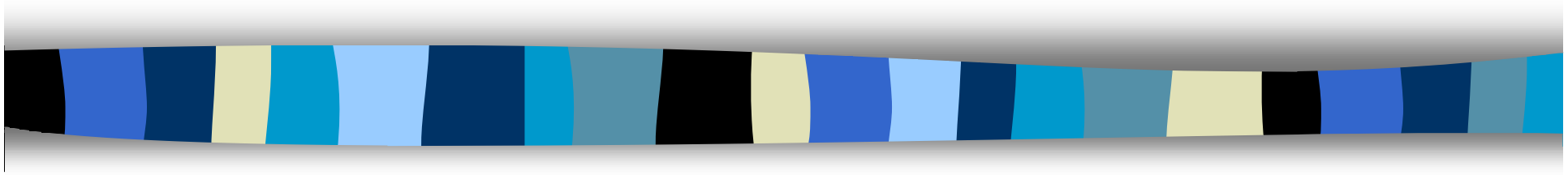
Programmation Réseau en Java



Didier Donsez

Université de Valenciennes
Institut des Sciences et Techniques de Valenciennes
donsez@univ-valenciennes.fr

L ’API Réseau de Java
`java.net`

- Classes et interfaces du paquetage `java.net`

- Adresses IP
  InetAddress
- Socket TCP
  Socket, ServerSocket
- Sockets UDP
  DatagramSocket, DatagramPacket
- Sockets MultiCast
  MulticastSocket, DatagramPacket
- Classes réseau niveau application (couche 7)
  URL, URLConnection, HttpURLConnection, JarURLConnection

La classe `java.net.InetAddress`

- Représente une adresse IP
  - utilisé par les classes **Socket** et **DatagramSocket**
- 3 méthodes statiques pour la résolution DNS

public static InetAddress **getByName(String hostname)** throws UnknownHostException

public static InetAddress[] **getByName(String hostname)** throws UnknownHostException

  - donne l 'adresse(s) de l 'hôte dont le DN est passé en paramètre

public static InetAddress **getLocalHost()** throws UnknownHostException

  - donne l 'adresse de l 'hôte local

- Exemple

```
InetAddress server;
try {
  if (args.length > 0) { server = InetAddress.getByName(args[0]); }
  else { server = InetAddress.getLocalHost(); }
  System.out.println(server); // affiche le nom du server
 } catch (UnknownHostException e) { System.out.println("Could not find this computer's address."); }
```

Exemple d'après [Rusty - chapitre 4] avec `java.net.InetAddress`

```
package java.net;          import java.util.StringTokenizer;
public class InetAddressFactory { // d'après [Rusty - chapitre 4]
  // Use a byte array like {199, 1, 32, 90} to build an InetAddressObject
  public static InetAddress newInetAddress(byte addr[])  throws UnknownHostException  {
    try {       InetAddress ia = new InetAddress();
                ia.address  =               (addr[3] & 0xFF) | ((addr[2] << 8) & 0xFF00);
                                            | ((addr[1] << 16) & 0xFF0000) | ((addr[0] << 24) & 0xFF000000);
      return ia;
    } catch (Exception e) { throw new UnknownHostException(e.toString()); }
  }
  // Use a String like 199.1.32.90 to build an InetAddressObject
  public static InetAddress newInetAddress(String s)  throws UnknownHostException {
    int num_bytes_in_an_IPv4_address = 4; byte addr[] = new byte[num_bytes_in_an_ IPv4_address];
    StringTokenizer st = new StringTokenizer(s, ".");
    if (st.countTokens() != addr.length) { throw new UnknownHostException(s + " is not a valid IP address"); }
    for (int i = 0; i < addr.length; i++) {
      int thisByte = Integer.parseInt(st.nextToken());
      if (thisByte < 0 || thisByte > 255) { throw new UnknownHostException(s + " is not a valid IP address"); }
      if (thisByte > 127) thisByte -= 256; addr[i] = (byte) thisByte;
    } return newInetAddress(addr); }  }
```

Les sockets en mode connecté
`java.net.Socket` et `java.net.ServerSocket`

- Représente une connexion fiable TCP/IP entre 2 processus (*qui peuvent ne être des JVM !*)
  - la connexion est fiable (contrôle d ’erreur, …)
  - et 2 flots de données sont établis entre les deux machines
- La connexion est asymétrique
  - Une machine est serveur : classe **ServerSocket**
    - elle attend les demandes de connexion et les accepte
  - Une machine est client : classe **Socket**
    - elle demande l ’établissement de la connexion avec le serveur
- Terminologie
  - La connexion est dite en mode connecté

Les sockets en mode connecté
Utilisation

- Coté serveur : classe **ServerSocket**
  - crée le **ServerSocket** :

    ServerSocket listenSocket = new **ServerSocket**(port);
  - attend les demandes de connexion et crée un socket pour le dialogue

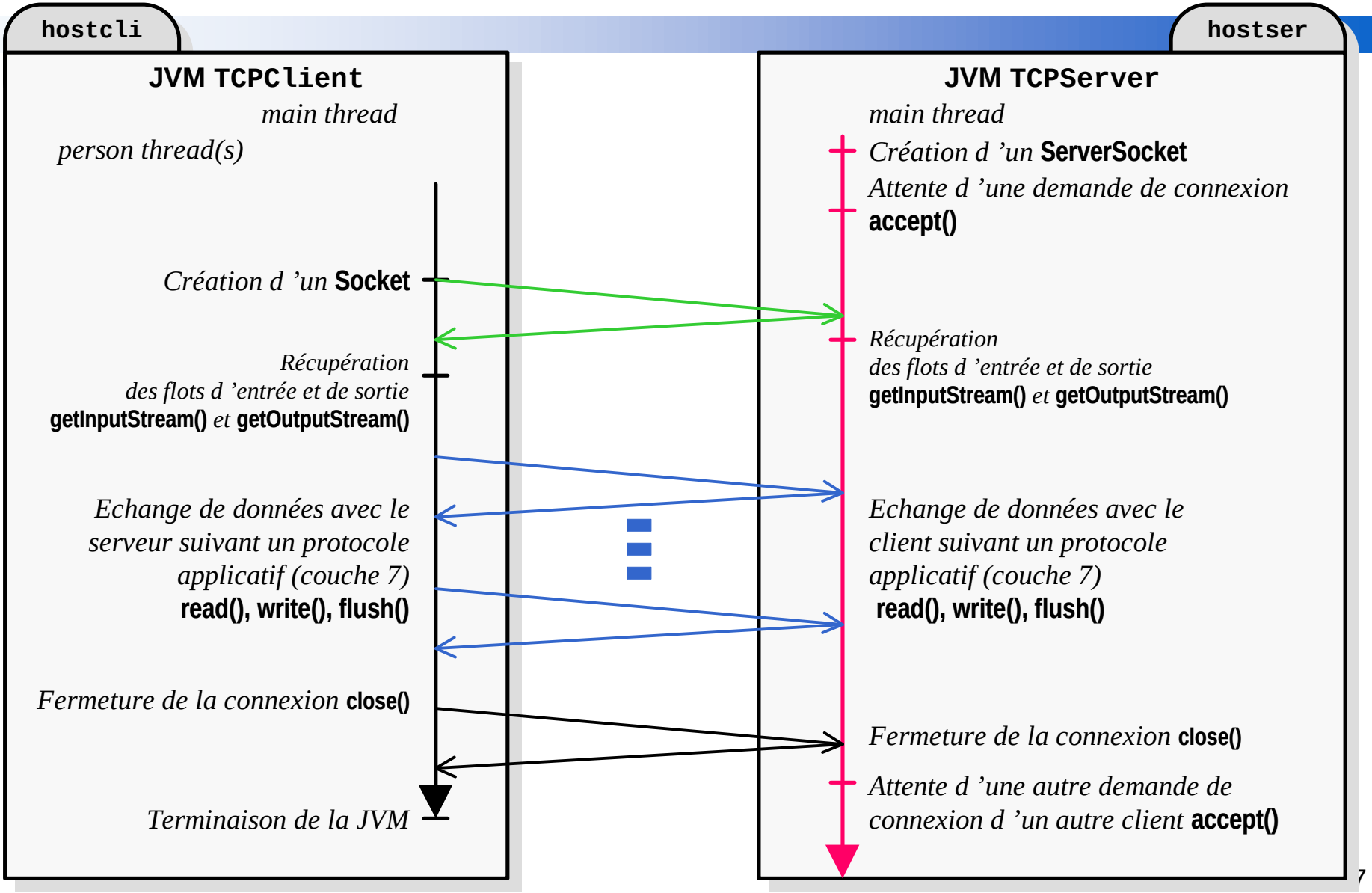
    Socket clientSocket = listenSocket.accept();
  - récupère les flux d ’entrée et de sortie

    InputStream in = clientSocket.getInputStream();
    OutputStream out = clientSocket.getOutputStream();
  - dialogue avec le serveur
- Coté client : classe **Socket**
  - crée le **Socket** :

    Socket server = new **Socket**(host,port);
  - récupère les flux d ’entrée et de sortie

    InputStream in = server.getInputStream();
    OutputStream out = server.getOutputStream();
  - dialogue avec le serveur

Socket TCP
Déroulement de l 'exécution



Socket TCP

Remarques lors du déroulement de l ’exécution

- Le serveur
  - Après la fermeture de la connexion, le serveur se met en attente du nouvelle connexion du même client ou d ’un autre
  - Si plusieurs demandes connexions arrivent simultanément, une seule est acceptée et les autres sont mises en attente jusqu ’au accept() suivant (modulo les timeouts)
    - pour accepter et traiter plusieurs connexions simultanées, la solution est de multithreadé le serveur
- Le protocole
  - Le client et le serveur doivent respecter un protocole valide d ’échange des données selon un automate.
    - le client envoie un entier mais le serveur attend un flottant
    - le client attend des données du serveur qui lui même attend des données du client

      il y a interblocage
  - Attention à l ’hétérogénéité des plates-formes client et serveur
    - un client sur Intel envoie un entier little-endian
    - le serveur sur Sparc reçoit cet entier et le traite en big-endian

Socket TCP *Remarques lors du déroulement de l 'exécution*

- Echange en mode Ligne
  - Les requêtes et les réponses sont constituées d 'une ou plusieurs lignes de texte (ASCII 7 bits, `www-url-encoded`, ...)

    Classes java.io.BufferedReader/BufferedWriter
    - Exemple de protocole en mode ligne : HTTP, FTP, SMTP, …
    - Avantage pour le deboggage : le client peut être un client telnet
      `telnet www 80`, `telnet mailhost 25`
    - Mais attention au terminateur de ligne selon les OS:
      tantôt LF **'\n'**, tantôt CR **'\r'**, tantôt CRLF **"\r\n"**, …
    - Certains protocoles limitent la longueur des lignes
- Echange en mode Bloc de bytes
  - Les requêtes et les réponses sont des blocs de bytes d 'une taille et d 'un format connus de l 'autre

    Classes java.io.DataInputStream/DataOutputStream
    - Exemples : RPC, RMI, CORBA, ...
    - Le bloc peut avoir un entête qui contient un code d 'opération et la longueur du corps du bloc qui contient les données

Exemple du Code du Client

```
import java.net.*; import java.io.*;
public class EchoClient {
  public static void main(String[] args) {
    Socket theSocket;
    DataInputStream theInputStream; DataInputStream userInput;
    PrintStream theOutputStream;
    String theLine;
   try {
      theSocket = new Socket(args[0], Integer.parseInt(args[1]));
      theInputStream = new DataInputStream(theSocket.getInputStream());
      theOutputStream = new PrintStream(theSocket.getOutputStream());
      userInput = new DataInputStream(System.in);
      while (true) {
        theLine = userInput.readLine();
        if (theLine.equals(".")) break;
        theOutputStream.println(theLine);
        System.out.println(theInputStream.readLine());
      }
    } catch (UnknownHostException e) { System.err.println(e);
    } catch (IOException e) { System.err.println(e); } } }
```

Exemple d ’un Code du Serveur

```
import java.net.*; import java.io.*;
public class EchoServer {
  void doService(Socket clientSocket) {
      DataInputStream in = new DataInputStream(clientSocket.getInputStream());
      PrintStream out = new PrintStream(clientSocket.getOutputStream());
      while (true) {String theLine=in.readLine(); out.println(theLine); }
}
  public static void main(String[] args) {
    ServerSocket listenSocket;
    try {
      listenSocket = new ServerSocket(Integer.parseInt(args[0])); // port
      while(true) {
        Socket clientSocket = listenSocket.accept();
        System.err.println("Connexion from:" + clientSocket.getInetAddress());
        doService(clientSocket);
    } catch (Exception e) { System.err.println(e); }
  }
}
```

Exemple d 'un Code d 'un Client SMTP (i)

```
// java SMTPMailer yourmailhost yourmaildomain yourcorrespondentdaddress youraddress Hi "Hello You"
import java.io.*;
import java.net.*;
public class SMTPMailer{
      static final String CFLF="\r\n";
      static final int SMTP_PORT=25;
      public static void main(String args[])
        try{
              Socket sock = new Socket(arvs[0], SMTP_PORT);
              DataInputStream in = new DataInputStream(sock.getInputStream());
              BufferedReader br = new BufferedReader(new InputStreamReader(in));
              DataOutputStream out = new DataOutputStream(sock.getOutputStream());

              SMTP(br,out,args[1], args[2], args[3], args[4], args[5]);}

              socket.close();
        } catch(IOException ioe) {
                        System.out.println("Erreur de connection : " + ioe.getMessage());
        }
      } ...
```

- Remarque :
  - l 'API JavaMail est préférable pour l 'envoi des courriers !

Exemple d 'un Code d 'un Client SMTP (ii)

```
static void SMTP(
                    BufferedReader br,      DataOutputStream out,
                    String MAILDOMAIN, String FROM, String TO, String SUBJECT, String MSG){
        String cmdstr, statusline;
        statusline = br.readLine();             System.out.print(statusline); // le serveur se présente
        cmdstr="HELO " + MAILDOMAIN + CRLF; // EHLO pour les ordres étendus
        out.writeBytes(cmdstr); out.flush(); System.out.print(cmdstr);
        statusline = br.readLine();             System.out.println(statusline);
        cmdstr ="MAIL FROM:"+ FROM + CRLF;
        out.writeBytes(cmdstr); out.flush(); System.out.print(cmdstr);
        cmdstr ="RCPT TO:" + TO + CRLF;
        out.writeBytes(cmdstr); out.flush(); System.out.print(cmdstr);
        statusline = br.readLine();             System.out.println(statusline);
        cmdstr =                    "DATA" +  CRLF
                    +               "TO:" TO +  CRLF
                    +               "SUBJECT:" + SUBJECT +  CRLF
                    +               MSG +  CRLF
                    +               "." +  CRLF;
        out.writeBytes(cmdstr); out.flush(); System.out.print(cmdstr);
        statusline = br.readLine();             System.out.println(statusline);
        cmdstr ="QUIT" +  CRLF;
        out.writeBytes(cmdstr); out.flush(); System.out.print(cmdstr);
} }
```

Les exceptions

- `java.net.BindException`
  - erreur de liaison à une adresse locale (le port est peut être déjà lié)
- `java.net.ConnectException`
  - refus de connexion par l ’hôte (pas de process qui écoute le port, …)
- `java.net.NoRouteToHostException`
  - le hôte n ’est pas joignable
- `java.net.ProtocolException`
  - erreur du protocole (TCP, …)
- `java.net.SocketException`
  - erreur du protocole (TCP, …)
- `java.net.UnknownHostException`
  - erreur de DNS

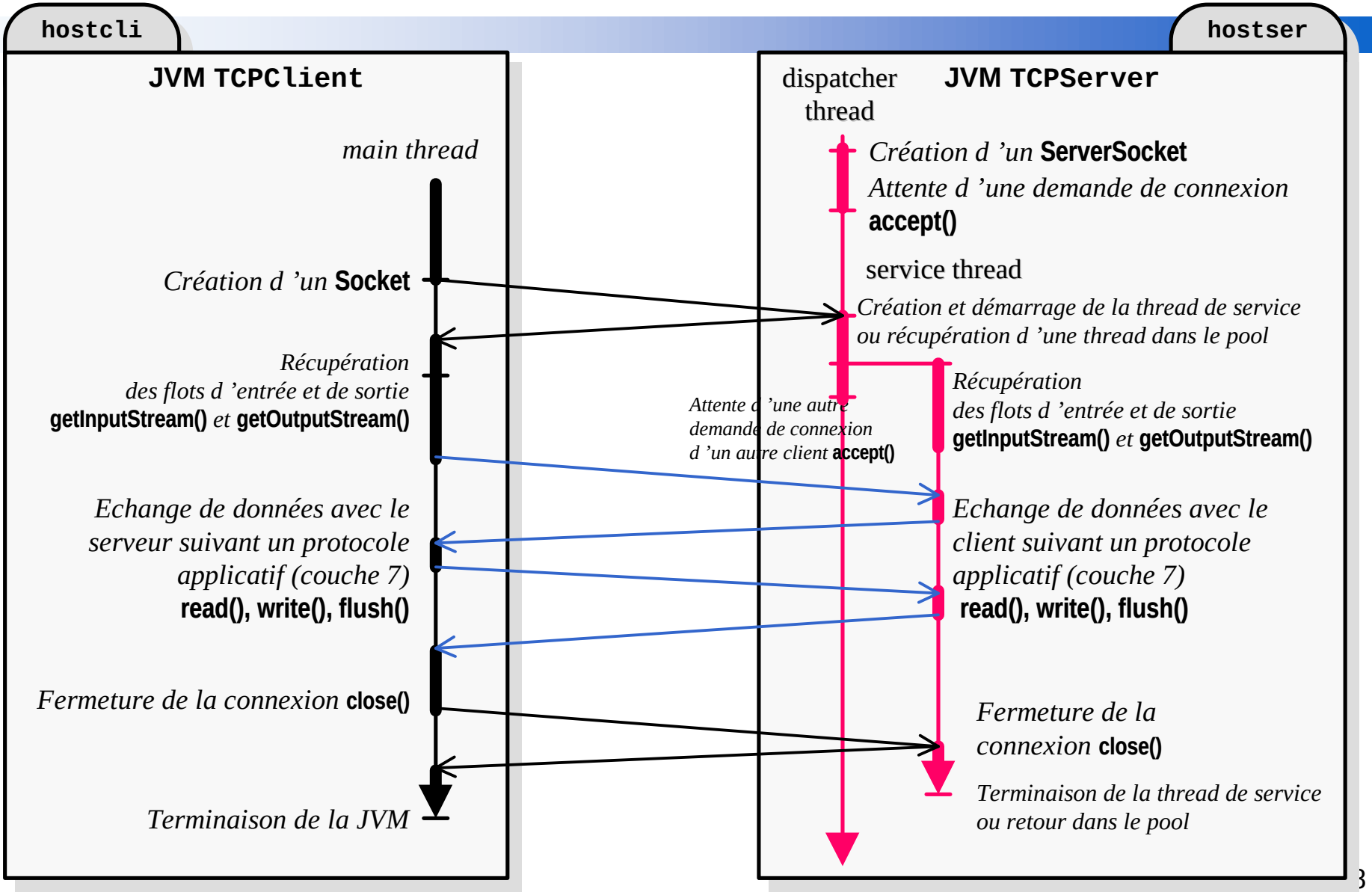
Serveur Multithreadé

- Motivation
  - Accepter (et servir) plusieurs connexions simultanées de plusieurs clients
- Méthode
  - une thread (dite *dispatcher*) attend les demandes de connexions
  - récupère un socket dédié à cette connexion avec client
  - crée une thread (dite de service) qui prend en charge le dialogue avec le client
  - et retourne en attente d ’une nouvelle demande de connexion
- Remarque
  - les threads de service peuvent être retirées d ’un pool
    - avantage : limiter le coût d ’initialisation des threads de service

      une thread sert plusieurs requêtes successives de clients différents

Socket TCP

Déroulement de l 'exécution multithreadée



Code du Serveur Multithreadé

```
import java.net.*; import java.io.*;
public class EchoServer extends Thread {
  private Socket clientSocket;
  EchoServer(Socket clientSocket) {this.clientSocket=clientSocket;}
  public static void main(String[] args) {
    ServerSocket listenSocket;
    try {
      listenSocket = new ServerSocket(Integer.parseInt(args[0])); // port
      while(true) { // le dispatcher est la thread qui exécute main()
        Socket clientSocket = listenSocket.accept();
        System.err.println("Connexion from:" + clientSocket.getInetAddress());
        EchoServer serviceThread = new EchoServer(clientSocket);
        serviceThread.start();
    } catch (Exception e) { System.err.println(e); }
  }
  public void run() { doService(clientSocket); }
  public void doService(Socket clientSocket) {
       DataInputStream in = new DataInputStream(clientSocket.getInputStream());
       PrintStream out = new PrintStream(clientSocket.getOutputStream());
       while (true) { String theLine=in.readLine(); out.println(theLine); }
  }
}
```

Personnaliser un type de Socket

- Motivation
  - il est souvent nécessaire de traiter les données à envoyer ou reçues d ’un `java.net.Socket` (compression, conversion, filtrage, chiffrage, …)
    - Post-traitement des données après réception (ex: Décompression, Déchiffrage)
    - Pré-traitement des données avant envoi (ex: Compression, Chiffrage)
- 2 Méthodes
  - Soient deux classes étendant **java.io.FilterOutputStream** et **java.io.FilterInputStream**
    - **CompressionOutputStream** et **CompressionInputStream**
  - Méthode 1:

```
Socket sock = new Socket(...);
InputStream in = new CompressionInputStream(sock.getInputStream());
OutputStream out = new CompressionOutputStream(sock.getOutputStream(), compressRate);
```

  - Méthode 2: Encapsulation par héritage

Personnaliser un type de Socket

- Méthode 2 : encapsulation par héritage
  - 1- Dériver 2 nouvelles classes des classes
    java.io.FilterOutputStream et **java.io.FilterInputStream**
  - 2- Dériver 2 nouvelles classes des classes
    java.net.Socket et **java.net.SocketServer**
  - Exemple
    CompressionOutputStream et **CompressionInputStream**
    CompressionSocket et **CompressionSocketServer**
- Remarque
  - Utilisé par les RMI pour personnaliser la couche Transport au moyen de *RMIClientSocketFactory/RMIClientSocketFactory*
    - voir jdk1.2.2\docs\guide\rmi\sockettype.doc.html

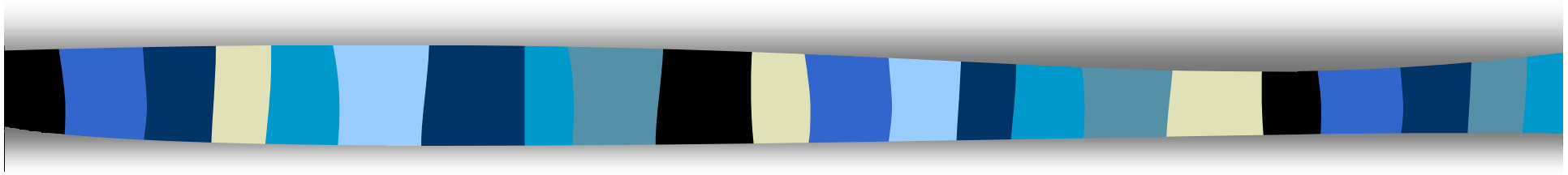
Personnaliser un type de Socket
Sous classe de Socket

```
import java.io.*;          import java.net.*;
class CompressionSocket extends Socket {
   private int compressionrate;
   private InputStream in;
   private OutputStream out;
   public CompressionSocket(int compressionrate) {
            super(); this.compressionrate=compressionrate;
   }
   public CompressionSocket(String host, int port, int compressionrate) throws IOException {
            super(host, port); this.compressionrate=compressionrate; }
   public InputStream getInputStream() throws IOException {
      if (in == null) in = new CompressionInputStream(super.getInputStream());
      return in; }
   public OutputStream getOutputStream() throws IOException {
      if (out == null) out = new CompressionOutputStream(super.getOutputStream(), compressionrate);
      return out;  }
   public synchronized void close() throws IOException {
      OutputStream o = getOutputStream(); o.flush(); super.close();
   } }
```

Personnaliser un type de Socket
Sous classe de SocketServer

```
import java.io.*;
import java.net.*;
class CompressionServerSocket extends ServerSocket {
   public CompressionServerSocket(int port, int compressionrate) throws IOException  {
      super(port);
   }
   public Socket accept() throws IOException {
      Socket s = new CompressionSocket(compressionrate);
      implAccept(s);
      return s;
   }
}
```

Programmation UDP en Java



Didier Donsez

Université de Valenciennes
Institut des Sciences et Techniques de Valenciennes
donsez@univ-valenciennes.fr

Les sockets en mode non connecté

- Rappel sur UDP
  - couche de transport non fiable en mode non connecté
    - contrôle des erreurs mais pas de reprise sur erreur, pas d ’ACK, séquencement des paquets non garanti
    - plus simple que TCP et plus efficace pour certaines applications (envoi de vidéo, audio, mesures, TFTP, NFS, DNS …)
- Principe
  - l ’envoyeur (*sender*) envoie un datagramme (paquets de données sur un socket que le receveur (*receiver*) écoute.
- La classe `java.net.DatagramSocket`
  - Représente le socket local ou distant en mode non connecté
- La classe `java.net.DatagramPacket`
  - Représente un packet (ou Datagramme) à envoyer ou reçu

Les sockets en mode non connecté
Utilisation

- Coté émetteur
  - crée le **DatagramSocket**

    DatagramSocket ms = new DatagramSocket(receiverInetAddress);
  - construit un **DatagramPacket**

    byte[] data = new byte[len]; // data doit être « rempli » : *data = {'H', 'e', 'l', 'l', 'o'};*

    DatagramPacket outputPacket = new DatagramPacket(data, data.length, receiverInetAddress, port);
  - envoie le **DatagramPacket** au recepteur

    ms.send(outputPacket);
- Coté récepteur
  - crée le **DatagramSocket** lié à un port d ’écoute

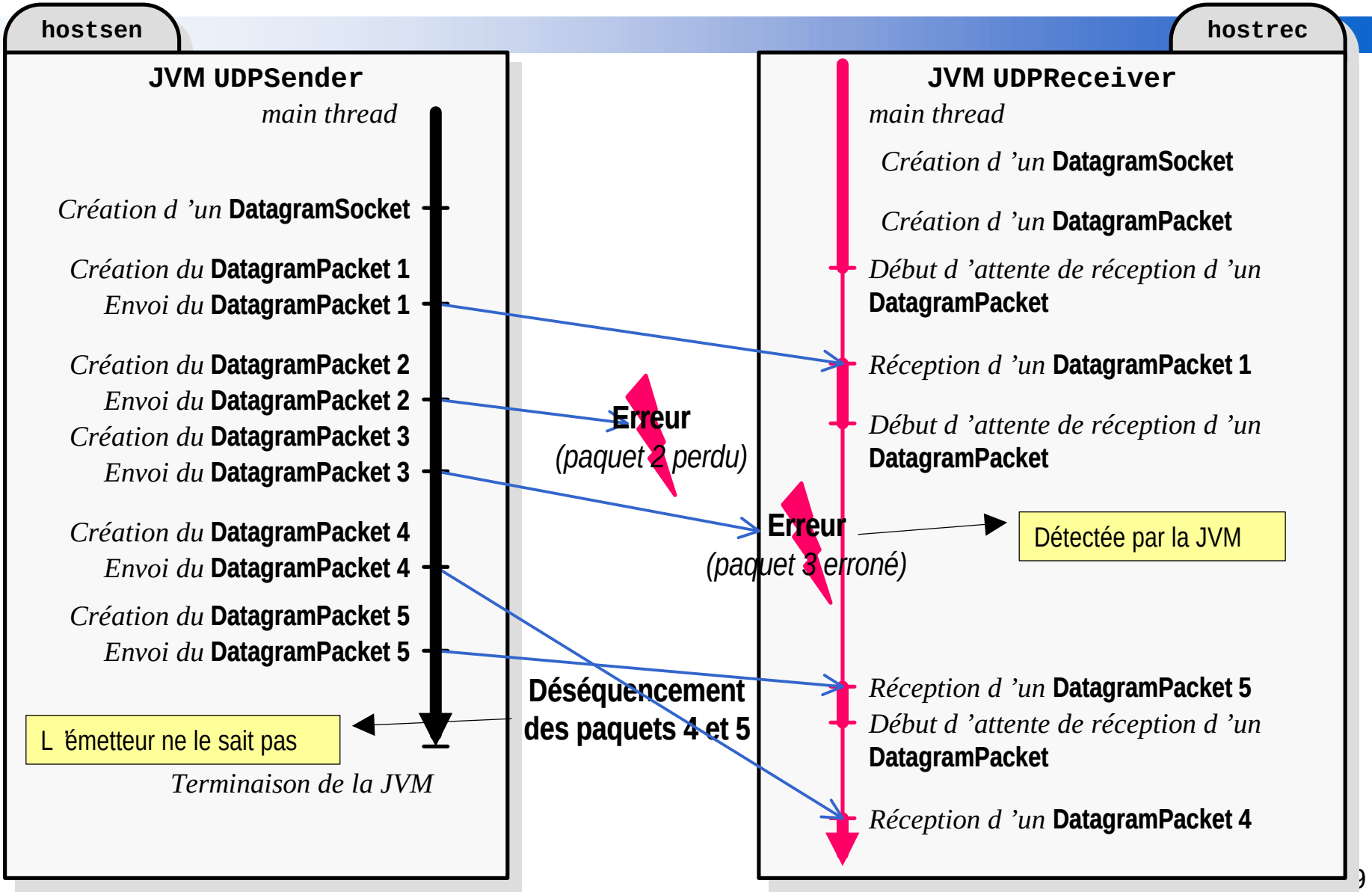
    DatagramSocket theSocket = new **DatagramSocket**(port);
  - construit un **DatagramPacket** pour la réception

    byte[] incomingData = new byte[MAXLEN]; // buffer vide

    DatagramPacket incomingPacket = new DatagramPacket(data, data.length);
  - réception le **DatagramPacket** du paquet puis utilisation

    theSocket.receive(incomingPacket);

Socket UDP
Déroulement de l 'exécution



Code du Emetteur
Exemple

```
import java.net.*; import java.io.*;
public class UDPSender {
  public static void main(String[] args) {
    try {
      InetAddress receiver = InetAddress.getByName(args[0]);
      int port = Integer.parseInt(args[1]);
      DatagramSocket theSocket = new DatagramSocket();
      DataInputStream userInput = new DataInputStream(System.in);
      while (true) {
        String theLine = userInput.readLine();
        if (theLine.equals(".")) break;
        byte[] data = new byte[theLine.length()];
        theLine.getBytes(0, theLine.length(), data, 0);
        DatagramPacket theOutput =
                new DatagramPacket(data, data.length, receiver, port);
        theSocket.send(theOutput);
      }
    } catch (Exception e) { System.err.println(e);}
  } }
```

Code du Récepteur
Exemple

```
import java.net.*; import java.io.*;
public class UDPReceiver {
  public static void main(String[] args) {
    try {
      // construction d'un DatagramSocket
      int port = Integer.parseInt(args[0]);
      DatagramSocket ds = new DatagramSocket(port);
      // construction d'un DatagramPacket
      int len = Integer.parseInt(args[1]);
      byte[] buffer = new byte[len];
      DatagramPacket incomingPacket = new DatagramPacket(buffer, buffer.length);
      while (true) {
        ds.receive(incomingPacket);
        String s =
              new String(incomingPacket.getData(), 0, 0, incomingPacket.getLength());
        System.out.println(incomingPacket.getAddress()
                                  + " at port " + incomingPacket.getPort() + " says " + s);
      }
    } catch (Exception e) { System.err.println(e); }
  } }
```

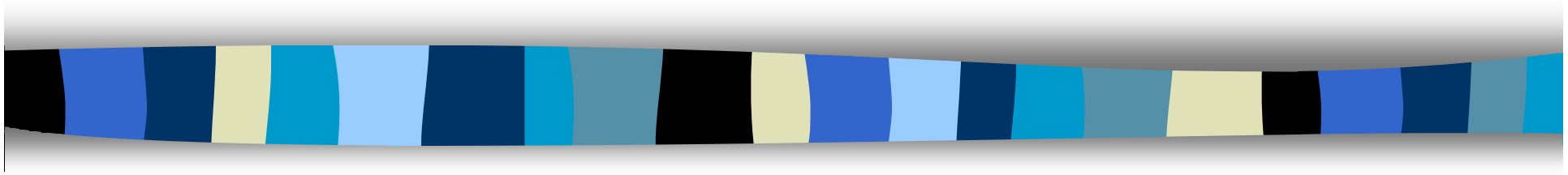
Remarques

- Fragmentation IP [Stevens Tome1 Chap11 Sect5]
  - le DatagramPacket peut être fragmenté pour les couches inférieur (IP,Ethernet,…) par la couche UDP
    - ! La taille maximun d ’un paquet IP est de 65535 octets
  - Il est reassemblé par la couche UDP du récepteur avec d ’être remis
- Fiabilité sur UDP
  - La fiabilité peut être garantie au niveau applicatif par le développeur
    - par l ’ajout de numéro de séquence, de date (horloge de Mattern), de timeout pour les reprises ...
    - dans les messages contenant les données ou de service
  - Lecture
    - les 3 tomes de Raynal

La classe `java.net.DatagramPacket`

- Représente un packet (ou Datagramme)
  à envoyer ou reçu par UDP ou IP MultiCast
  - Contient un tableau de byte
    qui peut être récupérer ou positionner
    avec les méthodes getData()/setData(), setLength()/getLength()
  - Contient également l'adresse et le port de l 'émetteur
    méthodes getAddress() getPort()
  - Contient également l'adresse et le port du récepteur
    méthodes setAddress() setPort()
- Peut être structuré pour l 'envoi de messages structurés
  - `DatagramPacket` ne peut être dérivé : classe finale
  - la structure peut contenir un numéro de séquence, horloge de Mattern, … pour assurer un niveau de fiabilité
  - peut contenir un objet sérialisé (ou externalisé)
    uniquement si l 'émetteur et le récepteur sont des JVMs.

Programmation MultiCast en Java



Didier Donsez

Université de Valenciennes
Institut des Sciences et Techniques de Valenciennes
donsez@univ-valenciennes.fr

Utilisation des sockets en mode diffusion restreinte (multicast)

- Rappel sur l 'IP MultiCast
  - couche de transport non fiable en mode diffusion restreinte
    - contrôle des erreurs mais pas de reprise sur erreur, pas d 'ACK, séquencement des paquets non garanti
    - utilisé pour les applications de diffusion restreinte
      vidéo/audio-conférence, cours de bourse … lookup de JINI
    - voir http://www.univ-valenciennes.fr/CRU/OR/
  - UDP Unicast vs IP MultiCast
    - UDP envoie un paquet de données d 'un point (hôte) vers un autre (hôte,port)
    - IP Multicast envoie une suite de paquets de données d 'un point (hôte) vers un groupe d 'N hôtes qui souhaitent recevoir ces données
  - Groupe MultiCast
    - spécifié par une adresse de classe D (de `224.0.0.1` à `239.255.255.255`)
    - et par un port (UDP)

Utilisation des sockets en mode diffusion multicast

- Principe
  - le diffuseur (*multicaster*) rejoint un groupe multicast et envoie un datagramme sur ce groupe
  - les récepteurs (*receiver*) qui ont rejoint le groupe reçoivent les datagrammes envoyés par les diffuseurs.
- La classe `java.net.MulticastSocket`
  - dérive de `java.net.DatagramSocket`
  - Représente le socket en diffusion
- La classe `java.net.DatagramPacket`
  - Représente un packet (ou Datagramme) à envoyer ou reçu

Les sockets en mode diffusion restreinte *Utilisation*

- Coté diffuseur
  - crée le **MulticastSocket**

    MulticastSocket ms = new MulticastSocket(); ms.joinGroup(groupInetAddress);
  - construit un **DatagramPacket** d 'émission

    byte[] data = new byte[len]; // data doit être « rempli » *: data = {'H', 'e', 'l', 'l', 'o'};*

    DatagramPacket outputPacket = new DatagramPacket(data, data.length, groupInetAddress, port);
  - envoie le **DatagramPacket** au groupe de diffusion

    ms.send(outputPacket, ttl);
- Coté récepteur
  - crée le **MulticastSocket** et rejoindre le groupe

    MulticastSocket ms = new MulticastSocket(port); ms.joinGroup(groupInetAddress);
  - construit un **DatagramPacket** de réception

    byte[] data = new byte[len]; // data doit être « rempli »

    DatagramPacket incomingPacket = new DatagramPacket(data, data.length);
  - reçoit le **DatagramPacket** diffusé

    ms.receive(incomingPacket);

Code du Diffuseur
Exemple

```
import java.net.*; import java.io.*;
public class MulticastSender {
  public static void main(String[] args) {
    try {
      InetAddress ia = InetAddress.getByName(args[0]);
      int port = Integer.parseInt(args[1]);
      byte ttl = (byte)Integer.parseInt(args[2]);
      MulticastSocket ms = new MulticastSocket();
      ms.joinGroup(ia);
      DataInputStream userInput = new DataInputStream(System.in);
      while (true) {
        String theLine = userInput.readLine(); if (theLine.equals(".")) break;
        byte[] data = new byte[theLine.length()];
        theLine.getBytes(0, theLine.length(), data, 0);
        DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
        ms.send(dp,ttl);
      }
      ms.leaveGroup(ia);
      ms.close();
    } catch (Exception e) { System.err.println(e); } } }
```

Code du Receveur
Exemple

```
import java.net.*; import java.io.*;
public class MulticastReceiver {
  public static void main(String[] args) {
    try {
      // paquet de réception
      byte[] buffer = new byte[65509];
      DatagramPacket dp = new DatagramPacket(buffer, buffer.length);

      InetAddress ia = InetAddress.getByName(args[0]); // adresse de classe D
      int port = Integer.parseInt(args[1]);
      MulticastSocket ms = new MulticastSocket(port);
      ms.joinGroup(ia);

      while (true) {
        ms.receive(dp);
        String s = new String(dp.getData(), 0, 0, dp.getLength());
        System.out.println(s);
      }
    } catch (Exception e) { System.err.println(e); }
  } }
```

Remarques sur MulticastSocket

- Un socket n ’a pas besoin être membre d ’un groupe multicast pour y envoyer de message.
- Les adresses de classes D déjà réservées sont sur

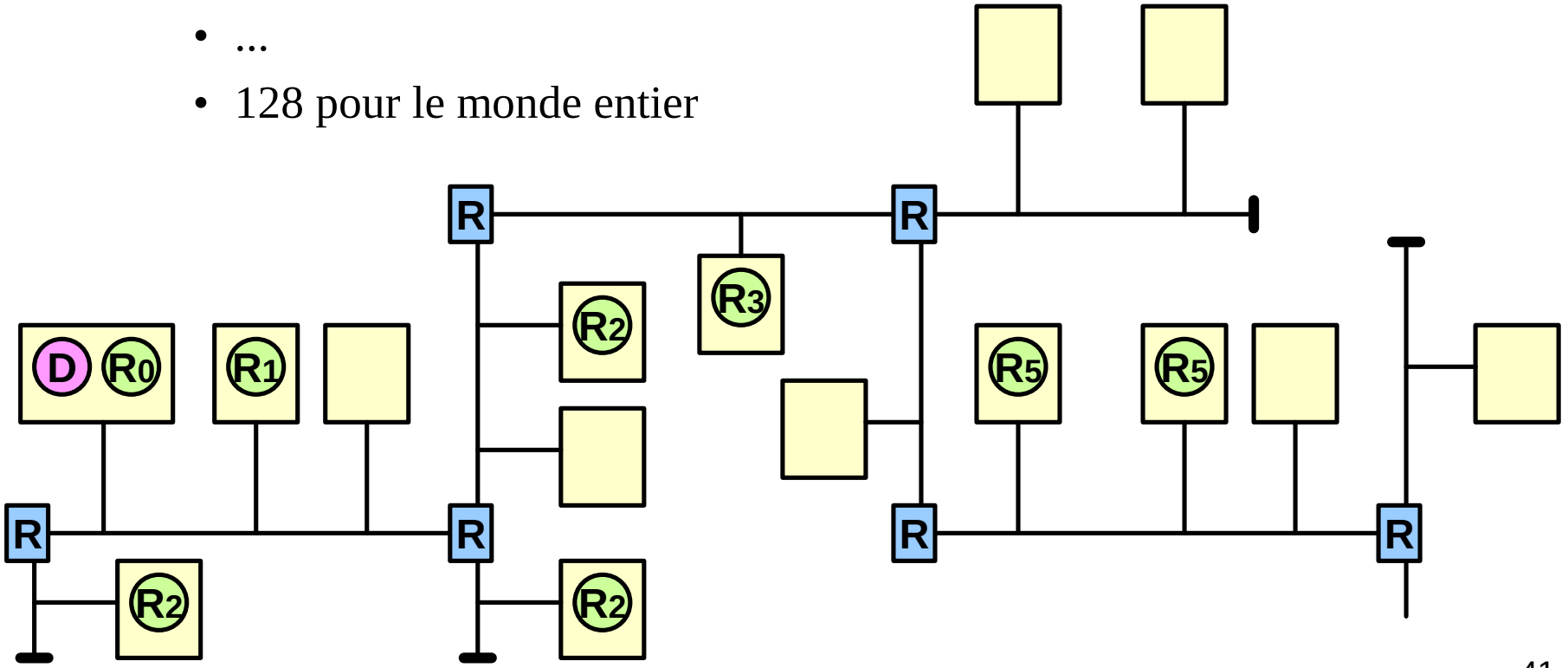
  ftp://ftp.isi.edu/in-notes/iana/assignments/multicast-addresses

```
...
224.0.18.000-224.0.18.255 Dow Jones
224.0.19.000-224.0.19.063 Walt Disney Company
...
```

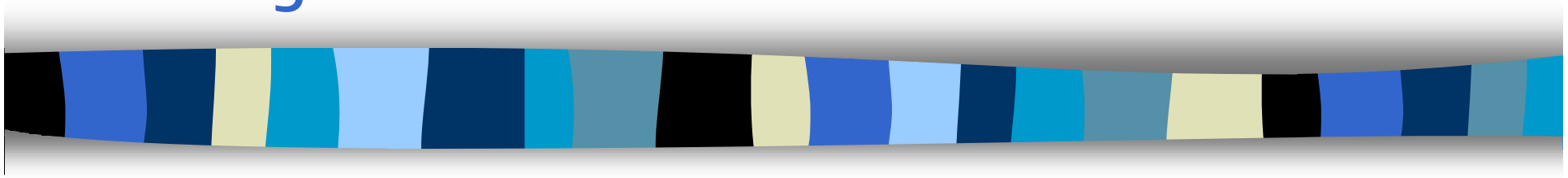
- Les applets ne sont pas autorisées à utiliser les sockets multicast
- Quand un message est envoyé, tous les membres d ’un groupe reçoivent le message dans l ’intervalle du TTL (time-to-live)

Remarques sur le Time-To-Live

- Le TTL (time-to-live) représente le nombre de routeurs (sauts/*hops*) le paquet peut traverser avant d ’être abandonné.
  - 0 pour l ’émetteur
  - 1 pour le LAN
  - ...
  - 128 pour le monde entier



Les classes de Connexion `java.net.URL` `java.net.URLConnection`



Didier Donsez

Université de Valenciennes

Institut des Sciences et Techniques de Valenciennes

donsez@univ-valenciennes.fr

java.net.URL

- Définit une URL
- Méthodes

String getProtocol(), String getHost(), String getPort(), String getRef()

InputStream openStream()

permet d 'obtenir un InputStream d 'une connexion au serveur

URLConnection openConnection()

permet d 'obtenir un URLConnection

- Constructeurs

```
URL localfile = new URL("/users/donsez/pages/cours/index.html");
URL gamelan = new URL("http://www.gamelan.com/pages/");
URL gamelanGames = new URL(gamelan, "Gamelan.game.html");
URL gamelanNetwork = new URL(gamelan, "Gamelan.net.html");
URL gamelanNetworkBottom = new URL(gamelanNetwork, "#BOTTOM");
URL gamelanNetwork2 = new URL("http", "www.gamelan.com", "/pages/Gamelan.net.html");
```

java.net.URL
Méthodes

- Exemple avec les méthodes getXXX()

```
import java.net.*; import java.io.*;
public class ParseURL {
        public static void main(String[] args) throws Exception {
          URL aURL = new URL("http://java.sun.com:80/docs/books/" +
                "tutorial/index.html#DOWNLOADING");
           System.out.println("protocol = " + aURL.getProtocol());
           System.out.println("host = " + aURL.getHost()+ " filename = " + aURL.getFile());
           System.out.println("port = " + aURL.getPort() + " ref = " + aURL.getRef());          } }
```

- Utilitaires

  static String URLDecoder.decode(String urlencoded)

  static String URLEncoder.encode(String str)

  - méthodes statiques de conversion

    `x-www-form-urlencoded` vers/depuis `String`

java.net.URL

Exemple de récupération d 'un document

```
import java.net.*;
import java.io.*;
        public class URLReader {
            public static void main(String[] args) throws Exception {
                URL aurl = new URL(args[0]);
                BufferedReader in = new BufferedReader(
                                    new InputStreamReader(aurl.openStream()));
                String inputLine;
                while ((inputLine = in.readLine()) != null) System.out.println(inputLine);
                in.close();
            }
        }
```

La classe de connexion `java.net.URLConnection`

- Définit une connexion à une URL
  - superclasse abstraite indépendante du schéma de connexion (`http, ftp, file, mailto, …`)

- Sous-classes
  - **HttpURLConnection, JarURLConnection**

java.net.URLConnection

- Utilisation
  - 1- l ’instance est récupéré de URL.openConnection()
  - 2- configuration des paramêtres de la connexion
    - **setAllowUserInteraction, setDoInput, setDoOutput, setIfModifiedSince, setUseCaches, setRequestProperty**
  - 3- connexion **connect()** et obtention d ’un InputStream
  - 4- récupération des champs d ’entête (*header fields*)
    - **getContent, getHeaderField, getContentEncoding, getContentLength, getContentType, getDate, getExpiration, getLastModifed**

java.net.URLConnection

Exemple de récupération d ’un document

```
import java.net.*;
import java.io.*;
        public class URLConnectionReader {
            public static void main(String[] args) throws Exception {
                URL aurl = new URL(args[0]);
                URLConnection aurlcnx = aurl.openConnection();
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(aurlcnx.getInputStream()));
                String inputLine;
                while ((inputLine = in.readLine()) != null) System.out.println(inputLine);
                in.close();
            } }
```

- Exercice :
  - écrire un robot qui récupère une hiérarchie de documents HTML à partir d ’une URL

java.net.URLConnection

Exemple d 'envoi de mail

```
import java.net.*;
import java.io.*;
        public class MailSender {
           public static void main(String[] args) throws Exception {
              URL mailto = new URL("mailto:"+args[0]);
              URLConnection mailtocnx= mailto.openConnection();
              mailtocnx.connect();
              PrintStream p = new PrintStream(mailtocnx.getOutputStream());
              p.println("Subject: Test\r\n\r\nSalut !");
              p.close();
           } }
```

java.net.HttpURLConnection

- Sous interface de `URLConnection`
  - Récupération d ’un document via une connexion HTTP
- Méthodes
  - Positionnement des champs de la requête HTTP
    - **setRequestedMethod()**
  - Récupération des champs de la réponse HTTP
    - **getResponseCode(), getResponseMessage()**
  - Clôture de la session HTTP
    - **disconnect()** pour les connexions keep-alive

java.net.HttpURLConnection

Exemple de récupération d 'un document

```
import java.net.*;  import java.io.*;
public class HttpURLConnectionReader {
  public static void main(String[] args) throws Exception {
    URL httpurl = new URL(args[0]);
    HttpURLConnection httpcnx = (HttpURLConnection)httpurl .openConnection();
    if(httpcnx.getResponseCode()== HttpURLConnection.HTTP_OK) {
          BufferedReader in = new BufferedReader(
                    new InputStreamReader(httpcnx.getInputStream()));
          String inputLine;
          while ((inputLine = in.readLine()) != null) System.out.println(inputLine);
          in.close();
    } else { System.err.println(httpcnx.getResponseMessage()); }
} }
```

java.net.JarURLConnection

- Sous interface de `URLConnection`
  - Récupération d ’un Jar file ou d ’une de ses entrées
- Syntaxe des URL

  `jar:<url>!/{entry}` `!/` est un séparateur
  - un Jar file : **jar:http://www.foo.com/bar/baz.jar!/**
  - un Jar file : **jar:file:/local/dev/bar/baz.jar!/**
  - un répertoire : **jar:http://www.foo.com/bar/baz.jar!/COM/foo/**
  - un entrée : **jar:http://www.foo.com/bar/baz.jar!/COM/foo/Quux.class**
- Exemple

```
url = new URL("jar:http://www.foo.com/bar/baz.jar!/");
JarURLConnection jarConnection=(JarURLConnection)url.openConnection();
Manifest manifest = jarConnection.getManifest();
```

Les gestionnaires pour URL

- Architecture logicielle ouverte autour d 'URL
  - Ajout de nouveaux schémas (`https, vod, ...`)
  - Ajout de nouveaux types de ressources (`image/fract`)
  - Personnalisation des gestionnaires existants (`http, ...`)
- 3 classes d 'objets
  - Gestionnaire de schéma
    - sous-classe de URLStreamHandler
    - par défaut sun.net.www.protocol.<schema>.Handler
  - Gestionnaire de connexion
    - sous-classe de URLConnection
  - Gestionnaire de contenu (type/soustype MIME)
    - sous classe de ContentHandler
    - par défaut : sun.net.www.content.<type>.<soustype>

Les gestionnaires pour URL

- Personnalisation des Gestionnaires
  - Constructeurs d ’URL
  - Méthodes d ’URL

  - Gestionnaire de schéma
    - **setURLStreamHandlerFactory(URLStreamHandlerFactory factory)**
  - Gestionnaire de contenu (type/soustype MIME)
    - **setContentHandlerFactory(ContentHandlerFactory factory)**

Conclusion

- API Réseau de Java
  - Socket/ServerSocket, DatagramSocket et MulticastSocket fournissent un API réseau bas niveau au dessus de TCP/IP, UDP/IP et IP MultiCast
- Cependant
  - les messages et flux de données ne sont pas structurés
  - Solution 1 pour du Client-Serveur
    - Des outils plus adaptés (RPC, RMI, CORBA) masquent au développeur les détails de la connexion et le codage des messages
  - Solution 2 pour du Client-Serveur
    - `java.net` offre aussi des classes URL et URLConnection pour la récupération de données au dessus de protocoles comme HTTP, FTP, … (voir cours sur HTTP).

Bibliographie
Architecture et Principe d 'IP

- Généralités
  - Guy Pujolle, "Les réseaux", Ed Eyrolles , 3ème éd., 2000, ISBN 2-212-09119-2
    - Chapitres 12 et 16 : IP dans les grandes lignes
    - mise à jour régulière
- Détail
  - W.R. Stevens, " TCP/IP Règles et Protocoles " Volume 1,2 et 3, Ed Vuibert (Addison-Wesley pour la VA de 1994), 1998, ISBN 2-7117-8639-0
    - très détaillé, plus que complet mais commence à dater
  - Douglas E. Comer, « Internetworking with TCP/IP volume I », Prentice Hall
  - Douglas E. Comer & David L. Stevens, « Internetworking with TCP/IP volume II (Implementation and Design Issues) », Prentice Hall
  - Douglas E. Comer & David L. Stevens, « Internetworking with TCP/IP volume III (Client / Server Programming & Apps.) » , Prentice Hall
- Liens
  - http://www.yahoo.com/Computers/Software/Protocols/IP/

Bibliographie

Programmation des Sockets

- Voir le cours « Socket sous Unix »

- Gilles Roussel, Étienne Duris, "Java et Internet, Concepts et programmation", Ed Vuibert, 01/2000, ISBN : 2-7117-8654-4
  - détaille bien la programmation TCP, UDP et MultiCast en Java
- Elliotte Rusty Harold, « Programmation Réseau avec Java », Ed O Reilly, 1997, ISBN 2-84177-034-6
- Scott Oaks, Henry Wong, "Java Threads", 2nd Edition,Ed Oreilly, 1999, 1-56592-418-5, existe en français
  - voir Chapitre 5 pour les serveurs multithreadés
- David Flanagan, « Java in a Nutshell », Oreilly
- E.R Harold, "Java I/O", Ed Oreilly, 1999, 1-56592-485-1
  - voir Chapitre 5 pour l ’utilisation de Input/Output streams
- Robert Orfali, Dan Harkey, “ Client/Server Programming with Java and Corba ”, 2ème édition, 1998, Ed Wiley, ISBN 0-471-24578-X.
  - voir le chapitre 10 qui compare les Sockets à CORBA