COURS *PROGRAMMATION SYSTÈME*

élaboré par Sofien Chtourou

Chapitre III : programmation réseau avec les sockets

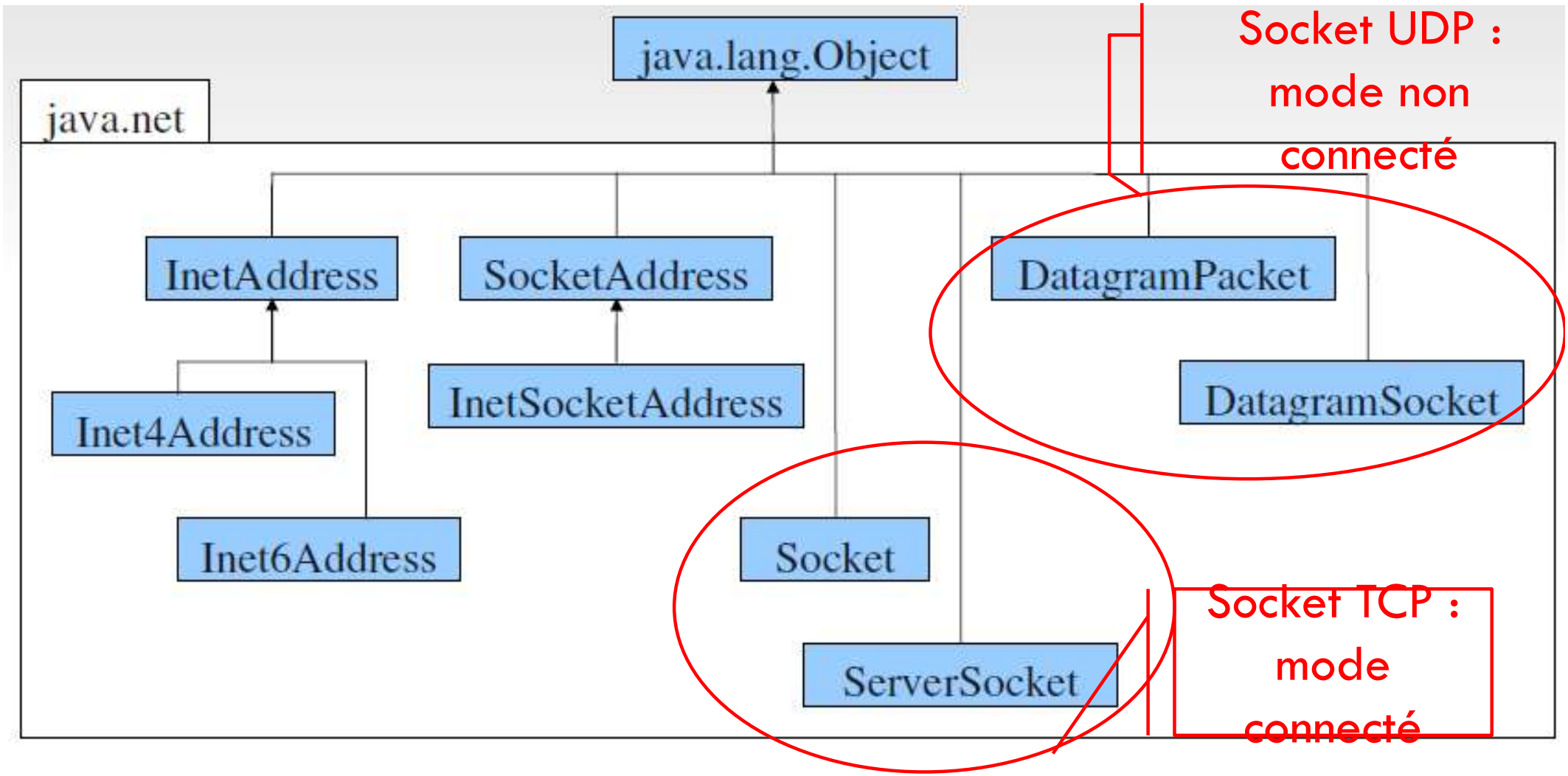
Introduction

- Un socket est un port de communication ouvert au niveau des couches réseaux et permettant de faire passer des flux de données.
- Socket java/ Socket C.
- La communication est en point à point en mode client/serveur.
- La communication est bidirectionnelle.
- La connexion se fait en mode connecté (TCP) ou non connecté (UDP).

Introduction

- Connexion réseau :
  - Adresse IP (ou nom DNS Domain Name System) de la machine.
  - Numéro de port.
- Pourquoi les ports
  - Sur une même machine, plusieurs services sont accessibles simultanément.
  - Ports logiques allant de 1 à 65535.
  - Les ports de 1 à1023 sont réservés aux services courant : http (80), SMTP (25), FTP, ….

Les Sockets en Java : le package java.net



InetAdresse

- Classe InetAdresse : manipulation des adresses IP ou nom DNS d'une machine.
  - Les deux versions d'IP (V4 32 bits , V6128 bits) sont supportées.
  - Pas de constructeur.
- 3 méthodes statiques : l'exception UnknownHostException
  - public static InetAddress **getByName(String host)**
    - Détermine l'adresse IP de la machine host.
  - public static InetAddress[] **getAllByName(String host)**
    - Retourne un tableau d'adresses IP selon la configuration du DNS.
  - public static InetAddress **getLocalHost()**
    - Retourne l'adresse IP de la machine locale.

InetAdresse

- Exemple

```
import java.net.*;
class InetAddressTest
{
public static void main(String args[]) {
            try{
                        InetAddress Address = InetAddress.getLocalHost();
                        System.out.println(Address.getHostAddress());
            }catch (UnknownHostException e) {System.out.println (e.toString())}
   }
}
```

- Ce code permet d'afficher l'adresse IP de la machine locale.

Socket et ServerSocket

- Deux types de Sockets qui utilisent le protocole de transport TCP.
- Classe Socket : connecteur réseau coté client:
  - Connexion à une machine distante.
  - Envoi/réception de données.
  - Fermeture d'une connexion.
- Classe ServerSocket :connecteur réseau coté serveur
  - Attachement à un port.
  - Attendre les demandes de connexion.
  - Acceptation d'une demande de connexion à un port local.

La classe Socket

- Constructeurs :
  - Public socket ()
    - Création de socket non connecté à un port.
  - public **Socket(String host, int port) throws** UnknownHostException, IOException:
    - Création d'un socket connecté à une machine host à travers un port.
  - public **Socket(InetAddress host, int port) throws IOException**
    - Création d'un socket connecté à une machine host à travers un port.

La classe Socket

- public **Socket(String host, int port, InetAddress hostLocal, int portLocal) throws IOException**
  - Création d'un socket connecté à une machine hostLocal à travers un portLocal. Le socket est connecté à une machine distante host.
- public **Socket(InetAddress host, int port, InetAddress hostLocal, int portLocal) throws IOException**
  - Création d'un socket connecté à une machine hostLocal à travers un portLocal. Le socket est connecté à une machine distante host.

La classe Socket

- Informations :
  - public InetAddress **getInetAddress();** public int **getPort()**
  - public int **getLocalPort();** public InetAddress **getLocalAddress()**
- Envoi/ réception de données :
  - public InputStream **getInputStream() throws IOException**
    - Récupérer le flux de données entrant.
  - public OutputStream **getOutputStream() throws IOException**
    - Récupérer le flux de données sortant.
- Fermeture de la connexion
  - void close () throws IOException

La classe Socket

- Options :
  - SO_LINGER
    - Attente ou non avant de fermer un socket au cas où il reste des données à envoyer
    - Méthodes **setSoLinger (boolean valid, int secondes) et getSoLinger()**
  - SO_TIMEOUT
    - Déclenchement d'exception si le délai spécifié est dépassé lors de la réception de données.
    - Méthodes **setSoTimeout (int ms) et getSoTimeout()**

La classe Socket

- Exceptions :
  - BindException : port déjà utilisé ou le programme n'a pas le droit de l'utiliser
  - ConnectException : hôte occupé ou aucun processus n'écoute sur le port.
  - NoRouteToHostException : dépassement de temps.

Exemple (socket client)


```
import java.io.*; import java.net.*;
public class Client {
static final int port = 8080;
static final String host = "127.0.0.1";
public static void main(String[] args) throws Exception {
            Socket socket = new Socket(host, port);
            BufferedReader plec = new BufferedReader( new
            InputStreamReader(socket.getInputStream()) );
            PrintWriter pred = new PrintWriter( new BufferedWriter( new
            OutputStreamWriter(socket.getOutputStream())), true);
            String str = "bonjour";
for (int i = 0; i < 10; i++) {
             pred.println(str); // envoi d'un message
            str = plec.readLine(); // lecture de l'écho
}
System.out.println("END"); // message de terminaison
pred.println("END") ;
plec.close(); pred.close(); socket.close(); } }
```

Exemple

- InputStreamReader(socket.getInputStream()) : permet de décoder un flux d'octets entrant à partir du socket en un flux de caractères.
- BufferedReader : Permet de stocker et de lire (readLine) un flux entrant de caractères.
- OutputStreamWriter(socket.getOutputStream()) : permet de coder un flux de caractères sortant à travers du socket.
- BufferedWriter : Permet de stocker un flux d'octets.
- PrintWriter : implémente la méthode println qui permet d'envoyer le flux de caractères à travers le socket

La classe ServerSocket

- Rôle : standartiste
- Gestion d'une connexion:
  - Création d'un nouvel objet ServerSocket affecté à un port : constructeur ServerSocket
  - Attente de connexion : accept()
  - Échange d'informations : getInputStream() et getoutputStream().
  - Cloture de la connexion par le client ou le serveur : close()
  - Nouvelle attente.

La classe ServerSocket

- Prise en compte des connexions
  - Un thread par connexion.

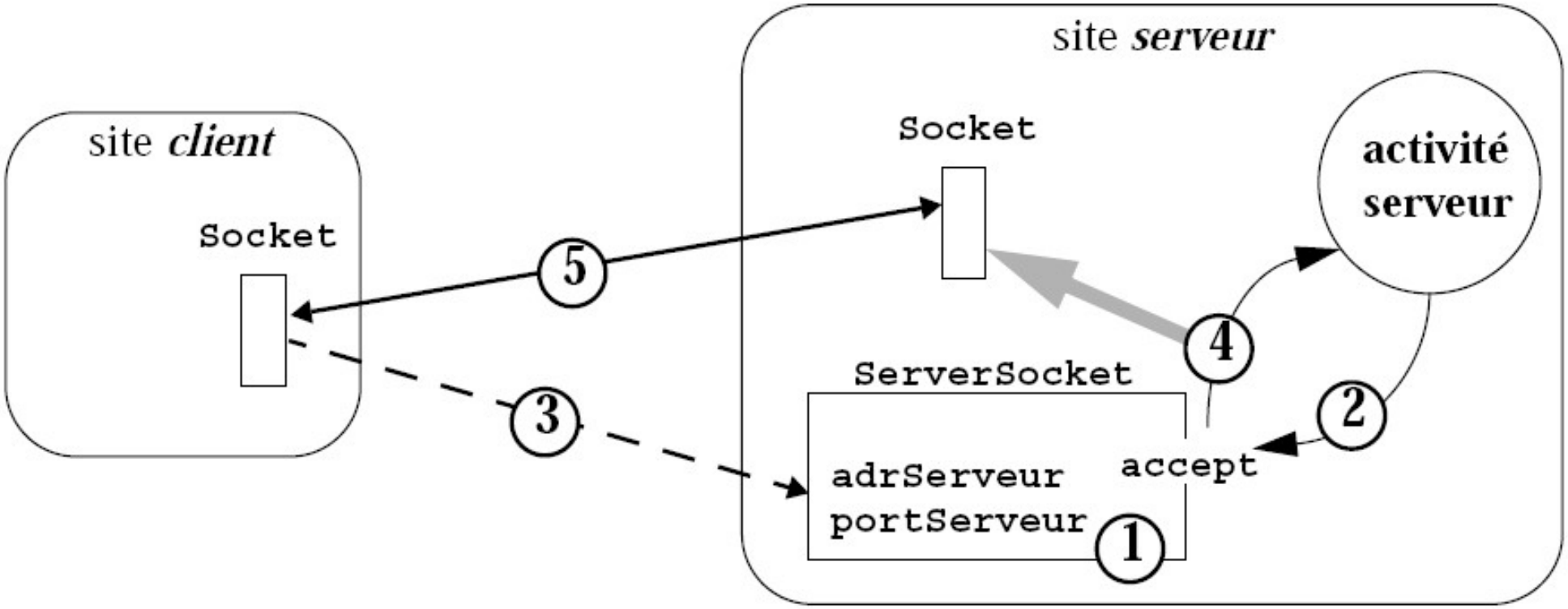
- Gestion des demandes de connexion
  - File d'attente gérée par le système
    - Taille = 50 par défaut, peut être modifiée.

La classe ServerSocket

- Constructeurs :
  - Public ServerSocket(int port) throws IOException, BindException
  - Public ServerSocket (int port, int taillefile) throws IOException, BindException
  - Public Server Socket(int port, int taillefile, InetAddress addresseAttache) throws IOException

- Le mécanisme d'établissement d'une connexion est illustré sur le schéma suivant :
  - (1) : l'activité serveur crée un ServerSocket sur le site serveur.
  - (2) : l'activité serveur appelle la méthode accept() de ce ServerSocket, ce qui la met en attente d'une manifestation de la part d'un client.
  - (3) : un site client crée un Socket en citant l'adresse et le port du serveur. Ceci sollicite le ServerSocket associé à cette adresse et ce port. Le ServerSocket crée alors un Socket sur le site serveur et le connecte à ce client.

- (4) : La méthode accept() termine en rendant ce Socket.
- (5) : Le client et le serveur peuvent alors dialoguer grâce aux flux d'entrée et de sortieofferts par les méthodes getInputStream() et getOutputStream() de leur Socket respectif.



La classe ServerSocket

- Information
  - Public InetAddress getInetAddress()
  - Public int getLocalPort()
- Options : SO_TIMEOUT
  - Doit être initialisé avant accept().
  - setSoTimeout(int ms) et getSoTimeout().

Exemple


```
import java.io.*; import java.net.*;
public class Serveur {
static final int port = 8080;
 public static void main(String[] args) throws
Exception {
ServerSocket s = new ServerSocket(port);
Socket soc = s.accept();
BufferedReader plec = new BufferedReader( new
InputStreamReader(soc.getInputStream()) ); // Un
BufferedReader permet de lire par ligne.
```

- PrintWriter pred = new PrintWriter(new BufferedWriter(new OutputStreamWriter(soc.getOutputStream())), true);
- while (true) { String str = plec.readLine(); // lecture du message
- if (str.equals("END")) break;
- pred.println(str); // renvoi d'un écho }
- plec.close();
- pred.close();
- soc.close(); } }

Serveur multi-clients

- On associe à chaque client un Thread coté serveur.
- Les Threads représentent des processus parallèles développés en Java.
- Deux façons pour créer des processus:
  - Méthode 1: Création d'objet qui hérite de la classe ***Thread***.
  - Méthode 2 :Exécution de la primitive ***new Thread()*** sur un objet qui implémente l'interface ***Runnable***.
- La classe Thread déclare une méthode ***run()*** qui doit comporter le programme principal du processus.

Exemple


```
public class exempleConcurrent extends Thread {
private static int compte = 0;
public void run() {
 int tmp = compte;
 try { Thread.sleep(1); // ms }
 catch (InterruptedException e) {
System.out.println("ouch!\n"); }
 tmp = tmp + 1;
 compte = tmp; }
```

Exemple


```
public static void main(String args[]) throws
InterruptedException {
 Thread T1 = new exempleConcurrent();
 Thread T2 = new exempleConcurrent();
 T1.start(); T2.start();
 T1.join(); T2.join();
 System.out.println( "compteur=" + compte ); } }
```

Serveur multi-clients

- La création d'un Thread est effectuée grâce à la primitive new.
- Le lancement d'un processus est effectué à travers la méthode ***start()***.

```
class ServeurThread extends Thread{
static final int port = 8080;
Socket soc=null;
ServeurThread(Socket s){
        super("threadserver");
        this.soc=s;}
```

Serveur multi-clients


```
public void run()
{
try{
BufferedReader plec = new BufferedReader( new
InputStreamReader(soc.getInputStream()) );
PrintWriter pred = new PrintWriter( new
BufferedWriter( new
OutputStreamWriter(soc.getOutputStream())), true);
… // envoi et réception de messages
plec.close(); pred.close(); soc.close();
```

```
}catch(Exception e){}
} }
public class Serveur{
 public static void main(String[] args) throws Exception
 {
ServerSocket serverSocket = new
ServerSocket(8080);
While(true){
System.out.println("serveur en ecoute");
 new ServeurThread(serverSocket.accept()).start();} }}
```

Socket UDP: Client


```
public class client {
public static void main (String arg[]){try{
// socket sur port libre & adresselocale
 DatagramSocket socket = new DatagramSocket() ;
byte[] buf = "Hello".getBytes("ASCII") ;
//creation d'un datagramme  au port 3333 de la machine locale
 DatagramPacket packet = new DatagramPacket(buf, buf.length,
InetAddress.getLocalHost(), 3333) ;
 // envoi du datagramme via la socket
 socket.send(packet);
//allocation & mise en place d'1 buffer pour recep.
  byte[] receiveBuffer = new byte[1024] ;
  packet.setData(receiveBuffer) ;
  System.out.println(packet.getLength()) ;
```

Socket UDP: Client


```
// affiche : 1024 (taille de la zone de stockage)
  // mise en attente de reception
  socket.receive(packet) ;
  System.out.println(packet.getLength()) ;
  //affiche le nombre d'octets bien recus (<= 1024)
  //construction d'une String correspondant aux octets recus
  String s = new String(receiveBuffer, 0, packet.getLength(),
"ASCII") ;
  System.out.println(s) ;
}
catch(Exception e) { }
 }
}
```

Socket UDP: Server


```
public class server {
public static void main (String arg[]) {try {
//socket d'attente de client, attachee au port 3333
DatagramSocket socket = new DatagramSocket(3333) ;
//datagramme pour la reception avec alloc. de buffer
byte[] buf = new byte[1024] ;
DatagramPacket packet=new DatagramPacket(buf,buf.length) ;
// message d'accueil
socket.receive(packet) ;//attente de reception bloquante
 String s = new String(buf, 0, packet.getLength(), "ASCII") ;
  System.out.println(s) ;
  byte[] msg = "You're welcome !".getBytes("ASCII") ;
```

Socket UDP: Server


```
// place les donnees a envoyer
packet.setData(msg) ;
socket.send(packet) ; // envoie la reponse
// replace la zone de reception
packet.setData(buf,0,buf.length) ;
}
catch (Exception e) {}
}

}
```

Exercice

- Créer une classe Serveur Multi-thread nommée « ServeurMultiClient » qui pourra gérer simultanément un nombre illimité de clients : Chaque client a sa propre socket de connexion et tourne dans un thread dédié.
- Créer un programme client qui lance 10 processus client connectés au serveur déjà développé.
- Créer un programme client qui permet de diffuser un message pour tous les clients.