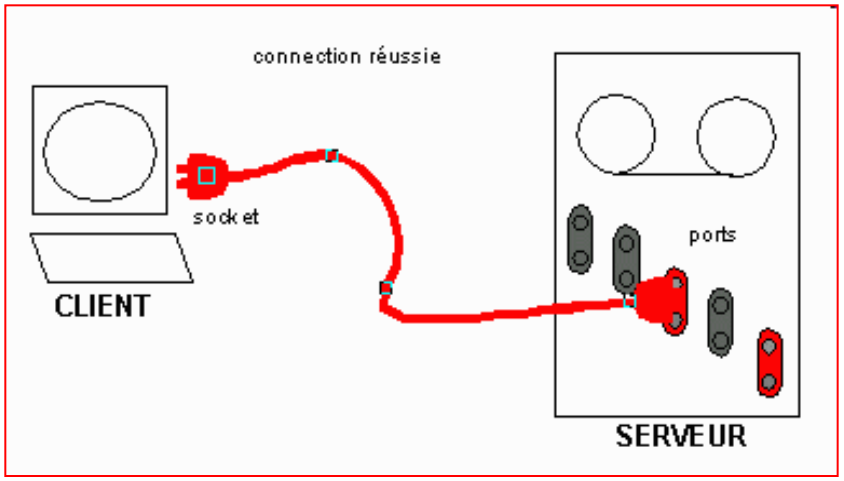
Côté du serveur

- Le serveur utilisera deux types de sockets.
  - **Le premier, appelé socket de connexion sert à attendre un client.**
    - En Java, créer un socket de connexion peut se faire simplement en instanciant un objet de la classe **ServerSocket** du package **java.net**.
    - **ServerSocket conn = new ServerSocket(10080);**
  - **Le second, appelé socket de communication sert à dialoguer avec le client.**
    - Une fois le socket de connexion créé, il suffit de lui demander d'attendre un client et de récupérer le socket de communication qui permettra de dialoguer avec le client.
    - **Socket comm = conn.accept();**
    - On peut ensuite communiquer avec le client en utilisant les flux d'entrée et de sortie associés au socket.

Class ServerSocket

- Implémente les sockets de connexion du côté du serveur.
- **ServerSocket (numeroPort) :** crée un objet ServerSocket sur ce numéro de port.
- **accept() :** attend une connexion d'une machine cliente, à l'arrivée d'une demande de connexion d'une machine cliente, un socket est crée pour connecter ce client au serveur. C'est l'objet retourné par la méthode bloquante accept().
- Avant et après la méthode **accept()** :





Quelques méthodes

- **accept() :** attend une connexion d'une machine cliente.
- **close() :** ferme le ServerSocket et toutes les sockets en cours obtenus par sa méthode accept.
- **isClosed() :** indique si le socket est fermé.
- **ServerSocket (numeroPort, int backlog) :** crée un objet ServerSocket sur ce numéro de port avec une queue d'attente de connexion de taille spécifiée par l'entier backlog.
  - Par défaut, la taille est 50.
  - Les demandes de connections, quand la queue est pleine, sont rejetées et provoque une exception du coté du client.
- **getLocalPort() :** retourne le numéro de port local.
- **InetAddress getInetAddress() :** retourne l'adresse du serveur.

Côté du client

- Contrairement au serveur, le client n'utilise qu'un seul socket : **le socket de communication.**

- Connexion au serveur et obtention d'un socket de communication.
  - **Socket comm = new Socket ("localhost", 10080).**

- On peut ensuite communiquer avec le serveur en utilisant les flux d'entrée et de sortie associés au socket.

Class Socket

- **9 constructeurs dont 2 « Deprecated », donc 7 constructeurs parmi eux :**
  - Socket (String host, int port)
  - Socket (InetAddress address, int port)
    - Utilise l'adresse IP : InetAdress
- **Quelques méthodes :**
  - close() : ferme proprement le socket est susceptible de lever une exception UnknownHostException.
  - isClosed() : indique si le socket est fermé.
  - isConnected() : retourne l'état de la connexion du socket.
  - getOutputStream() : retourne un flux de sortie OutputStream pour le socket.
  - getInputStream() : retourne un flux d'entrée InputStream pour le socket.
  - getPort() : retourne le numéro de port distant auquel le socket est connecté.
  - InetAddress getInetAddress() : retourne l'adresse de la machine distante.
    - **InetAddress** représente une adresse IP et éventuellement le nom de la machine.
      - **Quelques méthodes :**
        - getHostAdress() : retourne l'IP sous forme de String.
        - getHostName() : retourne le nom d'hôte.

Exemple d'utilisation de InetAddress

- **Objectif :** trouver l'adresse IP correspondant à un nom de machine passé comme argument sur la ligne de commande.

```
import java.net.* ;
public class ResoudreNom{
 public static void main(String[] args){
InetAddress adresse ;
try
{ adresse=InetAddress.getByName(args[0]);
System.out.println(adresse.getHostAddress()) ;
}catch(UnknownHostException e) { }
}
}
```

Exécution :

java ResoudreNom localhost

127.0.0.1

java ResoudreNom www.facebook.com

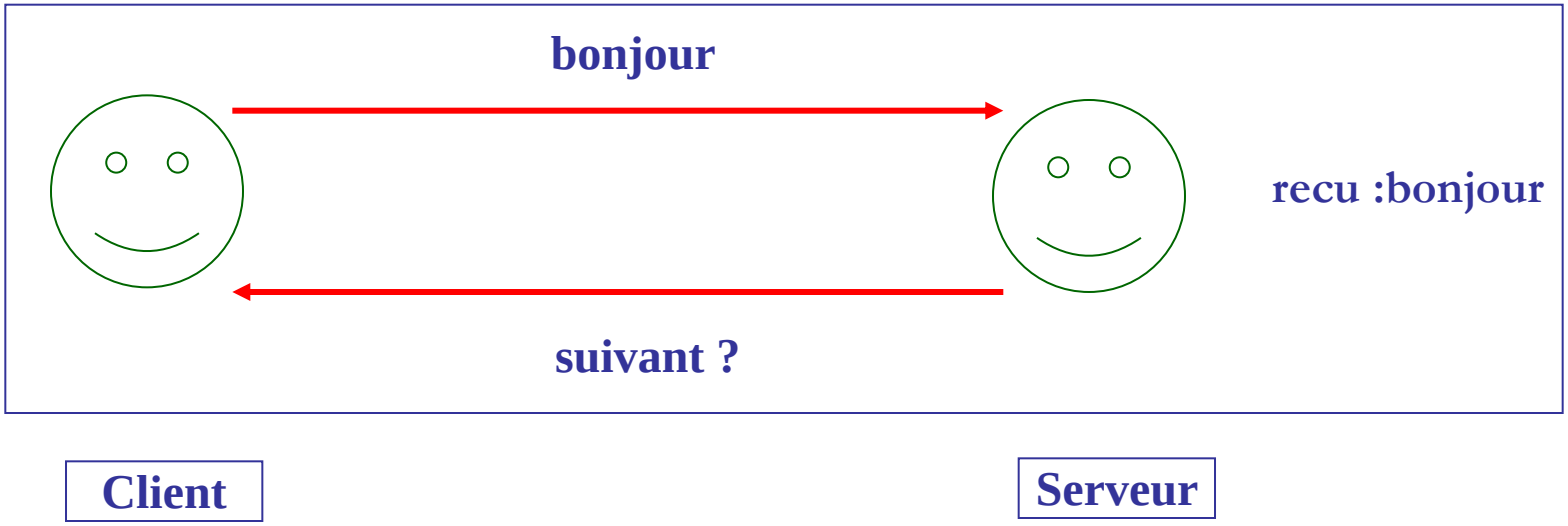
31.13.80.81

- Deux méthodes de la classe InetAddress sont illustrées dans cet exemple :
  - **getByName() :** permet de résoudre l'adresse IP d'une machine étant donné son nom.
  - **getHostAddress() :** permet de retourner sous forme de chaîne de caractères l'adresse IP mémorisé dans l'objet de type InetAddress.

Exemple de communication par socket

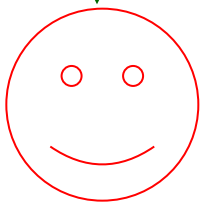
- **Une boucle :**
  - Le client envoie un message au serveur.
  - Le serveur confirme la réception et demande le mot suivant.

  - Pour se déconnecter, le client envoie le caractère « q ».



Exemple de communication par socket

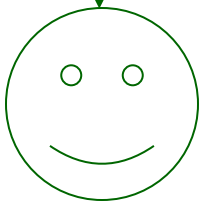
sockOut = new PrintWriter(sock.getOutputStream(), true) pour émission
sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream())) pour réception



Client

sockOut = new PrintWriter(sock.getOutputStream(), true) pour émission
sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream())) pour réception

Serveur



sockOut.println(message)

recu = sockIn.readLine()

System.out.println("recu :"+recu)

sockOut.println("suivant ? ")

String recu = sockIn.readLine()

System.out.println("serveur -> client :" + recu)

Serveur

```
import java.net.*;
import java.io.*;
class ServerEcho {
   public static void main(String args[]) {
   ServerSocket server = null;
   try {
    server = new ServerSocket(7777); // socket de connexion
    while (true) {
     Socket sock = server.accept(); // en attente
     System.out.println("connecte");
     PrintWriter sockOut = new PrintWriter(sock.getOutputStream(), true); //forcer l'écriture
     BufferedReader sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
     String recu;
     while ((recu = sockIn.readLine()) != null) {
      System.out.println("recu :"+recu);
      sockOut.println("suivant ? ");
     }
     sockOut.close();
     sock.close();
    } //fin while (true)
       } catch (IOException e) {
       try {server.close();}
       catch (IOException e2) {}
} // fin premier catch
}// fin main
 } // fin classe
```

Lire le message du client

Afficher le message du client

Envoyer le message "suivant ? " au client

Pour les sockets :

Réception : synchrone

Emission : asynchrone

Client

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class ClientEcho {
  public static void main(String[] args) throws IOException {
    Socket sock = null;
    PrintWriter sockOut = null;
    BufferedReader sockIn = null;
    try {sock = new Socket("localhost", 7777); // le socket de communication.
      sockOut = new PrintWriter(sock.getOutputStream(), true); //forcer l'écriture
      sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
    } catch (UnknownHostException e) {
      System.err.println("host non atteignable : localhost");
      System.exit(1);
    } catch (IOException e) {
      System.err.println("connection impossible avec : localhost");
      System.exit(1);
    }
    System.out.println("tapez q pour terminer");
    Scanner scan = new Scanner(System.in);
    String message = scan.next().toLowerCase();
    while (! message.equals("q")) {
      sockOut.println(message);
      String recu = sockIn.readLine();
      System.out.println("serveur -> client :" + recu);
      message = scan.next().toLowerCase();
    } sockOut.close();
    sockIn.close();
    sock.close();
  }
}
```

Envoyer le message au serveur

Lecture de la réponse du serveur

Afficher la réponse du serveur

PS : l'émission sans contraintes (asynchrone), la réception (synchrone) : il faut attendre la réponse du serveur (dans ce cas).

Exemple de communication par socket

```
java ServerEcho
connecte
recu :bonjour
recu :bye
```

```
java ClientEcho
tapez q pour terminer
bonjour
serveur -> client :suivant ?
bye
serveur -> client :suivant ?
q
```

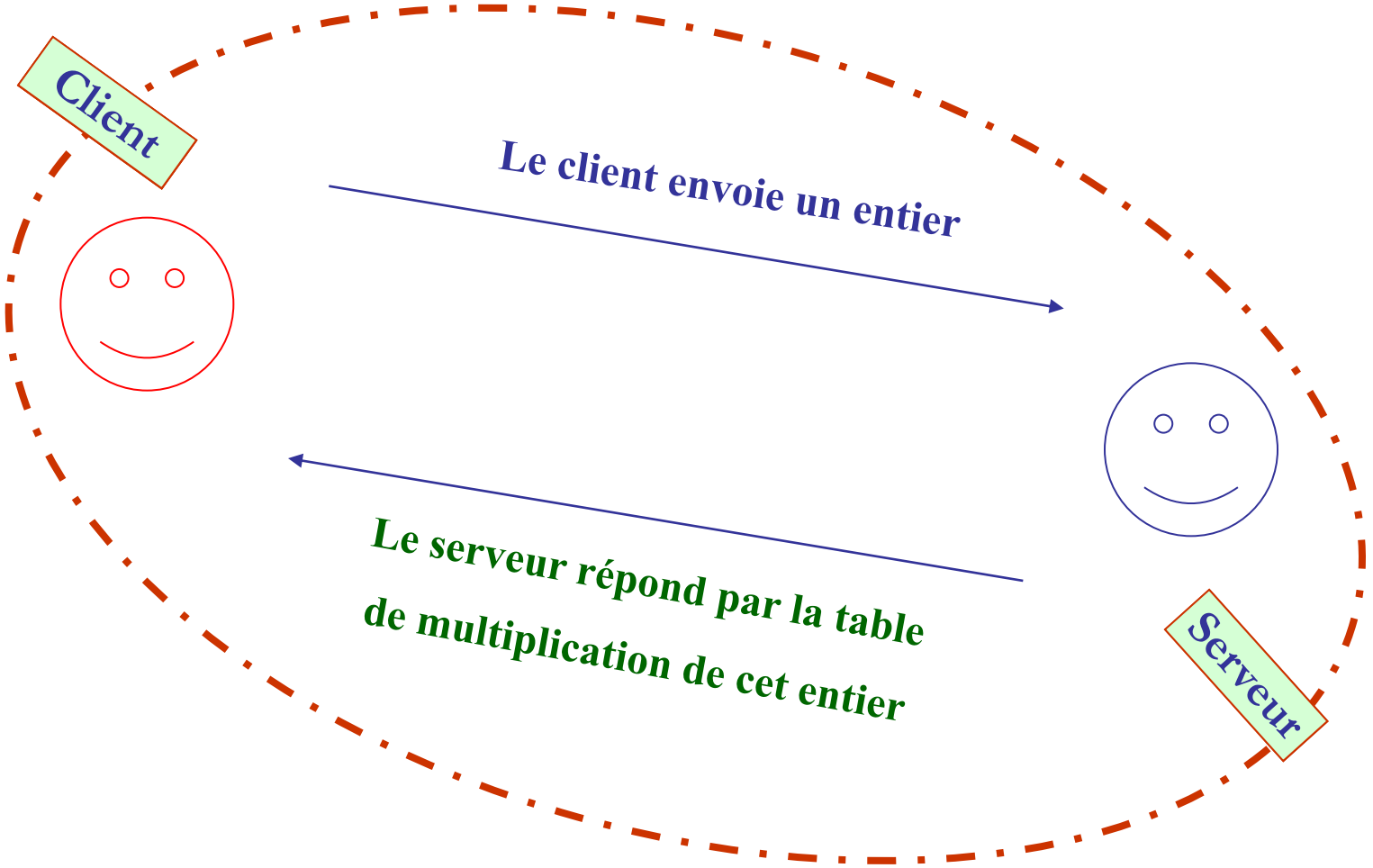
Exemple de communication par socket

- Le programme crée un serveur de socket sur son port 7777.

- Puis l'appel à accept() met le serveur en attente d'une connexion d'un client.
  - A la réception d'une connexion d'un client, un socket est crée pour assurer la communication.

- Lorsque le client a fini, le programme ferme le socket avec ce client et se remet en boucle d'attente d'une connexion d'un client.

Autre exemple



Exécution

- **java ServerEcho**

  connecte

  recu :5

  recu :6

- **java ClientEcho**

  tapez q pour terminer

  5

  Table de multiplication de 5

  0 5 10 15 20 25 30 35 40 45 50

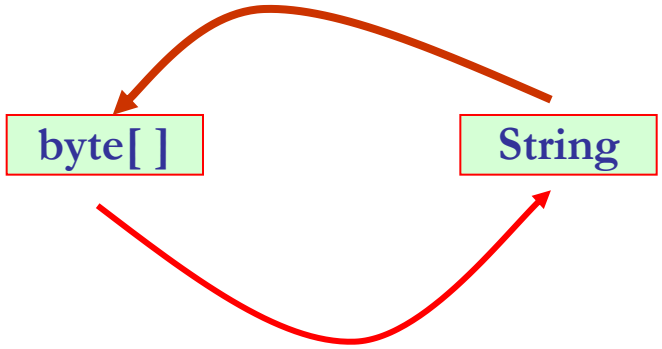
  6

  Table de multiplication de 6

  0 6 12 18 24 30 36 42 48 54 60

Caractère transmis et caractère Java (byte et char)

- Pour les caractères, le langage JAVA utilise le type « **char** » basé sur l'Unicode codé sur 16 bits. **La transmission sur les réseaux est basé sur l'octet : 8 bits.**
- Les méthodes de télécommunication de Java sont donc basées sur le transfert d'octets (byte).
  - **Convertir un String en byte[ ] :**
    - String chaine = "Ceci est une chaine";
    - **byte[ ] message=chaine.getBytes();**
  - **Convertir un byte[ ] en String :**
    - **chaine = new String(message);**



Lecture et écriture de plus haut niveau

- **Lecture ligne par ligne de texte d'un socket :**

```
try{
BufferedReader sockReader = new BufferedReader(new InputStreamReader(socket.getInputStream()))
String ligne;
while ((ligne = sockReader.readLine()) != null)     System.out.println(ligne);
      } catch (IOException e) {}
```

- On crée un BufferedReader attaché au flux InputStream.
- On peut lire alors directement des chaînes de caractères par readLine().
- readLine() renvoie la ligne lue sans le/les caractères de fin de ligne et null si la fin de flot est atteinte.

Lecture et écriture de plus haut niveau

- **Ecriture de texte dans un socket :**

```
PrintWriter sockWriter = new PrintWriter(socket.getOutputStream()));
sockWriter.print("chaine String");
sockWriter.print(1234);
sockWriter.print(12.34);
sockWriter.println(true);
```

- On crée un PrintWriter attaché au flux OutputStream.
- On peut lire alors directement n'importe quel type java par print ou println.
- Toutes ces méthodes ne génèrent pas d'Exception, elles sont pratiques mais insécurisé.

Lecture et écriture de plus haut niveau

- **Lecture de données Java à partir d'un socket :**

- **DataInputStream** permet de lire tous les types primitifs java sur tous les systèmes.

```
DataInputStream sockDataIn = null;
try {
  sockDataIn = new DataInputStream(socket.getInputStream());
  byte by = sockDataIn.readByte(); // retourne le prochain octet.
  char c = sockDataIn.readChar(); // retourne le prochain caractère.
  boolean bo = sockDataIn.readBoolean(); // lit un octet et retourne false si l'octet est == 0.
  int i = sockDataIn.readInt(); // lit 4 octets et retourne l'entier correspondant.
  double d = sockDataIn.readDouble(); // lit 8 octets et retourne le double correspondant.
  String s = sockDataIn.readUTF(); // décode les octets en caractère et retourne le String ainsi formé.
} catch (EOFException e) {
} catch (IOException e) {
}
```

- Toutes les méthodes ci-dessus lèvent l'exception EOFException si la fin de flot est atteinte.

Lecture et écriture de plus haut niveau

- **Ecriture de données Java dans un socket :**

- **DataOutputStream** permet d'écrire tous les types primitifs java sur tous les systèmes.

```
DataOutputStream sockDataOut = null;
try {
sockDataOut = new DataOutputStream(socket.getOutputStream());
sockDataOut.writeByte(6); //écrit le premier octet de l'entier.
sockDataOut.writeChar('c'); //écrit le caractère correspondant aux 2 premiers octets de l'entier.
  sockDataOut.writeBoolean(true); //écrit le booléen sur un octet.
  sockDataOut.writeInt(-1234); // écrit l'entier sur 4 octets.
  sockDataOut.writeDouble(12.34); // écrit le double sur 8 octets.
  sockDataOut.flush(); // devrait forcer l'écriture du contenu du tampon sur le flot, s'il y a un tampon.
  sockDataOut.writeUTF("chaine String"); // écrit le String en l'encodant en UTF-8 modifié.
} catch (IOException e) {
}
```

Lecture et écriture de plus haut niveau

- **Transmission de données en passant par une chaîne de caractères :**

```
try {
  OutputStream sockOut = socket.getOutputStream();
  byte[] buffer = new byte[1024];
  String envoi = "chaine String"+" "+(-1234)+" "+12.34+" "+true;
  buffer = envoi.getBytes();
  sockOut.write(buffer, 0, envoi.length());
  sockOut.flush();
} catch (IOException e) {}
```

Transmission d'objet par les sockets

- **Exemple :** un client qui se connecte sur un serveur pour connaître les caractéristiques d'un objet (exemple : un chien).
  - Un tableau de chien est stocké coté serveur.
  - Un client se connecte sur le serveur et souhaiterais connaître les caractéristique d'un chien.
    - Il doit envoyer au serveur le nom du chien.
  - Le serveur lit le nom, confirme la réception, parcours le tableau pour trouver le chien et enfin sauvegarde **(enregistre)** l'objet chien.
  - Cet objet est envoyé au client en utilisant les sockets.
  - L'objet est lu par le client **(restaurer)**.
  - Le client affiche les caractéristique de l'objet.

- L'objet transmis via le réseau doit être **sérialisable.**
  - **implements Serializable.**

Transmission d'objet par les sockets

```
import java.io.Serializable;
public class Chien implements Serializable{
  String nom;
  String aboiement;
  int nombrePuce;
  Chien (String s, String a, int i) {
    nom = s;
    aboiement = a;
    nombrePuce = i;
  }
    String getNom() {
    return nom;
  }
  public String toString() {
    return "Le chien est : "+nom+" aboie "+aboiement+" et a "+nombrePuce+" puces.";
  }
}
```

nom

aboiement

nombrePuce

Les attributs

getNom()

toString()

Les méthodes

La classe Chien est partagée entre le client et le serveur

```
import java.io.*; import java.net.*;
public class ClientChien {
  public static void main(String[] args) throws IOException {
    if (args.length != 2) {System.out.println("Usage : java ClientChien hostName nom_chien" );System.exit(1);}
    String hostName = args[0];  String Nomchien = args[1];
    Socket sock = null;
    PrintWriter sockOut = null;
    ObjectInputStream sockIn = null;
    try {
      sock = new Socket(hostName, 7777);
      sockOut = new PrintWriter(sock.getOutputStream(), true);
      sockIn = new ObjectInputStream(sock.getInputStream());
    } catch (UnknownHostException e) {System.err.println("host non atteignable : "+hostName);  System.exit(1); }
       catch (IOException e) {System.err.println("connection impossible avec : "+hostName); System.exit(1);}
    sockOut.println(Nomchien);
    try {
     Object recu = sockIn.readObject();
      if (recu == null) System.out.println("erreur de connection");
      else { Chien chien = (Chien)recu;
       System.out.println("serveur -> client : " + chien);
      }
    } catch (ClassNotFoundException e) {System.err.println("Classe inconnue : "+hostName); System.exit(1);}
     sockOut.close();
    sockIn.close();
    sock.close();
  }
}
```

Coté client

Envoyer le nom du chien au serveur

Restaurer l'objet chien reçu du serveur

Afficher les caractéristiques du chien

Coté serveur

```
import java.net.*;
import java.io.*;
class ServerChien {
  public static void main(String args[]) {
    Chien[ ] tabChien = {
        new Chien ("medor", "wouf", 3),
        new Chien ("milou", "wouah", 0),
        new Chien ("cerbere", "grrr", 12)};
    ServerSocket server = null;
    try {
      server = new ServerSocket(7777);
      while (true) {
        Socket sock = server.accept();
        System.out.println("connecte");
        ObjectOutputStream sockOut = new  ObjectOutputStream(sock.getOutputStream());
        BufferedReader sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
        String recu;  while ((recu = sockIn.readLine()) != null) {
        System.out.println("recu :"+recu);
        String nom = recu.trim();
        for (int i=0; i<tabChien.length; i++)
          if (tabChien[i].getNom().equals(nom)) {sockOut.writeObject(tabChien[i]);
            break;
          }
        }
        sockOut.close();
        sock.close();
      }
    } catch (IOException e) {
      try {server.close();} catch (IOException e2) {}
    }
  }
}
```

Tableaux de chiens

Lire le nom envoyé par le client

Confirmer la réception

Parcourir le tableau pour trouver le chien

Enregistrer l'objet Chien afin de l'envoyer au client

Transmission d'objet par réseau

- **java ServerChien**

  connecte
  recu :milou
  connecte
  recu :medor
  connecte
  recu :cerbere

- **java ClientChien localhost milou**

  serveur -> client : Le chien est : milou aboie wouah et a 0 puces.

- **java ClientChien localhost medor**

  serveur -> client : Le chien est : medor aboie wouf et a 3 puces.

- **java ClientChien localhost cerbere**

  serveur -> client : Le chien est : cerbere aboie grrr et a 12 puces.

Autre exemple