Java avancé et GL
Cours 3- Programmation réseau

Extrait du cours de « S.Vialle »

Supélec

Programmation par “sockets-Java”

1 - Principes des sockets en Java
- Principes de base
- Principales classes Java

2 - Programmation avec des sockets en Java
- Sockets UDP en Java
- Sockets TCP en Java

3 – Flux de données de haut niveau en TCP

1 - Principes des sockets en Java

- Principes de base
- Principales classes Java

Principes de base

Permettent l'échange de messages entre 2 processus, entre 2 machines

1- Chaque machine crée une socket,

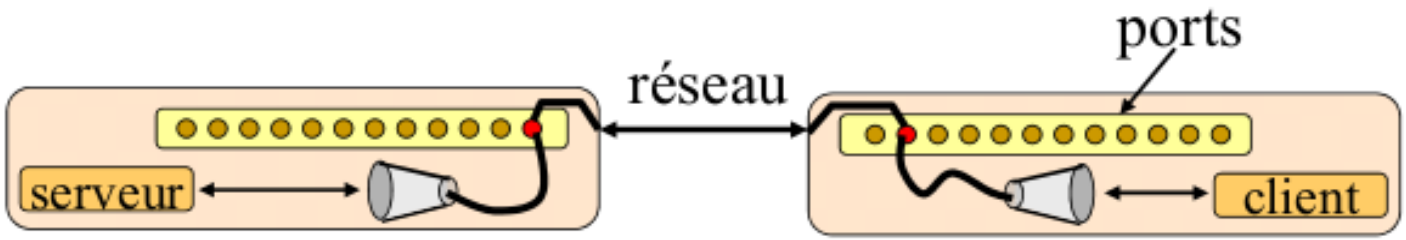
2- Chaque socket sera associée à un port de sa machine hôte,

3- Les deux sockets seront explicitement connectées si on utilise un protocole en mode connecté …,

4- Chaque machine lit et/ou écrit dans sa socket,

5- Les données vont d'une socket à une autre à travers le réseau,

6- Une fois terminé chaque machine ferme sa socket.



Principes de base

- **Les sockets sont des objets** (bien sur!)
- **Les sockets TCP doivent être associées à des « flux » (in et out)**
  → facilités de lecture et d'écriture par la suite

- **Classes de sockets :**

UDP

`DatagramSocket` →
- `DatagramSocket()` : client
- `DatagramSocket(ServerPort)` : serveur

TCP

`Socket` → `Socket(ServerName,ServerPort)` : client

`ServerSocket` → `ServerSocket(ServerPort)` : serveur

1 - Principes des sockets en Java

- Principes de base
- Principales classes Java

Syntaxe Java

Sockets UDP en Java (une démarche possible) :

class DatagramSocket : (coté client et coté serveur)

- Constructeur :
  - DatagramSocket() : creation d'une socket UDP, libre
  - DatagramSocket(int port) : creation d'une socket UDP, liée à un port
- Méthodes :
  - close() : ferme la socket.
  - receive(DatagramPacket p) : reçoit un « DatagramPacket » de cette socket.
  - send(DatagramPacket p): envoit un « DatagramPacket » sur cette socket.

class DatagramPacket :

- Constructeur :
  - DatagramPacket(byte[] buf, int length, InetAddress address, int port) : creation d'un packet à destination d'une machine et d'un port spécifiés

Programmation par « sockets »

Syntaxe Java

Sockets TCP en Java :

class Socket : (coté client, et coté serveur en partie)

- **Constructeur :**
  Socket(String host, int port) : creation d'une socket TCP connectée sur le port et la machine hôte spécifiés.
- **Méthodes :**
  close() : ferme la socket.
  OutputStream getOutputStream() :
  revoie un flux de sortie pour cette socket.
  IntputStream getIntputStream() :
  revoie un flux d'entrée pour cette socket.

connect implicite

class OutputStream :

- **Méthodes :**
  write(…) : écrit dans le flux.
  close() : ferme flux.

class InputStream :

- **Méthodes :**
  read(…) : lit le flux.
  close() : ferme le flux

Syntaxe Java

Sockets TCP en Java :

class ServerSocket : (coté serveur)

- **Constructeur :**
  ServerSocket(int port) : creation d'une socket TCP connectée sur le port spécifié de la machine hôte
- **Méthodes :**
  close() : ferme la socket.

  Socket accept() :
  écoute la socket et attend une requête de connection, retourne une nouvelle socket sur laquelle écouter le nouveau client (et lui seul)

bind implicite

listen implicite

Syntaxe Java

Sockets en Java : résolution de nom et d'adresse

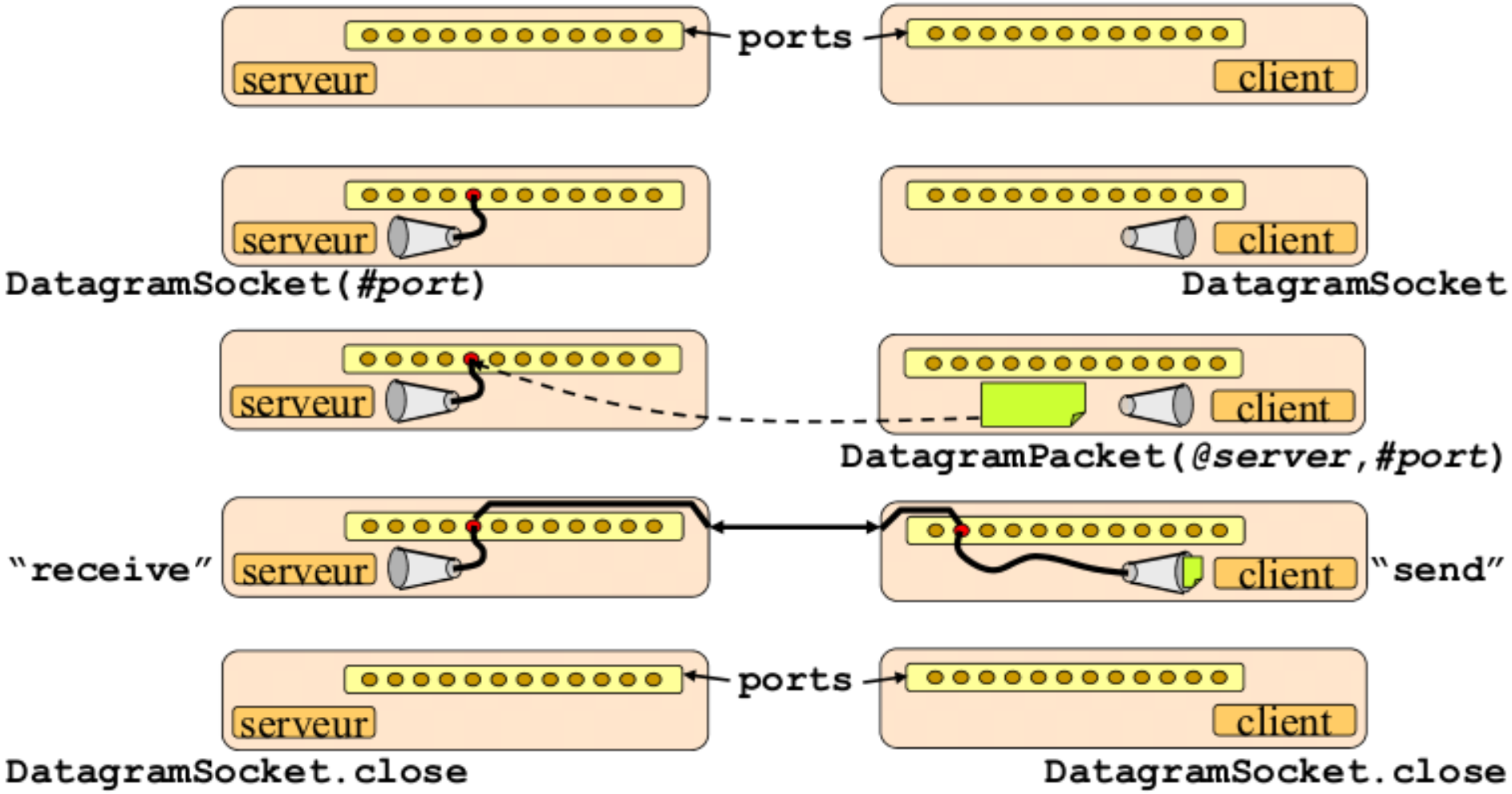
```
class InetAddress :

• Méthodes statiques : (appelable à tout moment)
   static InetAddress getLocalHost() :
          Retourne l'adresse IP de l'hote (local)
   static InetAddress getByName(String host)     :
   static InetAddress getAllByName(String host) :
          Retourne l'adresse ou un tableau d'adresse IP de la
          machine de nom spécifié

• Autres méthodes (applicables aux objets InetAddress)
   String getHostAddress() :
          Retourne l'adresse IP en format « String »
   String getHostName()     :
          Retourne le nom de la machine correspondant à
          l'adresse IP
```

2 - Les sockets en Java

- Sockets UDP en Java
- Sockets TCP en Java

Les sockets UDP en Java

Etapes d'une communication client-serveur en UDP avec Java :



Sockets UDP en Java

Code client UDP :

```
static int ServerPort = 0;                    // Server identification
static String ServerName = null;
static int NbStep = 10;                       // Features of msg to send
static int MsgSize = 5;

public static void main(String [] argv)
       throws IOException, SocketException, UnknownHostException {
 DatagramSocket socket = null;                // Socket of the client
 InetAddress ServerIpAdr = null;              // Ip Adr of the server
 DatagramPacket packet = null;                // Datagram to send
 byte msg[];                                  // Msg to send
 int s, i;                                    // Msg and byte counters

 // Command line parsing and init
 …
 ServerName = new String();
 ServerName = argv[0];
 ServerPort = (Integer.valueOf(argv[1])).intValue();

 // Socket creation (UDP), unbounded
 try {
   socket = new DatagramSocket();
   System.out.println("Socket created on client");
 } catch (SocketException e) {
   System.err.println("Failed to create the socket!");
   System.exit(1);
 }
```

Sockets UDP en Java

Code client UDP (fin) :

```
try {  // Get IP address of the server from its name
   ServerIpAdr = InetAddress.getByName(ServerName);
} catch (UnknownHostException e) {
   System.err.println("Address of server can not be found!");
   System.exit(1);
}
// Socket usage: send data in the socket
msg = new byte[MsgSize];
packet = new DatagramPacket(msg,MsgSize,ServerIpAdr,ServerPort);
for (s = 0; s < NbStep; s++) {                    // Dialog
   msg = …;                                       // - fill msg
   if (s == NbStep-1)                             // - signal end of dialog
     msg[0] = (byte)255;                          //   to the server
   try {
     socket.send(packet);                         // - send msg
     System.out.print("Datagram " + s + " : [");
     for (i = 0; i < MsgSize-1; i++)
       System.out.print(msg[i]+",");
     System.out.print(msg[MsgSize-1]);
     System.out.println("] sent");
   } catch (IOException e) {                      // - react on pb…
     System.err.println("Send datagram has failed!");
     System.exit(1);
   }
}
// Close the socket (no exception , no return value!)
socket.close();
System.out.println("Socket closed on client");
}
```

Sockets UDP en Java

Code serveur UDP :

```
static int ServerPort = 0; // Server identification
static int NbStep = 10;    // Features of msg to receive
static int MsgSize = 5;

public static void main(String [] argv)
       throws IOException, SocketException {
  DatagramSocket socket = null;            // Socket of the server
  DatagramPacket packet = null;            // Datagram to send
  byte msg[];                              // Msg to send
  int s, i;                                // Msg and byte counter

  // Command line parsing and init
  …
  ServerPort = (Integer.valueOf(argv[0])).intValue();

  // Socket creation (UDP), bounded to the target port
  try {
    socket = new DatagramSocket(ServerPort);
    System.out.println("Socket created and bound on server");
  } catch (SocketException e) {
    System.err.println("Failed to create the socket!");
    System.exit(1);
  }
```

Sockets UDP en Java

Code serveur UDP (fin) :

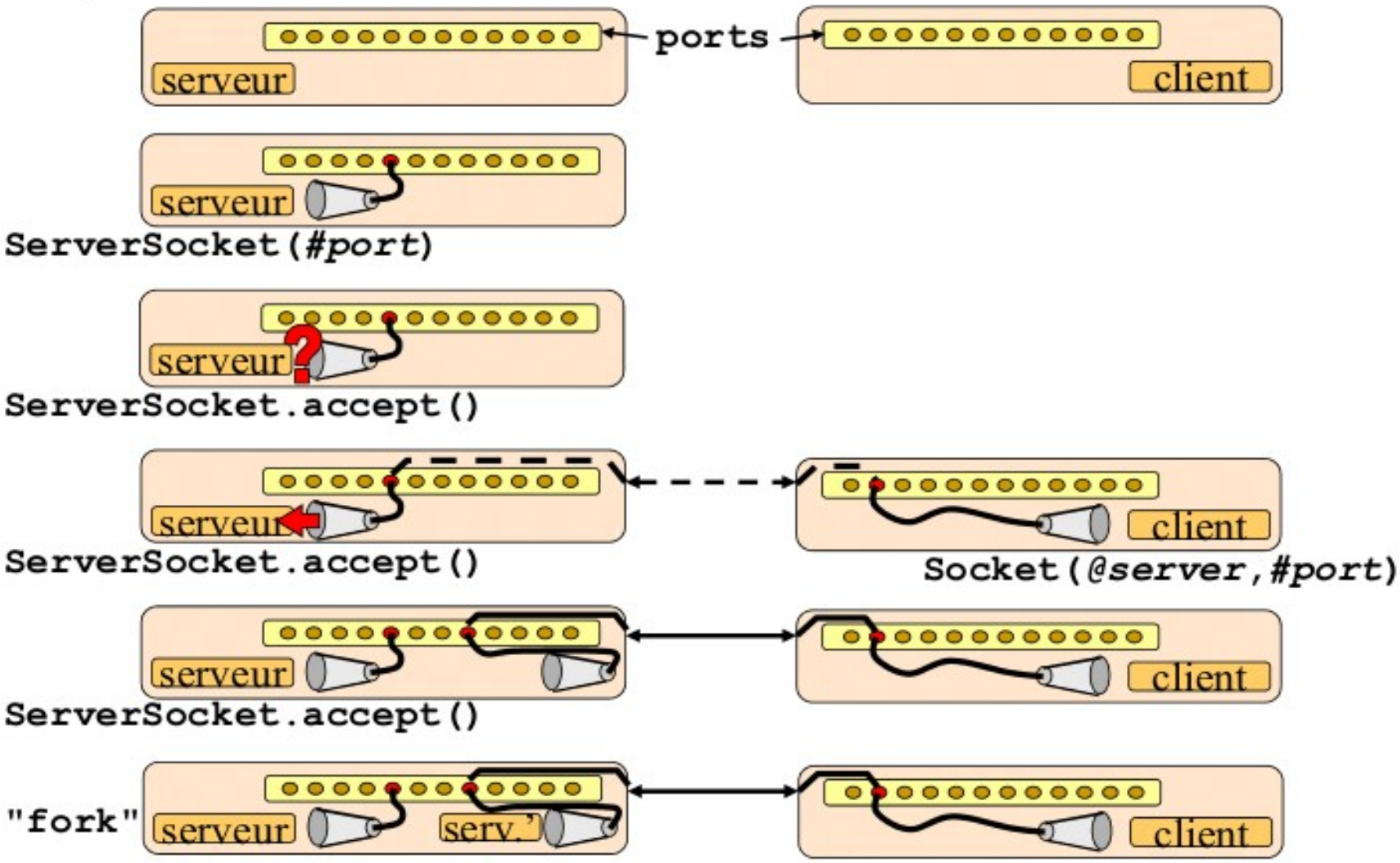
```
  // Server receives a set of msgs
  msg = new byte[MsgSize];                        // Buffer for msg
  packet = new DatagramPacket(msg,MsgSize);       // Create datagram
  s = 0;
  do {                                            // Dialog
    try {
      socket.receive(packet);                     // - recv msg
      System.out.print("Datagram " + s + " : ["); // - use msg
      for (i = 0; i < MsgSize-1; i++)
        System.out.print(msg[i]+",");
      System.out.print(msg[MsgSize-1]);
      System.out.println("] received");
    } catch (IOException e) {                     // - react on pb
      System.err.println("Receive datagram has failed!");
      System.exit(1);
    }
    s = s + 1;
  } while (msg[0] != ((byte)255));                // Stop dialog

  // Socket close on server
  socket.close();
  System.out.println("Socket closed on server");
}
```

2 - Les sockets en Java

- Sockets UDP en Java
- Sockets TCP en Java

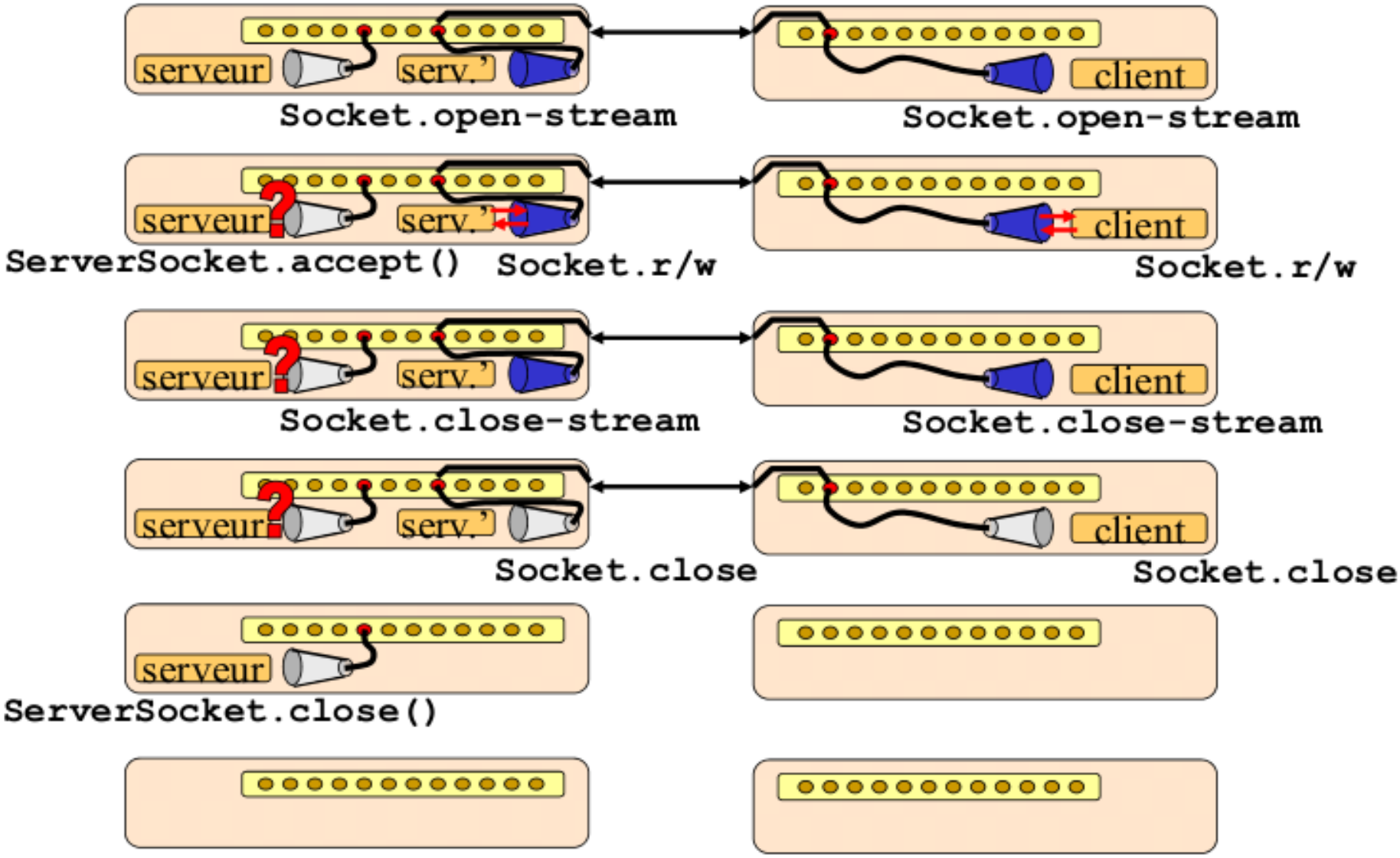
Sockets TCP en Java

Etapes d'une communication client-serveur en TCP avec Java :



Sockets TCP en Java

Etapes d'une communication client-serveur en TCP avec Java :



Sockets TCP en Java

Code client TCP :

```
public static void main(String [] argv)
       throws IOException, UnknownHostException {
  Socket socket = null;                  // Socket of the client
  byte msg[];                            // Msg to send
  int s;                                 // Msg and byte counters
  OutputStream sout = null;              // Output stream of the socket
  InputStream sin = null;                // Input stream of the socket
  // Command line parsing and init
  …
  ServerName = new String();
  ServerName = argv[0];
  ServerPort = (Integer.valueOf(argv[1])).intValue();
  // Socket creation (TCP), bound to the server and its port
  try {
    socket = new Socket(ServerName,ServerPort);
    System.out.println("Socket created and bound to server");
  } catch (UnknownHostException e) {
    System.err.println("Address of server can not be found!");
    System.exit(1);
  } catch (IOException e) {
    System.err.println("connection to server: " + ServerName +
                       "-" + ServerPort + " has failed!");
    System.exit(1);
  }
```

Sockets TCP en Java

Code client TCP (suite) :

```
  // Get native input and output streams of the socket
  try {
    sin = socket.getInputStream();
    sout = socket.getOutputStream();
    System.out.println("Input and Output streams get on client");
  } catch (IOException e) {
    System.err.println("Socket streams impossible to get");
    System.exit(1);
  }
  // Send a set of msgs to the server
  msg = new byte[MsgSize];
  for (s = 0; s < NbStep; s++) {          // Dialog
    msg = …;                              // - fill msg
    if (s == NbStep - 1)                  // - signal the end of the
      msg[0] = (byte) 255;                //   dialog to the server
    try {
      sout.write(msg);                    // - send msg
      System.out.print("Msg " + s + " : [");
      for (i=0; i < MsgSize-1; i++) System.out.print(msg[i]+",");
      System.out.print(msg[MsgSize-1] + "] sent");
    } catch (IOException e) {             // - react on pb…
      System.err.println("Write in output stream has failed!");
      System.exit(1);
    }
}
```

Sockets TCP en Java

Code client TCP (fin) :

```
  // Close the native input and output streams
  try {
    sin.close();
    sout.close();
    System.out.println("Streams closed on client");
  } catch (IOException e) {
      System.err.println("Stream close has failed on client!");
      System.exit(1);
  }
  // Close the socket of the client
  try {
    socket.close();
    System.out.println("Socket closed on server");
  } catch (IOException e) {
    System.err.println("Socket close has failed on client!");
    System.exit(1);
  }
}
```

Sockets TCP en Java

Code serveur TCP monothread :

```
public static void main(String [] argv)
        throws IOException, SecurityException {
  ServerSocket serverSocket = null; // Main socket of the server
  Socket socket = null;             // Communication socket
  byte msg[];                       // Msg to send
  int s, i;                         // Msg and byte counters
  OutputStream nsout = null;        // Output stream of socket
  InputStream nsin = null;          // Input stream of socket
  // Command line parsing and init
  if (argv.length == 1) {
    ServerPort = (Integer.valueOf(argv[0])).intValue();
  } else {
    System.out.println("Usage: java Server <server port nb>");
    System.exit(1);
  }
  // Socket creation (TCP), bounded to server and service port
  try {
    serverSocket = new ServerSocket(ServerPort);
    System.out.println("Socket created and bound to port");
  } catch (IOException e) {
    System.err.println("Socket creation and binding failed!");
    System.exit(1);
  }
```

Sockets TCP en Java

Code serveur TCP monothread (suite) :

```
// Sequential server waits for connections
try {
  System.out.println("Server waiting for conn. request...");
  socket = serverSocket.accept();
  System.out.println("Server has accepted a connection");
} catch (IOException e) {
  System.err.println("Socket connection has failed on server:"+
                     " IO error during connection");
  System.exit(1);
} catch (SecurityException e) {
  System.err.println("Socket connection has failed on server:"+
                     " Security manager refused the conn.");
  System.exit(1);
}
// Get input and output streams on the communication socket
try {
  nsin = socket.getInputStream();
  nsout = socket.getOutputStream();
  System.out.println("Input and Output streams get on server");
} catch (IOException e) {
  System.err.println("Socket streams impossible to get");
  System.exit(1);
}
```

Sockets TCP en Java

Code serveur TCP monothread (suite) :

```
// Server receives a set of msgs
msg = new byte[MsgSize];                // Buffer to store message
s = 0;
do {                                    // Dialog
  try {
    nsin.read(msg);                     // - receive a msg
    System.out.print("Msg " + s + " : ["); // - use the msg
    for (i = 0; i < MsgSize-1; i++)
      System.out.print(msg[i]+",");
    System.out.print(msg[MsgSize-1]);
    System.out.println("] received");
  } catch (IOException e) {             // - react on pb …
    System.err.println("Read in server input stream failed!");
    System.exit(1);
  }
  s++;
} while (msg[0] != (byte) 255);         // End of the dialog

// Close the input and output streams
try {
  nsin.close();
  nsout.close();
  System.out.println("Streams closed on client");
} catch (IOException e) {
  System.err.println("Stream close has failed on client!");
  System.exit(1);
}
```

Sockets TCP en Java

Code serveur TCP monothread (fin) :

```
  // Close the socket used to communicate with the client
  try {
    socket.close();
    System.out.println("Comm. socket is closed on the server");
  } catch (IOException e) {
    System.err.println("Comm. socket close failed on server!");
    System.exit(1);
  }

  // Close the main socket of the server
  try {
    serverSocket.close();
    System.out.println(« Main socket is closed on the server");
  } catch (IOException e) {
    System.err.println(« Main socket close failed on server!");
    System.exit(1);
  }
}
```

3 – Flux de données de haut niveau en TCP

Flux de haut niveau

Les flux natifs des sockets sont des flux de « bytes » :

- Les données doivent être transformées en tableaux de « bytes »
- Théoriquement simple, mais désagréable à implanter!

On crée des routines et objets qui réalisent la transformation :

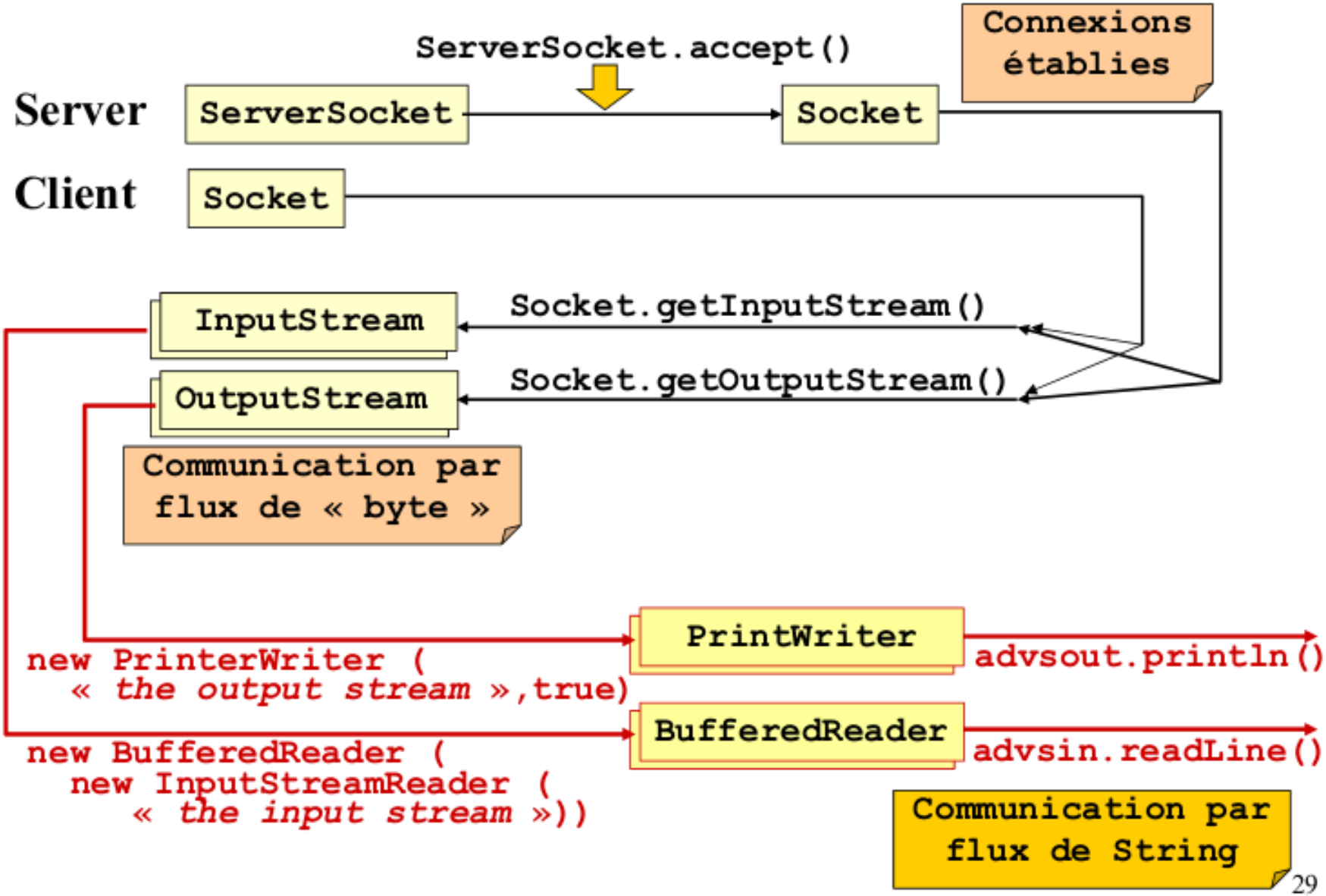
données → tableau de « bytes »

Et la transformation inverse :

tableau de « bytes » → données

Et on peut envoyer et recevoir directement des objets comme des **`Strings`** (ou des **`float`**, ou des **`double`** …) !

Flux de haut niveau



Programmation par « sockets-Java »

Flux de haut niveau

Code client

```
try {   // Connecion et ouverture des « flux avancées »
   socket = new Socket(host,serv);
   advsout = new PrintWriter(socket.getOutputStream(), true);
   advsin = new BufferedReader(
                  new InputStreamReader(socket.getInputStream()));
} catch (IOException e) {
   System.err.println("Pb a la connection/ouverture des flux");
   System.exit(1);
}
try {   // Ecriture d'un msg au serveur
   advsout.println("est-ce que tu recois ce message ?");
} catch (Exception e) {
   System.err.println("erreur sur le write ");
   System.exit(1);
}
try {   // Lecture de la reponse du serveur
   msg = advsin.readLine();
} catch (IOException e) {
   System.err.println("erreur reception ");
   System.exit(1);
}
```

Flux de haut niveau

Code serveur

```
try {  // Creation de la socket du serveur
  serverSocket = new ServerSocket(port);
} catch (IOException e) {
  System.err.println("Probleme de creation de la socket...");
  System.exit(1);
}
try {  // Acceptation de la connexion du client et ouverture des
       // « flux avancés »
  socket = serverSocket.accept();
  advsout = new PrintWriter(socket.getOutputStream(), true);
  advsin = new BufferedReader(
                 new InputStreamReader(socket.getInputStream()));
} catch (IOException e) {
  System.err.println("Erreur sur l'accept..");
  System.exit(1);
}
try {  // Lecture d'un message du client
  buf = advsin.readLine();
} catch (IOException e) {
  System.err.println("erreur reception ");
  System.exit(1);
}
try {  // Ecriture d'une reponse au client
  advsout.println("message bien recu sur le port" + port);
} catch (Exception e) {
  System.err.println("erreur sur le write ");
  System.exit(1);
}
```

Questions ?