Technologies Java

Houssem MAHMOUDI
houssem.mahmoudi@ensicaen.fr

Chapitre 1 : Programmation réseau avec Java

1. Notion de flux
2. Packages et classes
3. Les Sockets
   a. Les Sockets côtés serveur
   b. Les Sockets côtes clients
4. Sockets en mode connectés
5. Sockets en mode non connectés

I. Notion de flux

- Un flux (ou stream) est un canal de communication qui permet de gérer le transfert de données, que ce soit pour la lecture ou l'écriture.
- Java distingue deux types de flux :
  - **Flux binaires** : pour traiter des données brutes sous forme d'octets.
  - **Flux texte** : pour manipuler des données sous forme de caractères.
- Dans la suite de cours on va s'intéresser au flux réseau.

II. Packages et classes

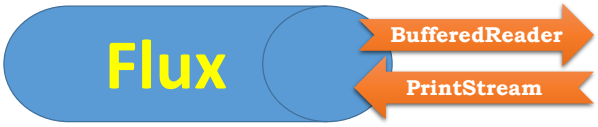
1. Packages

- Le package **java.io.*** : pour gérer le flux d'entrée sorties.
- Le package **java.net.*** : pour gérer le flux via le réseau, établir des connexions client-serveur, se communiquer via TCP ou UDP, manipuler le protocole HTTP,…

II. Packages et classes

2. Classes

- **Flux texte** :
  - Lecture des données : classe « **BufferedReader** » : utilisée avec la classe *InputStreamReader*.
  - Ecriture des données : classe « **PrintStream** / BufferedWriter » : utilisée avec la classe *OutputStreamReader*.
- Remaqrue : La classe *PrintStream* présente plus des méthodes que *BufferedWriter*.



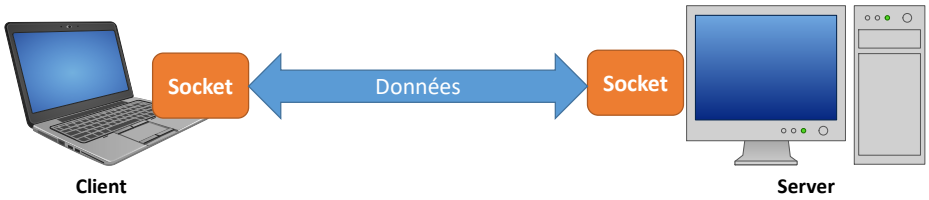
II. Packages et classes

2. Classes

- **Flux binaires** :
  - Lecture des données : classe « **BufferedInputStream** » : Utilisée avec la classe *InputStream*.
  - Ecriture des données : classe « **BufferedOutputStream** » : Utilisée avec la classe *OutputStream*.

III. Les sockets

- En Java, la communication entre systèmes à travers un flux réseau est réalisée via les **sockets**.



- En Java, une socket est un point de communication entre deux machines sur un réseau.
- Une socket est identifiée par une combinaison unique : **Adresse IP:Port**
- Exemple : 192.168.10.10**:**8080

III. Les sockets – Serveur

- Créer une socket **coté serveur** :
  - Classe : ServerSocket
  - Constructeur : ServerSocket(int port), créé une socket attachée au port spécifié.
  - La méthode **accept()** de l'objet ServerSocket : permet **d'attendre** une demande de connexion entrante, de l'accepter, et retourner une socket de communication avec le client.
- Squelette d'une socket coté serveur :

```
ServerSocket socketServer = new ServerSocket(9090);
Socket service = socketServer.accept();
```

Serveur se bloque en attente d'une connexion sur le port **9090**

III. Les sockets – Serveur

La classe java.net.ServerSocket :

- Classe basée sur le protocole **TCP**
- **Constructeurs** :
  - public ServerSocket (int **port**) throws IOException
  - public ServerSocket (int **port**, int **count**) throws IOException
    - port : Le port d'écoute
    - count : taille de la file d'attente (50 par défaut)
- **Méthodes** :
  - public Socket accept()
  - public void close()
  - public InetAddress getInetAddress()
  - public int getLocalPort()
  - public int getSoTimeout() //Obtenir la valeur du délai d'attente
  - public void setSoTimeout(int timeout) //Définir la valeur du délai d'attente

III. Les sockets – Client

- Créer une socket **coté client** :
  - Classe : Socket
  - Constructeur : Socket(String adresseIP, int port), créé une socket en spécifiant l'adresse IP du serveur et attachée à un port libre, une demande de connexion est lancée vers le serveur.
  - Lorsque la connexion est acceptée, le constructeur termine la création de l'objet, et l'objet Socket est prête pour communiquer.
- Squelette d'une socket coté client :

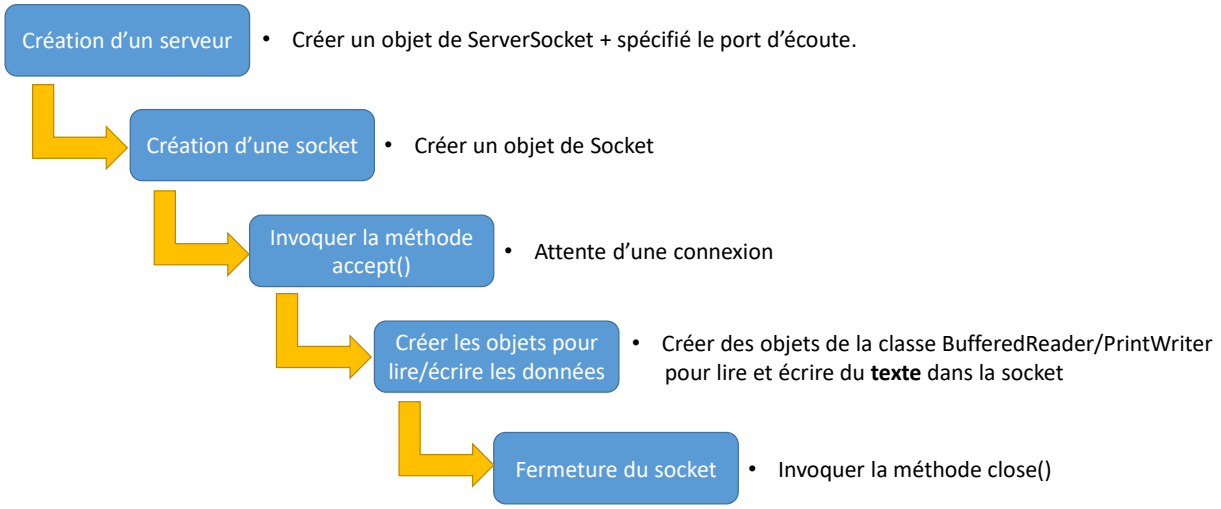
```
Socket service = new Socket ("127.0.0.1", 9090);
```

III. Les sockets – Client

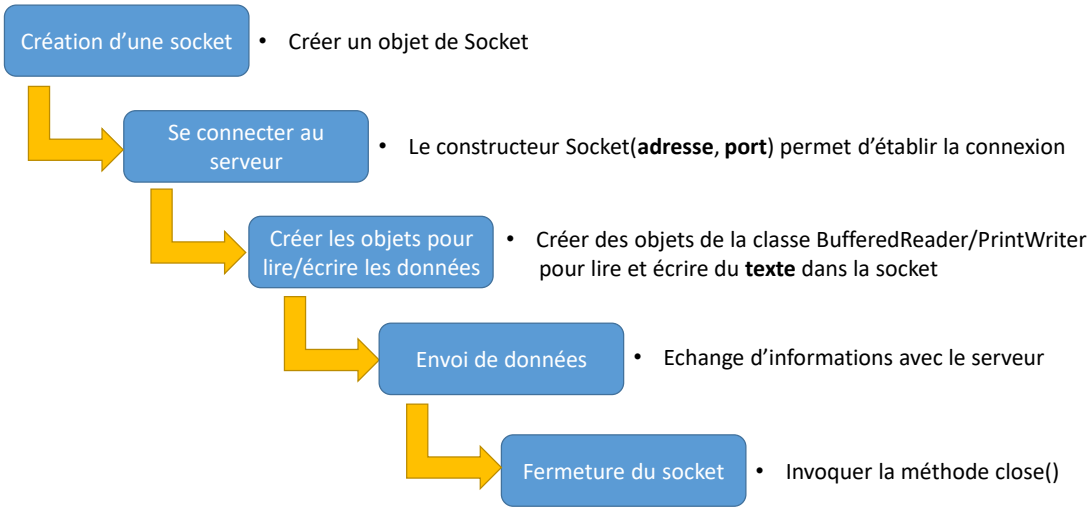
La classe java.net.Socket :

- Classe basée sur le protocole **TCP**
- **Constructeurs** :
  - public Socket (String **host**, int **port**) throws *UnknownHostException*, *IOException*
  - public Socket (InetAddress **address**, int **port**) throws IOException
    - host : adresse IP du serveur
    - port : Numéro de port d'écoute du serveur.
    - address : Objet de type *InetAddress* contenant l'adresse IP du serveur.
- **Méthodes** :
  - public boolean isConnected() //vérifie si la socket est connectée au serveur
  - public void close()
  - public int getPort() // retourne le port du serveur
  - public int getLocalPort() // retourne le port du client
  - public getInputStream() // retourne un flux de sortie pour envoyer des données
  - public int getOutputStream() // retourne un flux d'entrée pour lire les données reçues

III. Les sockets – Etapes pour créer un serveur



III. Les sockets – Etapes pour créer un client



IV. Sockets en mode connecté – Flux texte

Application 1 :

Communication textuelle entre Client et Serveur en mode connecté.

- **Fonctionnement** :
  - ❑ Le client transmet un message.
  - ❑ Le serveur renvoie une duplication de ce message.
  - ❑ L'envoi d'un message vide, par le client, termine la connexion.

IV. Sockets en mode TCP – serveur (1/2)

```
import java.io.*;
import java.net.*;

public class Serveur {
public static void main(String[] args) throws IOException {
    final int port = 9092;
    ServerSocket serversock = null;
    try {
        serversock = new ServerSocket(port);
    }
    catch(IOException ioe) {
        System.out.println("Error on the server " + ioe);
        System.exit(1);
    }
    Socket service = null;
    System.out.println("Listening for connection on port: " + port);
    try {
        service = serversock.accept();
    }
    catch(IOException ioe) {
        System.out.println("Accept connection failed !");
        System.exit(1);
    }
```


IV. Sockets en mode TCP – serveur (2/2)

```
    System.out.println("Connection successful");
    System.out.println("Waiting for data ...");

    BufferedReader input = new BufferedReader(new InputStreamReader(service.getInputStream()));
    PrintWriter output = new PrintWriter((service.getOutputStream()),true);

    do {
        String inputLine = input.readLine();
        if(inputLine.isEmpty()) {
            break;
        }
        System.out.println(" - Client: "+ inputLine);
        output.println(inputLine + " - " + inputLine);
    }while(true);

    output.println("Connection terminated - Bye");
    output.close();
    input.close();
    service.close();
    serversock.close();
   }
}
```

IV. Sockets en mode TCP – client (1/2)

```
import java.io.*;
import java.net.*;

public class Client {
public static void main(String[] args) throws IOException {
    final int port = 9092;
    Socket client = null;
    BufferedReader input = null;
    PrintWriter output = null;
    try {
        client = new Socket("127.0.0.1", port);
        input = new BufferedReader(new InputStreamReader(client.getInputStream()));
        output = new PrintWriter(client.getOutputStream(),true);
    }
    catch(UnknownHostException e) {
        System.out.println("Unknown Host ! ");
        System.exit(1);
    }
    catch(IOException ioe) {
        System.out.println("Cannot connect to the server");
        System.exit(1);
    }
```


IV. Sockets en mode TCP – client (2/2)

```
    System.out.println("Connection successful to the server:"+client.getRemoteSocketAddress());
    BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

    String userInput;
    do {
        System.out.print("Client: ");
        userInput = stdin.readLine();
        output.println(userInput);
        String res = input.readLine();
        System.out.println("Server: " + res);
    }while(userInput!="");

    stdin.close();
    output.close();
    input.close();
    client.close();
    }
}
```

Standard input

IV. Sockets en mode connecté – Flux binaires (fichier PDF)

Application 2 :

Uploader un Communication PDF vers le Serveur.

- **Fonctionnement** :
  - ❑ Le client procède à la transmission (upload) d'un fichier PDF.
  - ❑ Le serveur accepte le fichier PDF transmis.
  - ❑ La connexion se termine après la fin de la transmission.

IV. Sockets en mode connecté – serveur (1/2)

```
import java.io.*;
import java.net.*;

public class PDFServeur {
  public static void main(String[] args) throws IOException{
    final int port = 12345;
    ServerSocket server = null;
    Socket soc = null;
    try
    {
        server = new ServerSocket(port);
        System.out.println("Server waiting for connection...");
        soc = server.accept();
        System.out.println("Connection successful");
        System.out.println("Waiting for data ...");
    }catch(IOException ex) {
        System.out.println("error on the server " + ex);
        System.exit(1);
    }
    String saveDir ="C:\\myFiles\\uploads";
    new File(saveDir).mkdir();
    // Receive the file
```

IV. Sockets en mode connecté – serveur (2/2)

```
    // Receive the file
    InputStream input = null;
    FileOutputStream filercv = null;
    try
    {
        input = soc.getInputStream();
        filercv = new FileOutputStream(saveDir + "\\received_file.pdf");
        byte[] buffer = new byte[4096];
        int bytesRead;
        while ((bytesRead = input.read(buffer)) != -1) {
                filercv.write(buffer, 0, bytesRead);
        }
        System.out.println("File received successfully.");
    }catch(IOException ioe) {
        ioe.printStackTrace();
    }
    filercv.close();
    input.close();
    soc.close();
    server.close();
}
}
```


IV. Sockets en mode connecté – client (1/2)

```
import java.io.*;
import java.net.*;

public class PDFClient {
   public static void main(String[] args) throws IOException {
      String serverAddress = "127.0.0.1";
      final int port = 12345;

      Socket client = null;
      FileInputStream fileIn = null;
      OutputStream output = null;

      String filePath = "C:\\myFiles\\sample.pdf";
      try {
          client = new Socket(serverAddress, port);
          fileIn = new FileInputStream(filePath);
          output = client.getOutputStream();
          byte[] buffer = new byte[4096];
          int bytesRead;
          while ((bytesRead = fileIn.read(buffer)) != -1) {
                  output.write(buffer, 0, bytesRead);
          }
```

IV. Sockets en mode connecté – client (2/2)

```
            System.out.println("Fichier envoyé avec succès.");
        }
        catch(IOException e) {
             System.out.println("Error on the client " + e);
        }
        output.close();
        fileIn.close();
        client.close();
    }
}
```


V. Sockets en mode non connecté

- Utilisation de la classe java.net.**DatagramSocket** : Utilisée à la fois pour l'envoi et la réception d'un paquet UDP via une socket (*DatagramPacket*).
- Constructeurs :
  - public DatagramSocket ()
  - public DatagramSocket (int port)
  - public DatagramSocket (int port, InetAddr laddr)
- Méthodes :
  - public void receive (DatagramPacket p)
  - public void send (DatagramPacket p)
  - public void close()
  - public byte[] getData() //Retourne les données du paquet sous forme de tableau de bytes
  - public int getLength() //Retourne la longueur des données utiles dans le paquet.

V. Sockets en mode non connecté

Étapes pour une communication UDP :

- **Côté serveur :**
  - Créer un objet **DatagramSocket** lié à un port spécifique.
  - Préparer un **DatagramPacket** pour recevoir les données.
  - Utiliser la méthode **receive**() pour attendre des paquets.
  - Traiter les données reçues.
- **Côté client :**
  - Créer un objet **DatagramSocket**.
  - Préparer les données à envoyer sous forme de tableau de bytes.
  - Créer un **DatagramPacket** contenant les données, l'adresse IP, et le port de la destination.
  - Utiliser la méthode **send**() pour envoyer le paquet.

VI. Sockets en mode non connecté – serveur (1/2)

```
import java.net.DatagramSocket;
import java.net.DatagramPacket;

public class UDPServer {
   public static void main(String[] args) {
      final int port = 54321;
      DatagramSocket datasoc = null;
      DatagramPacket rcvpacket = null;
      try
      {
          datasoc = new DatagramSocket(port);
          System.out.println("UDP Server wainting on the port " + port+ "\n");

          //Prepare Buffer and packet for received data
          byte[] rcvdata = new byte[1024];
          rcvpacket = new DatagramPacket(rcvdata, rcvdata.length);

          //Blocking method
          datasoc.receive(rcvpacket);
          String msgclient = new String(rcvpacket.getData(), 0, rcvpacket.getLength());
          System.out.println("Client Message : " + msgclient);
```

VI. Sockets en mode non connecté – serveur (2/2)

```
            //Prepare the response to the client
            String resp = "Message well received";
            byte[] resp2client = resp.getBytes();
            DatagramPacket respacket = new DatagramPacket(resp2client,
            resp2client.length, rcvpacket.getAddress(), rcvpacket.getPort());

            //Send the response to the client
            datasoc.send(respacket);
            System.out.println("info: Response sent to client.");

        }catch(Exception e) {
             e.printStackTrace();
        }
        datasoc.close();
    }
}
```


VI. Sockets en mode non connecté – client (1/2)

```
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;

public class UDPClient {
   public static void main(String[] args) {
       String serverAddress = "127.0.0.1";
       int serverPort = 54321;
       DatagramSocket clientSocket = null;
       try {
            clientSocket = new DatagramSocket();

            //Prepare the message
            String msgClient = "Hello My UDP Server !";
            byte [] bufferClient = msgClient.getBytes();
            InetAddress serverIP = InetAddress.getByName(serverAddress);

            //Send packet to Server
            DatagramPacket clientPacket = new DatagramPacket(bufferClient, bufferClient.length,
            serverIP, serverPort);
            clientSocket.send(clientPacket);
            System.out.println("info: Message sent to server.");
```

VI. Sockets en mode non connecté – client (2/2)

```
            //Prepare a packet to receive the response
            byte[] bufferServer = new byte[1024];
            DatagramPacket serverPacket = new DatagramPacket(bufferServer,
            bufferServer.length);
            clientSocket.receive(serverPacket);

            //Show the server message
            String respServer = new String(serverPacket.getData(), 0,
            serverPacket.getLength());
            System.out.println("Server response: "+respServer);

        }catch(Exception e) {
             e.printStackTrace();
        }
        clientSocket.close();
    }
}
```