Rappels programmation réseau Java

Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

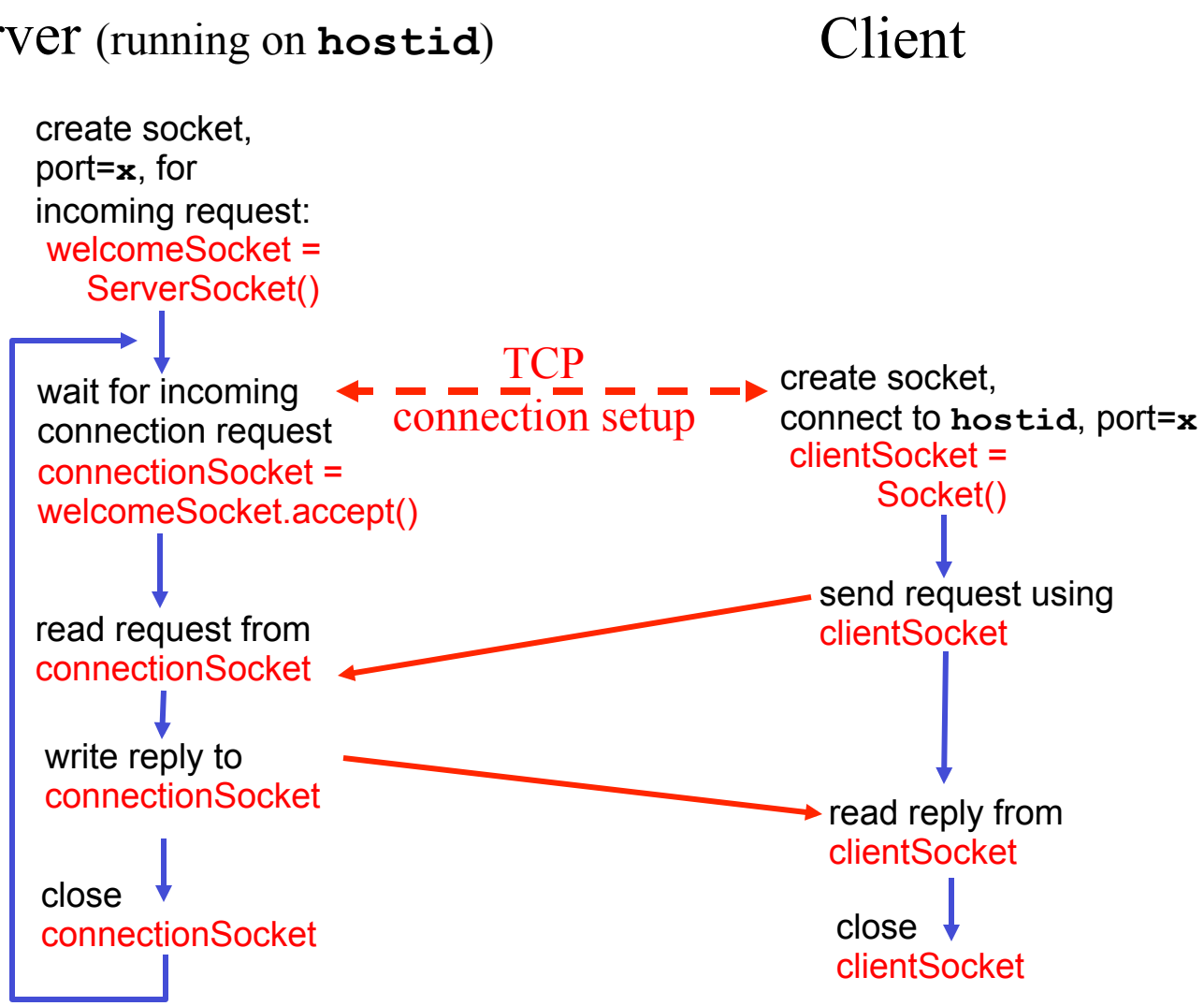
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

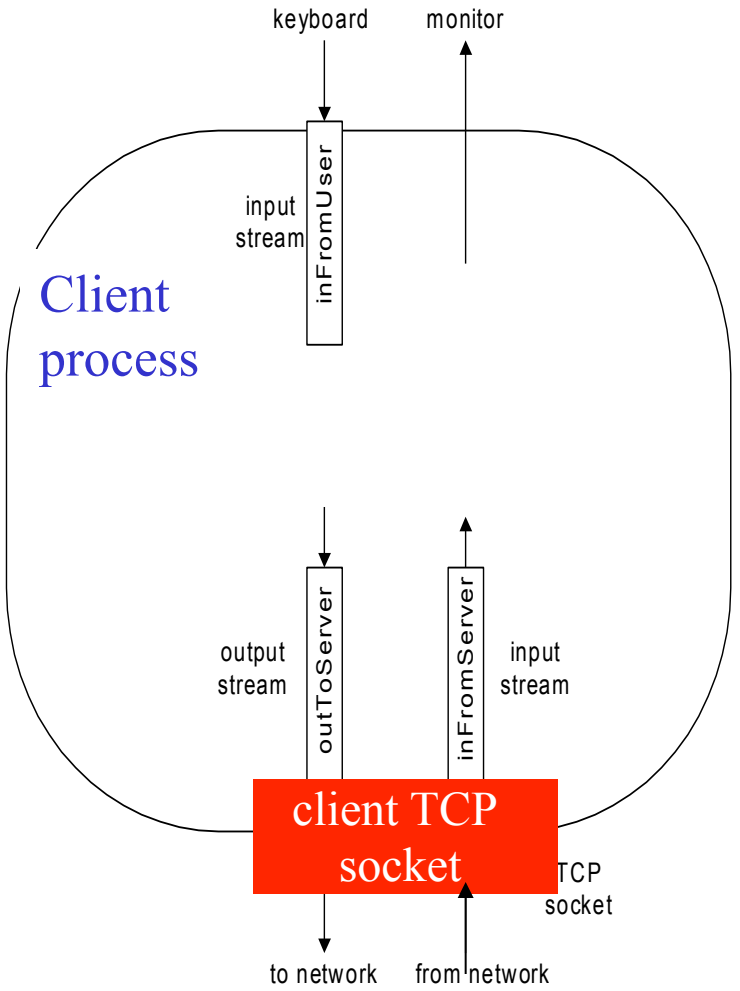
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/server socket interaction: TCP



Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



Socket programming with TCP

Example client-server app:

1) client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**`inFromServer`** stream)

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Send line to server

Read line from server

Example: Java server (TCP)

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

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

End of while loop, loop back and wait for another client connection

TCP observations & questions

- Server has two types of sockets:
  - ServerSocket and Socket
- When client knocks on serverSocket's "door," server creates connectionSocket and completes TCP conx.
- Dest IP and port are not explicitly attached to segment.
- Can multiple clients use the server?

Sockets et java

- Rappels Java
  - Rappels généraux: streams, threads
  - Sockets java

Entrées-sorties java

- Streams
  - Output streams
  - Input streams
  - Filter streams
  - Readers et writer

  - (non blocking I/O)

OuputStream

- public abstract class OutputStream
  - public abstract void write(int b) throws IOException
  - public void write(byte[] data) throws IOException
  - Public void write(byte[] data, int offset, int length) throws IOException
  - public void flush( ) throws IOException
  - public void close( ) throws IOException

InputStream

- public abstract class InputStream
  - public abstract int read( ) throws IOException
  - public int read(byte[] input) throws IOException
  - public int read(byte[] input, int offset, int length) throws IOException
  - public long skip(long n) throws IOException
  - public int available( ) throws IOException
  - public void close( ) throws IOException
  - public void mark(int readAheadLimit)
  - public void reset( ) throws IOException
  - public boolean markSupported( )

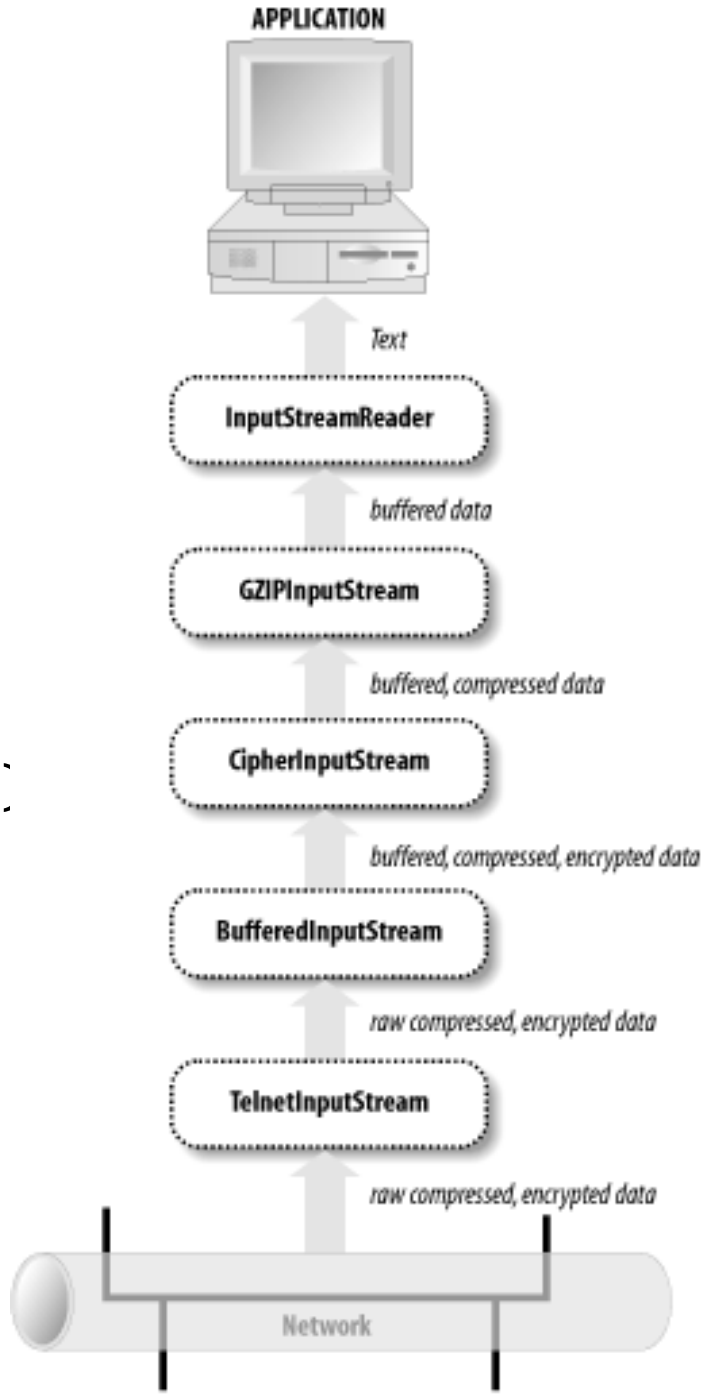
Lecture:

```
int bytesRead=0;
int bytesToRead=1024;
byte[] input = new byte[bytesToRead];
while (bytesRead < bytesToRead) {
  int result = in.read(input, bytesRead,
  bytesToRead - bytesRead);
  if (result == -1) break;
  bytesRead += result;
}
```

Filtres

- Chainage des filtres:

```
DataOutputStream dout = new
      DataOutputStream(new
        BufferedOutputStream(new
          FileOutputStream
                ("data.txt"));
```



Filtres

- Streams avec buffer
  - BufferedInputStream
  - BufferedOutputStream
- PrintStream (System.out)
- PushbackInputStream
- Streams de données (lire et écrire des données java en binaire) le codage est celui de java
  - DataInputStream
  - DataOutputStream
- Streams avec compression
- Streams avec digest
- Streams cryptées

Attention

- Une méthode comme println est dépendante de la plate-forme:
  - Le séparateur de ligne est soit \n, soit \r, soit \r\n
  - Le codage par défaut des caractères dépend de la plate-forme
  - PrintStream capte les exceptions

Compression

```
public class DeflaterOutputStream extends FilterOutputStream
public class InflaterInputStream extends FilterInputStream
public class GZIPOutputStream extends DeflaterOutputStream
public class GZIPInputStream extends InflaterInputStream
public class ZipOutputStream extends DeflaterOutputStream
public class ZipInputStream extends InflaterInputStream
```

décompresser une archive:

```
FileInputStream fin = new FileInputStream("shareware.zip");
ZipInputStream zin = new ZipInputStream(fin);
ZipEntry ze = null;
int b = 0;
while ((ze = zin.getNextEntry( )) != null) {
  FileOutputStream fout = new FileOutputStream(ze.getName( ));
  while ((b = zin.read( )) != -1) fout.write(b);
  zin.closeEntry( );
  fout.flush( );
  fout.close( );
}
zin.close( );
```

Décompresser un fichier

```
FileInputStream fin   = new
  FileInputStream("allnames.gz");
GZIPInputStream gzin  = new GZIPInputStream(fin);
FileOutputStream fout = new
  FileOutputStream("allnames");
int b = 0;
while ((b = gzin.read( )) != -1) fout.write(b);
gzin.close( );
out.flush( );
out.close( );
```

digest

- public class DigestOutputStream extends FilterOutputStream
- public class DigestInputStream extends FilterInputStream

Digest exemple:

```
MessageDigest sha = MessageDigest.getInstance("SHA");
DigestOutputStream dout = new DigestOutputStream(out,
   sha);
byte[] buffer = new byte[128];
while (true) {
  int bytesRead = in.read(buffer);
  if (bytesRead < 0) break;
  dout.write(buffer, 0, bytesRead);
}
dout.flush( );
dout.close( );
byte[] result = dout.getMessageDigest( ).digest( );
```

Cryptage décryptage

- public CipherInputStream(InputStream in, Cipher c)
- public CipherOutputStream(OutputStream out, Cipher c)
- Exemple

```
byte[] desKeyData =    "Monmotdepasse".getBytes( );
DESKeySpec desKeySpec = new DESKeySpec(desKeyData);
SecretKeyFactory keyFactory =
   SecretKeyFactory.getInstance("DES");
SecretKey desKey = keyFactory.generateSecret(desKeySpec);
Cipher des = Cipher.getInstance("DES");
des.init(Cipher.DECRYPT_MODE, desKey);
CipherInputStream cin = new CipherInputStream(fin, des);
```

Exemple

```
String infile   = "secrets.txt";
String outfile  = "secrets.des";
String password = "Un mot de passe";
try {
   FileInputStream fin = new FileInputStream(infile);
   FileOutputStream fout = new FileOutputStream(outfile);
   // register the provider that implements the algorithm
   Provider sunJce = new com.sun.crypto.provider.SunJCE( );
   Security.addProvider(sunJce);
   char[] pbeKeyData = password.toCharArray( );
   PBEKeySpec pbeKeySpec = new PBEKeySpec(pbeKeyData);
   SecretKeyFactory keyFactory =
   SecretKeyFactory.getInstance("PBEWithMD5AndDES");
   SecretKey pbeKey = keyFactory.generateSecret(pbeKeySpec);
```

Exemple suite

```
 // use Data Encryption Standard
    Cipher pbe = Cipher.getInstance("PBEWithMD5AndDES");
    pbe.init(Cipher.ENCRYPT_MODE, pbeKey);
    CipherOutputStream cout = new CipherOutputStream(fout, pbe);
    byte[] input = new byte[64];
    while (true) {
      int bytesRead = fin.read(input);
      if (bytesRead == -1) break;
      cout.write(input, 0, bytesRead);
    }
    cout.flush( );
    cout.close( );
    fin.close( );
}
catch (Exception ex) {
    System.err.println(ex);
}
```

Readers et Writers

- Hiérarchie de classe pour les caractères (avec encodage) au lieu d'octets.
- Writer et Reader classes abstraites
  - OutputStreamWriter
  - InputStreamReader
  - Filtres
    - BufferedReader, BufferedWriter
    - LineNumberReader
    - PushbackReader
    - PrintReader

Reader et Writer

- OutputStreamWriter reçoit des caractères, les convertit en octets suivant un certain codage
- `public OutputStreamWriter(OutputStream out, String encoding)  throws UnsupportedEncodingException`
- `public OutputStreamWriter(OutputStream out)`
- Exemple:

```
OutputStreamWriter w = new
          OutputStreamWriter( new
          FileOutputStream("russe.txt",
                           "Cp1251"));
```

Reader et Writer

- InputStreamReader lit des octets et les convertit suivant un certain codage
  - public InputStreamReader(InputStream in)
  - public InputStreamReader(InputStream in, String encoding) throws UnsupportedEncodingException

```
public static String getMacCyrillicString(InputStream in)
 throws IOException {
  InputStreamReader r = new InputStreamReader(in, "MacCyrillic");
  StringBuffer sb = new StringBuffer( );
  int c;
  while ((c = r.read( )) != -1) sb.append((char) c);
  r.close( );
  return sb.toString( );
}
```

Filtres

- BufferedReader
- BufferedWriter
- LineNumberReader
- PushbackReader
- PrintWriter

Threads

Threads

- threads: plusieurs activités qui coexistent et partagent des données
  - exemples:
    - pendant un chargement long faire autre chose
    - coopérer
    - processus versus threads
  - problème de l'accès aux ressources partagées
    - verrous
    - moniteur
    - synchronisation

Principes de base

- extension de la classe Thread
  - méthode run est le code qui sera exécuté.
  - la création d'un objet dont la superclasse est Thread crée la thread (mais ne la démarre pas)
  - la méthode `start` démarre la thread (et retourne immédiatement)
  - la méthode `join` permet d'attendre la fin de la thread
  - les exécutions des threads sont asynchrones et concurrentes

Exemple

```
class ThreadAffiche extends Thread{
    private String mot;
    private int delay;
    public ThreadAffiche(String w,int duree){
        mot=w;
        delay=duree;
    }
    public void run(){
        try{
            for(;;){
                System.out.println(mot);
                Thread.sleep(delay);
            }
        }catch(InterruptedException e){
        }
    }
}
```

Suite

```
public static void main(String[] args) {
      new ThreadAffiche("PING", 10).start();
      new ThreadAffiche("PONG", 30).start();
      new ThreadAffiche("Splash!",60).start();
 }
```

Alternative: Runnable

- Une autre solution:
  - créer une classe qui implémente l'interface `Runnable` (cette interface contient la méthode `run`)
  - créer une `Thread` à partir du constructeur `Thread` avec un `Runnable` comme argument.

Exemple

```
class RunnableAffiche implements Runnable{
    private String mot;
    private int delay;
    public RunnableAffiche(String w,int duree){
        mot=w;
        delay=duree;
    }
    public void run(){
        try{
            for(;;){
                System.out.println(mot);
                Thread.sleep(delay);
            }
        }catch(InterruptedException e){
        }
    }
}
```

Suite

```
  public static void main(String[] args) {
      Runnable ping=new RunnableAffiche("PING",
   10);
      Runnable pong=new RunnableAffiche("PONG",
   50);
      new Thread(ping).start();
      new Thread(pong).start();
}
```

Synchronisation

- les threads s'exécutent concurremment et peuvent accéder concurremment à des objets:
  - il faut contrôler l'accès:
    - thread un lit une variable (R1) puis modifie cette variable (W1)
    - thread deux lit la même variable (R2) puis la modifie (W2)
    - R1-R2-W2-W1
    - R1-W1-R2-W2 résultat différent!

Exemple

```
class X{
    int val;
}
class Concur extends Thread{
    X x;
    int i;
    String nom;
    public Concur(String st, X x){
        nom=st;
        this.x=x;
    }
        public void run(){
        i=x.val;
        System.out.println("thread:"+nom+" valeur x="+i);
        try{
        Thread.sleep(10);
        }catch(Exception e){}
        x.val=i+1;
        System.out.println("thread:"+nom+" valeur x="+x.val);
    }
}
```

Suite

```
public static void main(String[] args) {
        X x=new X();
        Thread un=new Concur("un",x);
        Thread deux=new Concur("deux",x);
        un.start(); deux.start();
        try{
            un.join();
            deux.join();
        }catch (InterruptedException e){}
        System.out.println("X="+x.val);
    }
```

donnera (par exemple)

- thread:un valeur x=0
- thread:deux valeur x=0
- thread:un valeur x=1
- thread:deux valeur x=1
- X=1

Deuxième exemple

```
class Y{
    int val=0;
    public int increment(){
        int tmp=val;
        tmp++;
        try{
            Thread.currentThread().sleep(100);
        }catch(Exception e){}
        val=tmp;
        return(tmp);
    }
    int getVal(){return val;}
}
class Concur1 extends Thread{
    Y y;
    String nom;
    public Concur1(String st, Y y){
        nom=st;
        this.y=y;
    }
    public void run(){
        System.out.println("thread:"+nom+" valeur="+y.increment());
    }
}
```

Suite

```
  public static void main(String[] args) {
        Y y=new Y();
        Thread un=new Concur1("un",y);
        Thread deux=new Concur1("deux",y);
        un.start(); deux.start();
        try{
            un.join();
            deux.join();
        }catch (InterruptedException e){}
        System.out.println("Y="+y.getVal());
    }
-----------
```

- thread:un valeur=1
- thread:deux valeur=1
- Y=1

Verrous

- à chaque objet est associé un verrou
  - *synchronized*(*expr*) {*instructions*}
    - *expr* doit s'évaluer comme une référence à un objet
    - verrou sur cet objet pour la durée de l'exécution de *instructions*
  - déclarer les méthodes comme synchronized: la thread obtient le verrou et le relâche quand la méthode se termine

synchronised(x)

```
class Concur extends Thread{
    X x;
    int i;
    String nom;
    public Concur(String st, X x){
        nom=st;
        this.x=x;
    }
    public void run(){
        synchronized(x){
            i=x.val;
            System.out.println("thread:"+nom+" valeur x="+i);
            try{
                Thread.sleep(10);
            }catch(Exception e){}
            x.val=i+1;
            System.out.println("thread:"+nom+" valeur x="+x.val);
        }
    }
}
```

Méthode synchronisée

```
class Y{
    int val=0;
    public synchronized int increment(){
        int tmp=val;
        tmp++;
        try{
            Thread.currentThread().sleep(100);
        }catch(Exception e){}
        val=tmp;
        return(tmp);
    }
    int getVal(){return val;}
}
------------
```

- thread:un valeur=1
- thread:deux valeur=2
- Y=2

Mais…

- la synchronisation par des verrous peut entraîner un blocage:
  - la thread un (XA) pose un verrou sur l'objet A et (YB) demande un verrou sur l'objet B
  - la thread deux (XB) pose un verrou sur l'objet B et (YA) demande un verrou sur l'objet A
  - si XA –XB : ni YA ni YB ne peuvent êter satisfaites -> blocage
- (pour une méthode synchronisée, le verrou concerne l'objet globalement et pas seulement la méthode)

Exemple

```
class Dead{
    Dead partenaire;
    String nom;
    public Dead(String st){
        nom=st;
    }
    public synchronized void f(){
        try{
            Thread.currentThread().sleep(100);
        }catch(Exception e){}
        System.out.println(Thread.currentThread().getName()+
                " de "+ nom+".f() invoque "+ partenaire.nom+".g()");
        partenaire.g();    }
    public synchronized void g(){
        System.out.println(Thread.currentThread().getName()+
                " de "+ nom+".g()");
    }
    public void setPartenaire(Dead d){
        partenaire=d;
    }
}
```

Exemple (suite)

```
        final Dead un=new Dead("un");
        final Dead deux= new Dead("deux");
        un.setPartenaire(deux);
        deux.setPartenaire(un);
        new Thread(new Runnable(){public void run(){un.f();}
        },"T1").start();
        new Thread(new Runnable(){public void run(){deux.f();}
        },"T2").start();
------------
```

- T1 de un.f() invoque deux.g()
- T2 de deux.f() invoque un.g()

Synchronisation…

- wait, notifyAll notify
  - attendre une condition / notifier le changement de condition:

```
synchronized void fairesurcondition(){
    while(!condition)
        wait();
        faire ce qu'il faut qaund la condition est vraie
        }
------------------
synchronized void changercondition(){
   … changer quelque chose concernant la condition
   notifyAll(); // ou notify()
}
```

Exemple (file: rappel Cellule)

```
public class Cellule<E>{
   private Cellule<E> suivant;
   private E element;
   public Cellule(E val) {
      this.element=val;
   }
   public Cellule(E val, Cellule suivant){
      this.element=val;
      this.suivant=suivant;
   }
   public E getElement(){
      return element;
   }
   public void setElement(E v){
      element=v;
   }
   public Cellule<E> getSuivant(){
      return suivant;
   }
   public void setSuivant(Cellule<E> s){
      this.suivant=s;
   }
```

Files synchronisées

```
class File<E>{
    protected Cellule<E> tete, queue;
    private int taille=0;

    public synchronized void enfiler(E item){
        Cellule<E> c=new Cellule<E>(item);
        if (queue==null)
            tete=c;
        else{
            queue.setSuivant(c);
        }
        c.setSuivant(null);
        queue = c;
        notifyAll();
 }
```

File (suite)

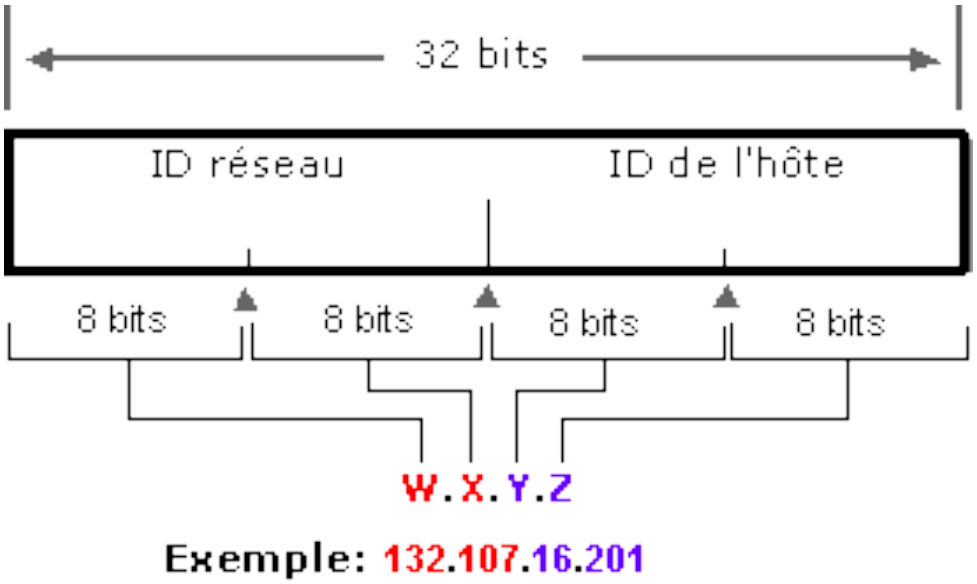
```
public synchronized E defiler() throws InterruptedException{
        while (tete == null)
            wait();
        Cellule<E> tmp=tete;
        tete=tete.getSuivant();
        if (tete == null) queue=null;
        return tmp.getElement();
}
```

Réseau et Java

- sockets

Adresses internet

- Adresse IP: adresse réseau + site sur le réseau
- Exemple:



Classe d'adresses Internet

Classe	Bits départ	Début	Fin	Notation CIDR	Masque ss-réseau
Classe A	0	0.0.0.0	127.255.255.255	/8	255.0.0.0
Classe B	10	128.0.0.0	191.255.255.255	/16	255.255.0.0
Classe C	110	192.0.0.0	223.255.255.255	/24	255.255.255.0
Classe D (mcast)	1110	224.0.0.0	239.255.255.255	/4	non défini
Classe E (réservée)	1111	240.0.0.0	255.255.255.255	/4	non défini

Classe	Nombre de réseaux possibles	Nombre d'ordinateurs maxi sur chacun
A	126	16777214
B	16384	65534
C	2097152	254

Connexion

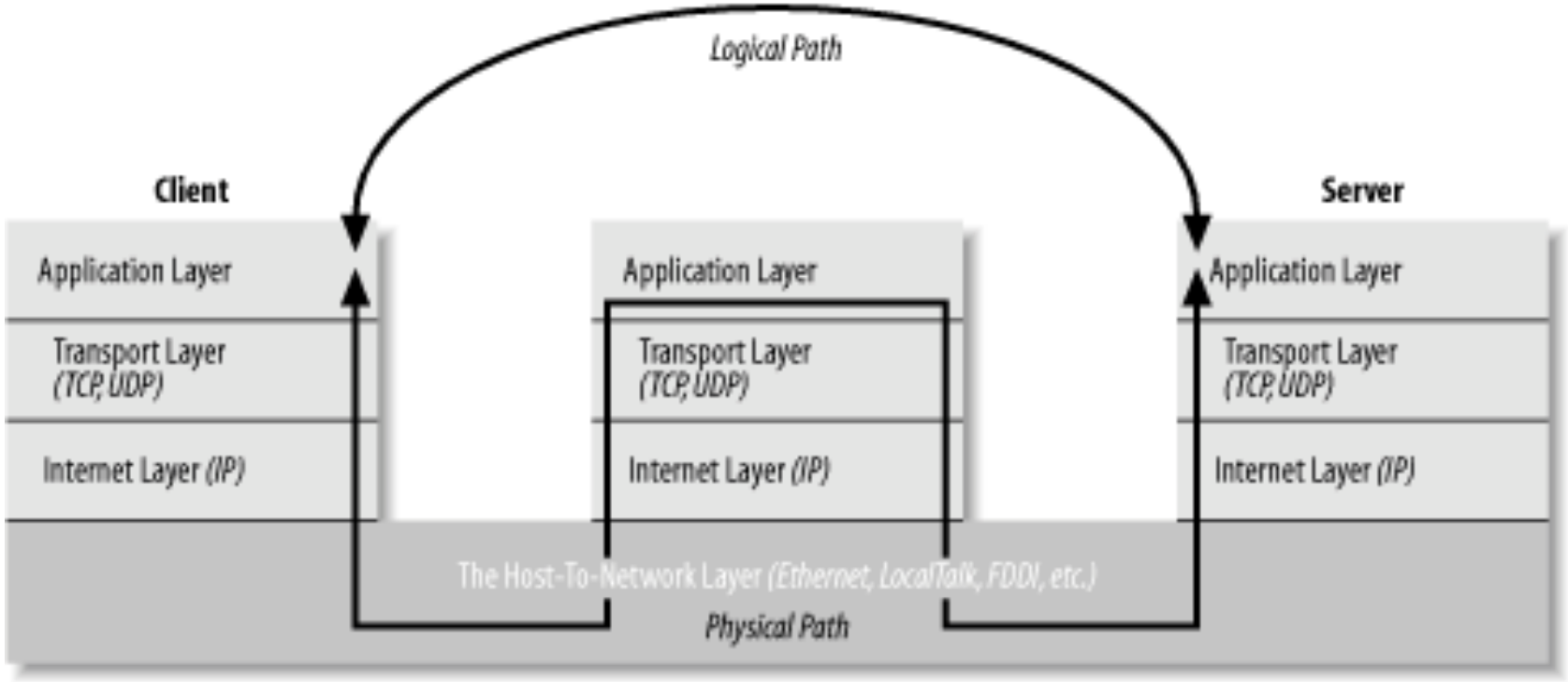
- Adresse IP +port
- Ports réservés
- Ports libres

Quelques ports

Protocol	Port	Protocol
echo	7	TCP/UDP
discard	9	TCP/UDP
daytime	13	TCP/UDP
FTP data	20	TCP
FTP	21	TCP
SSH	22	TCP
telnet	23	TCP
smtp	25	TCP
time	37	TCP/UDP

Protocol	Port	Protocol
whois	43	TCP
finger	79	TCP
HTTP	80	TCP
POP3	110	TCP
NNTP	119	TCP
IMAP	143	TCP
RMI Registry	1099	TCP

Proxys

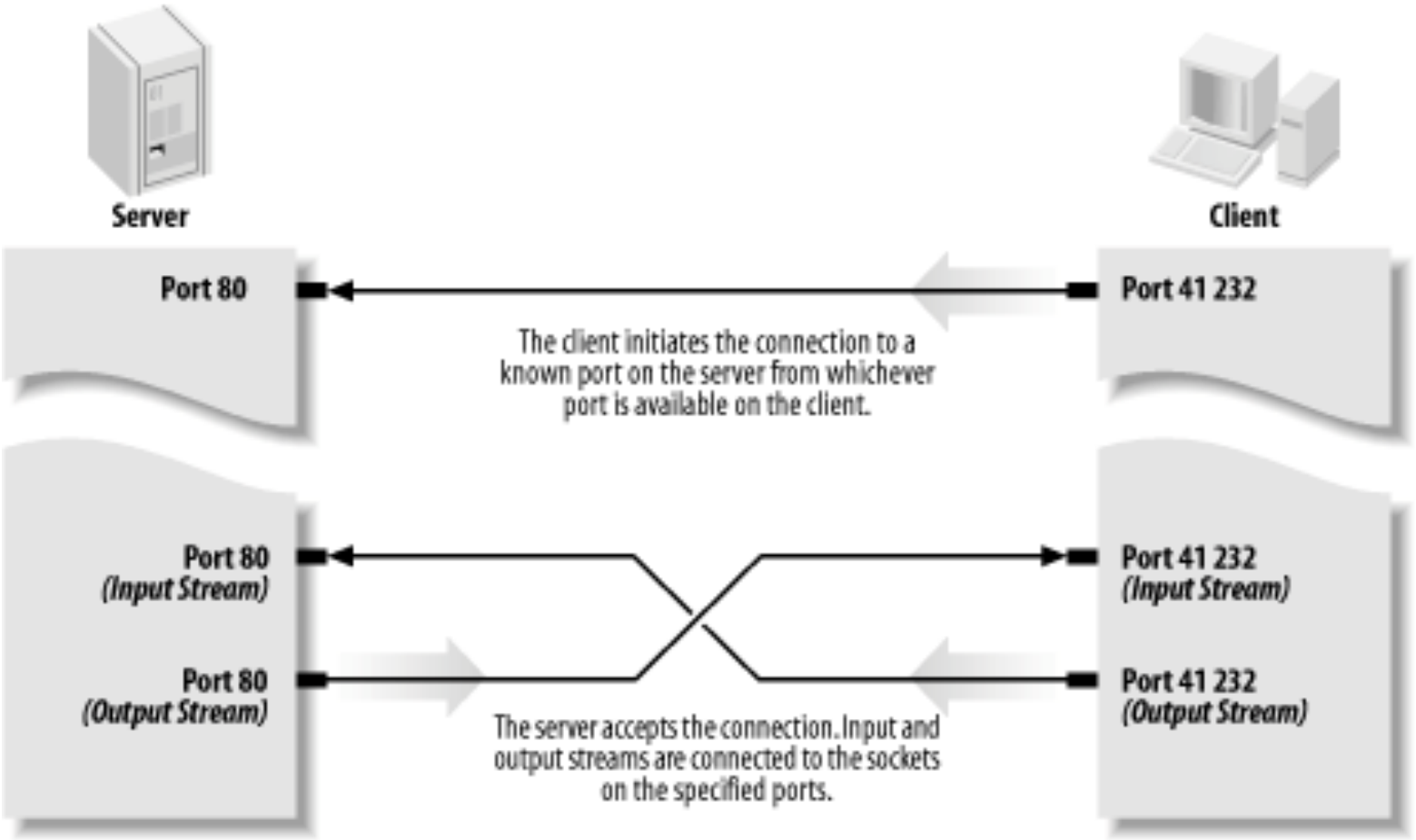

Logical Path
Client
Server
Application Layer
Transport Layer
(TCP, UDP)
Internet Layer (IP)
Application Layer
Transport Layer
(TCP, UDP)
Internet Layer (IP)
Application Layer
Transport Layer
(TCP, UDP)
Internet Layer (IP)
The Host-To-Network Layer (Ethernet, LocalTalk, FDDI, etc.)
Physical Path

Comment travailler avec un proxy?

- Régler le navigateur… les applets du navigateur utilisent ces réglages
- Pour une application java il faut préciser les propriétés: `socksProxyHo socksProxyPor` (SOCKS proxy server), `http.proxySet`, `http.proxyHost`, `http.proxyPort`, `https.proxySet`, `https.proxyHost`, `https.proxyPort`, `ftpProxySet`, `ftpProxyHost`, `ftpProxyPort`, `gopherProxySet`, `gopherProxyHost`, `gopherProxyPort` (pour les autres protocoles).
- Pour cela:

```
java -DsocksProxyHost= socks.cloud9.net -
  DsocksProxyPort= 1080 MyClass
```

Client-serveur



Classes

- java.net.InetAddress (implements java.io.Serializable)
- java.net.Inet4Address
  - java.net.Inet6Address
- java.net.DatagramPacket
- java.net.DatagramSocket
  - java.net.MulticastSocket
- java.net.ServerSocket
  - javax.net.ssl.SSLServerSocket
- java.net.Socket
  - javax.net.ssl.SSLSocket
- java.net.SocketAddress (implements java.io.Serializable)
  - java.net.InetSocketAddress

Classes

- java.net.InetAddress (implements java.io.Serializable)
- java.net.Inet4Address
  - java.net.Inet6Address
- java.net.DatagramPacket
-
- java.net.DatagramSocket
  - java.net.MulticastSocket
- java.net.ServerSocket
  - javax.net.ssl.SSLServerSocket
- java.net.Socket
  - javax.net.ssl.SSLSocket
- java.net.SocketAddress (implements java.io.Serializable)
  - java.net.InetSocketAddress

Classes

- java.net.**InetAddress** (implements java.io.Serializable)
  - java.net.**Inet4Address**
  - java.net.**Inet6Address**
- java.net.DatagramPacket
- java.net.DatagramSocket
  - java.net.MulticastSocket
- java.net.**ServerSocket**
- javax.net.ssl.**SSLServerSocket**
- java.net.**Socket**
  - javax.net.ssl.**SSLSocket**
- java.net.**SocketAddress** (implements java.io.Serializable)
  - java.net.**InetSocketAddress**

Classes

Channel (lien local)

- java.nio.channels.spi.**AbstractInterruptibleChannel** (implements java.nio.channels.Channel, java.nio.channels.InterruptibleChannel)
  - java.nio.channels.**SelectableChannel** (implements java.nio.channels.Channel)
    - java.nio.channels.spi.**AbstractSelectableChannel**
      - java.nio.channels.**DatagramChannel** (implements java.nio.channels.ByteChannel, java.nio.channels.GatheringByteChannel, java.nio.channels.ScatteringByteChannel)
      - java.nio.channels.**SocketChannel** (implements java.nio.channels.ByteChannel, java.nio.channels.GatheringByteChannel, java.nio.channels.ScatteringByteChannel)

Classes

Channel:

- java.nio.channels.spi.**AbstractInterruptibleChannel** (implements java.nio.channels.Channel, java.nio.channels.InterruptibleChannel)
  - java.nio.channels.**SelectableChannel** (implements java.nio.channels.Channel)
    - java.nio.channels.spi.**AbstractSelectableChannel**
      - java.nio.channels.**DatagramChannel** (implements java.nio.channels.ByteChannel, java.nio.channels.GatheringByteChannel, java.nio.channels.ScatteringByteChannel)
      - java.nio.channels.**ServerSocketChannel**
      - java.nio.channels.**SocketChannel** (implements java.nio.channels.ByteChannel, java.nio.channels.GatheringByteChannel, java.nio.channels.ScatteringByteChannel)
-

II) Adresses internet

- Classe `InetAddress`:
- Ontenir une `InetAddress`:
  - En utilisant le DNS
    - `public static InetAddress getByName(String hostName) throws UnknownHostException`
    - `public static InetAddress[] getAllByName(String hostName) throws UnknownHostException`
    - `public static InetAddress getLocalHost( ) throws UnknownHostException`
  - Sans DNS
    - `public static InetAddress getByAddress(byte[] address) throws UnknownHostException`
    - `public static InetAddress getByAddress(String hostName, byte[] address) throws UnknownHostException`

Exemples

```
import java.net.*;
/...
public static void main (String[] args){
   try {
       InetAddress adresse =
       InetAddress.getByName("liafa.jussieu.fr");
             System.out.println(adresse);
   } catch (UnknownHostException ex) {
       System.out.println("liafa.jussieu.fr ??");
   }
}
```

Exemples

```
public static void main (String[] args){
   try {
        InetAddress ad =
               InetAddress.getByName("192.227.93.1");
        System.out.println(ad);
   } catch (UnknownHostException ex) {
        System.out.println("192.227.93.1 ??");
   }
}
```

Toutes les adresses…

```
public static void AllAdresses(String st) {
   try {
       InetAddress[] addresses =
                     InetAddress.getAllByName(st);
       for (int i = 0; i < addresses.length; i++)
  {
                System.out.println(addresses[i]);
       }
   } catch (UnknownHostException ex) {
       System.out.println(st+"est inconnu");
   }
}
```

Mon adresse

```
public static String MonAdresse() {

  try {
     InetAddress moi = InetAddress.getLocalHost();
     return( moi.getHostAddress());
   } catch (UnknownHostException ex) {
      return("Mon adresse est inconnue");
      }

}
```

InetAddress méthodes…

```
public String getHostName( )
public byte[] getAddress( )
public String getHostAddress( )
```

Exemple:

```
public static void main (String[] args) {
 try {
   InetAddress ia= InetAddress.getByName("192.168.22.1");
   System.out.println(ia.getHostName( ));
 } catch (Exception ex)
        { System.err.println(ex); }

 }
```

Divers…

- `public boolean isAnyLocalAddress( )`
  « wildcard »?
- `public boolean isLoopbackAddress( )`
- `public boolean isMulticastAddress( )`
- Java 1.5
  - `public boolean isReachable(int timeout) throws IOException`
  - `public boolean isReachable(NetworkInterface interface, int ttl, int timeout) throws IOException`
  - IPV4 et IPV6:
    - `public final class Inet4Address extends InetAddress`
    - `public final class Inet6Address extends InetAddress`

NetworkInterface

- Exemple:

```
try {
   NetworkInterface ni =
      NetworkInterface.getByName("eth0");
   if (ni == null) {
      System.err.println(" pas de:eth0" );
   }
} catch (SocketException ex) { }
```

Exemple

```
public static String lookup(String host) {
        InetAddress node;
        // récupérer l'adresse par getByName
        try {
            node = InetAddress.getByName(host);
        } catch (UnknownHostException ex) {
            return "hôte inconnu " + host;
        }
        if (isHostname(host)) {
            return node.getHostAddress();
        } else {
            return node.getHostName();
        }
    }
```

sockets (client)

Note

- Pour l'instant on s'intéresse à des socketsTCP, on verra les sockets UDP plus tard.

Généralités

- Une connexion:
  - (IP adresse+port, IP adresse +port)
  - On peut lire et écrire sur la socket
- Serveur:
  - Associer une socket à une adresse connue (IP+port)
  - Ecoute sur la socket
  - Quand une connexion arrive accept : une nouvelle socket est créée
    - Rendre le service envoyer/recevoir
      - (en général dans une thread)
    - Continuer à écouter
- Client:
  - Crée une socket
  - Demande connexion sur adresse +port du serveur
  - Connexion
  - Envoyer/recevoir
  - Fin de la connexion

Socket en Java

- Serveur
  - Classe ServerSocket
    - (bind (mais en général par constructeur)
    - listen)
    - Accept
    - getInputStream, getOutputStream
    - close
- Client
  - Classe Socket
    - (bind)
    - connect (mais en général par constructeur)
    - getInputStream, getOutputStream
    - close

Attention!

- L'accès aux ports est souvent restreint
- Des firewall peuvent empêcher les connexions
- Il faut être root pour utiliser des ports réservés…

Côté client

- Création:
- `public Socket(InetAddress address, int port) throws IOException`
- Crée une socket + une connexion avec IP adresse et port
  - En fait:
    - Création d'une socket locale attachée à un port + une adresse locale
    - Etablissement de la connexion
    - IOException en cas d'échec

Exemple

```
public static void regarderPortBas(String host) {
  for (int i = 1; i < 1024; i++) {
     try {
          Socket s = new Socket(host, i);
          System.out.println("Il y a un serveur sur "
                 + i + " de "+ host);
           } catch (UnknownHostException ex) {
              System.err.println(ex);
              break;
           } catch (IOException ex) {
              // exception s'il n'y a pas de serveur
           }
       }
   }
```

Attention

- Cet exemple peut ne pas bien fonctionner…
  - Pour des raisons de sécurité la tentative de connexion peut être bloquante

Autres constructeurs

```
try {
  InetAddress inward =
      InetAddress.getByName("router");
  Socket socket = new Socket("mail", 25, inward,
  0);
}
catch (UnknownHostException ex) {
  System.err.println(ex);
}
```

- Connexion à partir de l'interface réseau et du port spécifié,
- '0' signifie n'importe quel port

Avec un proxy

```
SocketAddress proxyAddress = new
    InetSocketAddress("myproxy.example.com", 1080);
Proxy proxy = new Proxy(Proxy.Type.SOCKS,  proxyAddress)
Socket s = new Socket(proxy);
SocketAddress remote = new
    InetSocketAddress("login.ibiblio.org", 25);
s.connect(remote);
```

Obtenir des infos…

- public InetAddress getInetAddress( )
- public int getPort( )
- public InetAddress getLocalAddress( )
- public int getLocalPort( )

Exemple

```
public static void socketInfo(String ... args) {
        for (int i = 0; i < args.length; i++) {
            try {
                Socket theSocket = new Socket(args[i], 80);
                System.out.println("Connecté sur " +
                                              theSocket.getInetAddress()
                + "  port "  + theSocket.getPort() + " depuis port "
                       + theSocket.getLocalPort() + " de "
                       + theSocket.getLocalAddress());
            }
            catch (UnknownHostException ex) {
                System.err.println("Hôte inconnu " + args[i]);
            } catch (SocketException ex) {
                System.err.println("Connection impossible " +
   args[i]);
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
```

Communiquer…

- public InputStream getInputStream( ) throws IOException
- public OutputStream getOutputStream( ) throws IOException

Exemple: dayTime

```
public static void time(String ... hlist) {
        for (int i=0;i<hlist.length;i++){
            try {
                Socket theSocket = new Socket(hlist[i], 13);
                InputStream timeStream = theSocket.getInputStream();
                StringBuffer time = new StringBuffer();
                int c;
                while ((c = timeStream.read()) != -1) time.append((char) c);
                String timeString = time.toString().trim();
                System.out.println("Il est " + timeString + " à " +
                          hlist[i]);
            }
            catch (UnknownHostException ex) {
                System.err.println(ex);
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
```

Exemple: echo

```
public static void echo(String hostname, int port) {
       PrintWriter out = null;
       BufferedReader networkIn = null;
       try {
           Socket theSocket = new Socket(hostname, port);
           networkIn = new BufferedReader(
                   new InputStreamReader(theSocket.getInputStream()));
           BufferedReader userIn = new BufferedReader(
                   new InputStreamReader(System.in));
           out = new PrintWriter(theSocket.getOutputStream());
           System.out.println("Client: Connecté au serveur d'echo "+ theSocket);
           while (true) {
               String theLine = userIn.readLine();
               out.println(theLine);
               out.flush();
               if (theLine.equals(".")){out.close(); break;}
               System.out.println(networkIn.readLine());
           }
       }
       catch (IOException ex) {System.err.println(ex);
       } finally {
           try {
               if (networkIn != null) networkIn.close();
               if (out != null) out.close();
           } catch (IOException ex) {}
       }
   }
}
```

Echo suite

```
catch (IOException ex) {
     System.err.println(ex);
} finally {
    try {
      if (networkIn != null)
   networkIn.close();
      if (out != null) out.close();
    } catch (IOException ex) {}

}
```

Fermeture

- public void close( ) throws IOException
- Fermeture de la socket:
  - Automatique si une des parties fait un close
  - garbage collector
  - (le réseau utilise des ressources systèmes qui sont par définition partagées et limitées)
  - (a priori à mettre dans une clause finally)

En plus

- public boolean isClosed( )
- public boolean isConnected( )
- public boolean isBound( )
- public void shutdownInput( )
  throws IOException
- public void shutdownOutput( )
  throws IOException

Et aussi

- TCP_NODELAY
- SO_TIMEOUT
- SO_LINGER
- SO_SNDBUF
- SO_RCVBUF
- SO_KEEPALIVE
- OOBINLINE
- SO_REUSEADDR

ServerSocket

Principe

1. Création d'un ServerSocket par constructeur
2. Association (bind) de la socket à une adresse et un port ((1) et (2) peuvent être simultanés)
3. Écoute et connexion par accept
    1. Communication getInputStream et getOutputStream
    2. close (par le client ou le serveur ou les deux)
4. Aller en (2)
   (en général 3 est dans une thread)

Constructeurs

- `public ServerSocket(int port) throws BindException, IOException`
- `public ServerSocket(int port, int queueLength) throws BindException, IOException`
- `public ServerSocket(int port, int queueLength, InetAddress bindAddress) throws IOException`

- Ces constructeurs associent un port et une adresse au ServerSocket l'usage du port est exclusif et si le port est déjà occupé une exception est lancée
  - `public ServerSocket( ) throws IOException`

Exemple

```
public static void portsLibres() {
        for (int port = 1; port <= 65535; port++) {
            try {
                // exception si le port est utilisé
                ServerSocket server = new
                                    ServerSocket(port);
            } catch (IOException ex) {
                System.out.println("serveur sur port"
                            + port );
            }
        }
    }
```

Remarques

- port 0: choisi par le système
- on peut donner une taille sur la file des connexions en attente
- on peut choisir une adresse particulière sur la machine locale
- En java >1.4 on peut faire un "bind" explicite:
  - `public void bind(SocketAddress endpoint) throws IOException`
  - `public void bind(SocketAddress endpoint, int queueLength) throws IOException`

Exemple

```
public static void portQuelconque() {

   try {
        ServerSocket server = new
  ServerSocket(0);
        System.out.println("Le port obtenu est "
                    + server.getLocalPort());
    } catch (IOException ex) {
           System.err.println(ex);
    }
}
```

Connexion accept()

- crée et retourne une nouvelle socket pour la connexion associée (IP, port)(IP, port)

Exemple

```
 ServerSocket server = new
  ServerSocket(5776);
while (true) {
  Socket connection = server.accept( );
  OutputStreamWriter out = new
  OutputStreamWriter(
     connection.getOutputStream( ));
  out.write("Connecté:" +connection+"\r\n");
  connection.close( );
}
```

Exemple plus complet

```
public final static int DEFAULT_PORT = 13;
public static void dayTime(){
        dayTime(DEFAULT_PORT);
    }
public static void dayTime(int port) {
        if (port < 0 || port >= 65536) {
          System.out.println("Erreur port:");
          return;
        }
        try {
          ServerSocket server = new ServerSocket(port);
            Socket connection = null;
```

Exemple suite

```
while (true) {
        try {
           connection = server.accept();
           Writer out = new OutputStreamWriter(

         connection.getOutputStream());
           Date now = new Date();
           out.write(now.toString() +"\r\n");
           out.flush();
           connection.close();
         } catch (IOException ex) {} finally {
            try {
                 if (connection != null) connection.close();
            } catch (IOException ex) {}
         }
    }
 } catch (IOException ex) {
             System.err.println(ex);
    }
 }
```

Fermeture

```
public void close( ) throws IOException
```

Ferme le ServerSocket et toutes les connexions créées par accept sur la ServerSocket

Serveur echo

```
public static void serveurEcho(int port) {
        try {
           ServerSocket server = new ServerSocket(port,100);
           System.out.println("Serveur:"+server+" en écoute sur le port: "
                    + server.getLocalPort()+" est lancé");
           while (true) {
                Socket connection = server.accept();
                System.out.println("Serveur connexion avec: "
                        + connection);
                Thread echo=new EchoThread(connection);
                echo.start();
         } catch (IOException ex) {
            System.out.println("le port" + port + " est occupé");
            System.out.println("On suppose donc que le service estlancé");
        }
    }

}
```

serveur echo: EchoThread

```
class EchoThread extends Thread {
    BufferedReader in;
    PrintWriter out;
    Socket connection;
    public EchoThread(Socket connection) {
        try{
            this.connection=connection;
            InputStream in=connection.getInputStream();
            OutputStream out=connection.getOutputStream();
            this.in = new BufferedReader(new

        InputStreamReader(in));
            this.out = new PrintWriter(out);
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
```

run

```
public void run()  {
        try {
            while (true) {
                String st;
                st = in.readLine();
                if (st.equals("."))
                    in.close();
                    out.close();
                    break;
                }
                System.out.println("Serveur a reçu:"+st+" de "+connection);
                out.println(st);
                out.flush();
            }
        } catch (SocketException ex) { ex.printStackTrace();
        } catch (IOException ex) { System.err.println(ex);
        }
        try {
            in.close();
            out.close();
        } catch (IOException ex) { ex.printStackTrace();}
    }
}
```

Remarques

- utilisation des threads pour traiter le service et éviter de faire attendre les clients
- on peut aussi utiliser des entrées/sorties non bloquantes

Autres méthodes

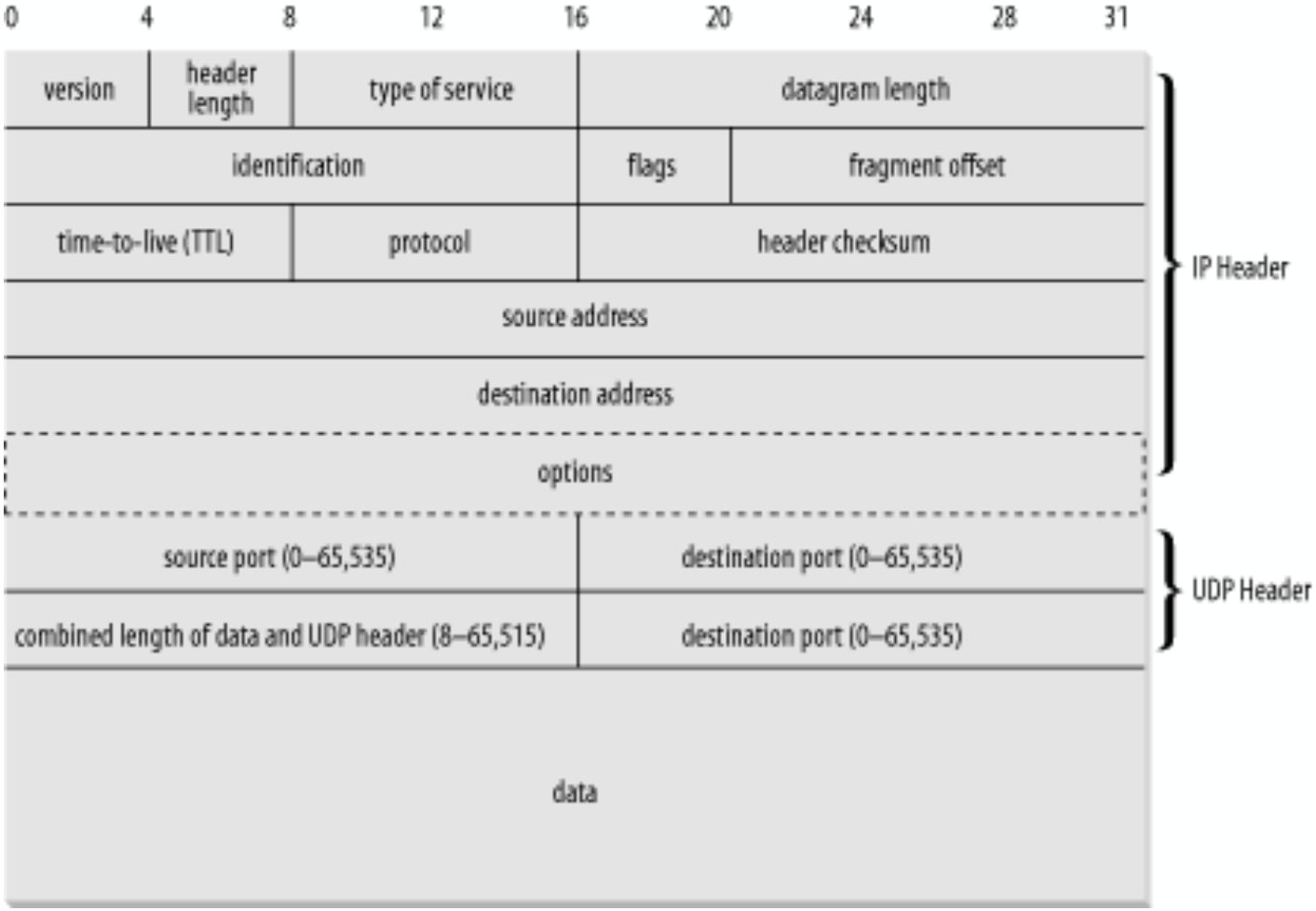
- public InetAddress getInetAddress( )
- public int getLocalPort( )

Options

- SO_TIMEOUT
- SO_REUSEADDR
- SO_RCVBUF
- public void setPerformancePreferences(int connectionTime, int latency, int bandwidth

Socket UDP

UDP


0
4
8
12
16
20
24
28
31
version
header length
type of service
datagram length
identification
flags
fragment offset
time-to-live (TTL)
protocol
header checksum
source address
destination address
options
source port (0–65,535)
destination port (0–65,535)
combined length of data and UDP header (8–65,515)
destination port (0–65,535)
data
IP Header
UDP Header

DatagramPacket

- Un paquet contient au plus 65,507 bytes
- Pour construire les paquet
  - public DatagramPacket(byte[] buffer, int length)
  - public DatagramPacket(byte[] buffer, int offset, int length)
- Pour construire et envoyer
  - public DatagramPacket(byte[] data, int length, InetAddress destination, int port)
  - public DatagramPacket(byte[] data, int offset, int length, InetAddress destination, int port)
  - public DatagramPacket(byte[] data, int length, SocketAddress destination, int port)
  - public DatagramPacket(byte[] data, int offset, int length, SocketAddress destination, int port)

Exemple

```
String s = "On essaie…";
byte[] data = s.getBytes("ASCII");

try {
  InetAddress ia =

   InetAddress.getByName("www.liafa.jussieu.fr");
  int port = 7;// existe-t-il?
  DatagramPacket dp = new DatagramPacket(data,
      data.length, ia, port);
}
catch (IOException ex)
}
```

Méthodes

- Adresses
  - public InetAddress getAddress( )
  - public int getPort( )
  - public SocketAddress getSocketAddress( )
  - public void setAddress(InetAddress remote)
  - public void setPort(int port)
  - public void setAddress(SocketAddress remote)

Méthodes (suite)

- Manipulation des données:
  - `public byte[] getData( )`
  - `public int getLength( )`
  - `public int getOffset( )`
  - `public void setData(byte[] data)`
  - `public void setData(byte[] data, int offset, int length )`
  - `public void setLength(int length)`

Exemple

```
import java.net.*;
public class DatagramExample {
  public static void main(String[] args) {
    String s = "Essayons.";
    byte[] data = s.getBytes( );
    try {
      InetAddress ia = InetAddress.getByName("www.liafa.jussieu.fr");
      int port =7;
      DatagramPacket dp = new DatagramPacket(data, data.length, ia,
   port);
      System.out.println(" Un packet pour" + dp.getAddress( ) + " port "
   +             dp.getPort( ));
      System.out.println("il y a " + dp.getLength( )  +
                 " bytes dans le  packet");
      System.out.println(
        new String(dp.getData( ), dp.getOffset( ), dp.getLength( )));
    }
    catch (UnknownHostException e) {
      System.err.println(e);
    }
  }
}
```

DatagramSocket

- Constructeurs
  - public DatagramSocket( ) throws SocketException
  - public DatagramSocket(int port) throws SocketException
  - public DatagramSocket(int port, InetAddress interface) throws SocketException
  - public DatagramSocket(SocketAddress interface) throws SocketException
  - (protected DatagramSocket(DatagramSocketImpl impl) throws SocketException)

Exemple

```
java.net.*;
public class UDPPortScanner {
  public static void main(String[] args) {

    for (int port = 1024; port <= 65535; port++) {
      try {
        // exception si utilisé
        DatagramSocket server = new DatagramSocket(port);
        server.close( );
      }
      catch (SocketException ex) {
        System.out.println("Port occupé" + port + ".");
      } // end try
    } // end for
  }
}
```

Envoyer et recevoir

- `public void send(DatagramPacket dp)`
  `throws IOException`
- `public void receive(DatagramPacket dp)`
  `throws IOException`

Un exemple: Echo

- UDPServeur
  - UDPEchoServeur
- UDPEchoClient
  - SenderThread
  - ReceiverThread

Echo: UDPServeur

```
import java.net.*;
import java.io.*;
public abstract class UDPServeur extends Thread {
  private int bufferSize;
  protected DatagramSocket sock;
  public UDPServeur(int port, int bufferSize)
   throws SocketException {
    this.bufferSize = bufferSize;
    this.sock = new DatagramSocket(port);
  }
  public UDPServeur(int port) throws SocketException {
    this(port, 8192);
  }
  public void run() {
    byte[] buffer = new byte[bufferSize];
    while (true) {
      DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
      try {
        sock.receive(incoming);
        this.respond(incoming);
      }
      catch (IOException e) {
        System.err.println(e);
      }
    } // end while
  }
 public abstract void respond(DatagramPacket request);
}
```

UDPEchoServeur

```
public class UDPEchoServeur extends UDPServeur {
    public final static int DEFAULT_PORT = 2222;
    public UDPEchoServeur() throws SocketException {
        super(DEFAULT_PORT);
    }
    public void respond(DatagramPacket packet) {
        try {
            byte[] data = new byte[packet.getLength()];
            System.arraycopy(packet.getData(), 0, data, 0, packet.getLength());
            try {
                String s = new String(data, "8859_1");
                System.out.println(packet.getAddress() + " port "
                        + packet.getPort() + " reçu " + s);
            } catch (java.io.UnsupportedEncodingException ex) {}
            DatagramPacket outgoing = new DatagramPacket(packet.getData(),
                    packet.getLength(), packet.getAddress(), packet.getPort());
            sock.send(outgoing);
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

Client: UDPEchoClient

```
public class UDPEchoClient {
  public static void lancer(String hostname, int port) {
    try {
      InetAddress ia = InetAddress.getByName(hostname);
      SenderThread sender = new SenderThread(ia, port);
      sender.start();
      Thread receiver = new ReceiverThread(sender.getSocket());
      receiver.start();
    }
    catch (UnknownHostException ex) {
      System.err.println(ex);
    }
    catch (SocketException ex) {
      System.err.println(ex);
    }

  }  // end lancer
}
```

ReceiverThread

```
class ReceiverThread extends Thread {
    DatagramSocket socket;
    private boolean stopped = false;
    public ReceiverThread(DatagramSocket ds) throws SocketException {
        this.socket = ds;
    }
    public void halt() {
        this.stopped = true;
    }
    public DatagramSocket getSocket(){
        return socket;
    }
    public void run() {
        byte[] buffer = new byte[65507];
        while (true) {
            if (stopped) return;
            DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
            try {
                socket.receive(dp);
                String s = new String(dp.getData(), 0, dp.getLength());
                System.out.println(s);
                Thread.yield();
            } catch (IOException ex) {System.err.println(ex); }
        }
    }
}
```

SenderThread

```
public class SenderThread extends Thread {
    private InetAddress server;
    private DatagramSocket socket;
    private boolean stopped = false;
    private int port;
    public SenderThread(InetAddress address, int port)
    throws SocketException {
        this.server = address;
        this.port = port;
        this.socket = new DatagramSocket();
        this.socket.connect(server, port);
    }
    public void halt() {
        this.stopped = true;
    }
//…
```

SenderThread

```
//…
    public DatagramSocket getSocket() {
        return this.socket;
    }
    public void run() {

        try {
            BufferedReader userInput = new BufferedReader(new
    InputStreamReader(System.in));
            while (true) {
                if (stopped) return;
                String theLine = userInput.readLine();
                if (theLine.equals(".")) break;
                byte[] data = theLine.getBytes();
                DatagramPacket output
                        = new DatagramPacket(data, data.length, server, port);
                socket.send(output);
                Thread.yield();
            }
        }  // end try
        catch (IOException ex) {System.err.println(ex); }
    }  // end run
}
```

Autres méthodes

- `public void close( )`
- `public int getLocalPort( )`
- `public InetAddress getLocalAddress( )`
- `public SocketAddress getLocalSocketAddress( )`
- `public void connect(InetAddress host, int port)`
- `public void disconnect( )`
- `public void disconnect( )`
- `public int getPort( )`
- `public InetAddress getInetAddress( )`
- `public InetAddress getRemoteSocketAddress( )`

Options

- SO_TIMEOUT
  - public synchronized void setSoTimeout(int timeout) throws SocketException
  - public synchronized int getSoTimeout( ) throws IOException
- SO_RCVBUF
  - public void setReceiveBufferSize(int size) throws SocketException
  - public int getReceiveBufferSize( ) throws SocketException
- SO_SNDBUF
  - public void setSendBufferSize(int size) throws SocketException
  - int getSendBufferSize( ) throws SocketException
- SO_REUSEADDR (plusieurs sockets sur la même adresse)
  - public void setReuseAddress(boolean on) throws SocketException
  - boolean getReuseAddress( ) throws SocketException
- SO_BROADCAST
  - public void setBroadcast(boolean on) throws SocketException
  - public boolean getBroadcast( ) throws SocketException