Protocoles internet

rappel java

Rappels Java

- Entrées-sorties
- Thread
- Adresse Internet
- Socket TCP
- Socket UDP

Entrées-sorties java

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

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;
}
```

Fichier et filtres (quelques)

- OutputSream (abstract)
  - FileOutputStream
  - FilterOutputStream
    - BufferedOutputStream, Stream avec buffer
    - DataOutputStream, Stream de données, le codage est celui du java
    - CipherOutputStream, Stream avec cryptage
    - DigestOutputStream, Stream avec résumé
    - DeflaterOutputStream, Stream pour la compression
      - GZIPOutputStream, ZipOutputStream
    - InflaterOutputStream, Stream pour la décompression
    - PrintStream (System.out, System.err)

```
OutputStream outputStream2       = new
FileOutputStream("output2.txt");
      BufferedOutputStream      osw2 = new
BufferedOutputStream(outputStream2);
      String str = « du texte « ;
     byte[] byteArr = str.getBytes();
      osw2.write(byteArr,0,byteArr.length);
      osw2.flush();
      osw2.close();
```

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

Fichier et filtres (quelques)

- InputStream (abstract)
  - FileInputStream
  - FilterInputStream
    - BufferedInputStream, Stream avec buffer
    - DataInputStream, Stream de données, le codage est celui du java
    - CipherInputStream, Stream avec cryptage
    - DigestInputStream, Stream avec résumé
    - DeflaterInputStream, Stream pour la compression
    - InflaterInputStream, Stream pour la décompression
      - GZIPOutputStream, ZipOutputStream

Readers et Writers

- Hiérarchie de classe pour les caractères (avec encodage) au lieu d'octets.
- Writer et Reader classes abstraites
  - OutputStreamWriter
    - Filewriter
  - InputStreamReader
    - FileReader
  - BufferedReader, BufferedWriter, PrintWriter

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");
```

```
 OutputStream outputStream = new
FileOutputStream("output.txt");
 Writer osw = new OutputStreamWriter(outputStream);
 osw.write("sera bien écrit dans le fichier");
 osw.close();
```

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
- PrintWriter

```
FileWriter fw = new FileWriter("output.txt");
Writer       bw = new BufferedWriter(fw);
bw.write("sera bien écrit dans le fichier");
bw.close();
```

```
FileWriter fw1 = new FileWriter("output5.txt");
PrintWriter       bw1 = new PrintWriter(fw1);
bw1.println("sera bien écrit dans le fichier »);
bw1.println( (int) 123); bw1.println( (float) 1.23);
bw1.close();
```

```
sera bien écrit dans le fichier
123
1.23
```

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();
}
```

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(); x.val=0;
      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. Si un verrou est déjà posé, elle est mise en attente.

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);
        }
    }
}
```

- thread:un valeur x=0
- thread:un valeur x=1
- thread:deux valeur x=1
- thread:deux valeur x=2
- X=2

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 être satisfaites -> blocage
- (pour une méthode synchronisée, le verrou concerne l'objet globalement et pas seulement la méthode)

Synchronisation…

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

```
synchronized void fairesurcondition(){
        while(!condition)
                wait();
                faire ce qu'il faut quand 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

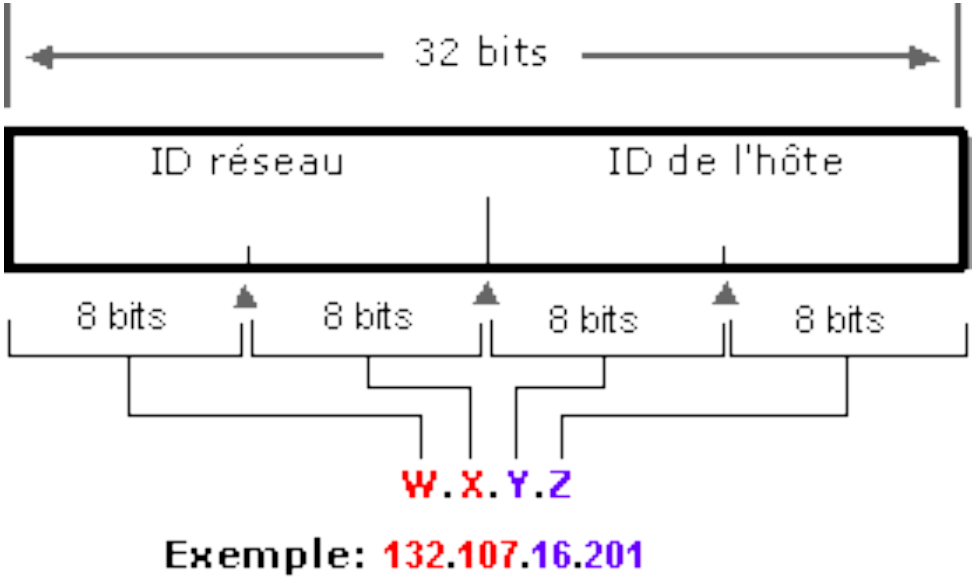
```
class File<E>{
    protected Cellule<E> tete, queue;
    private int taille=0;

    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();
}
```

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

Classes

- java.net.**InetAddress** (implements java.io.Serializable)
- java.net.**Inet4Address**
  - java.net.**Inet6Address**

Obtenir 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(« www.irif.univ-paris-diderot.fr");
            System.out.println(adresse);
    } catch (UnknownHostException ex) {    System.out.println("liafa.univ-
    paris-diderot.fr ??");
    }

}
```

— —

www.irif.univ-paris-diderot.fr/81.194.27.171

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 ??");
    }
}
```

——

/192.227.93.1

—-avec 192.9.1

/192.9.0.1

—avec 19x.227.93.1

192.227.93.1 ??

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("81.194.27.171");
      System.out.println(ia.getHostName( )); System.out.println(ia.getHostAddress( ));
 } catch (Exception ex)
                { System.err.println(ex); }

 }
```

———

www.irif.univ-paris-diderot.fr

81.194.27.171

Divers…

- public boolean isAnyLocalAddress( )
  « wildcard »?
- public boolean isLoopbackAddress( )
- public boolean isMulticastAddress( )
- public boolean isReachable(int timeout) throws IOException

- IPV4 et IPV6:
  - public final class Inet4Address extends InetAddress
  - public final class Inet6Address extends InetAddress

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
IMAP (v3)	220	TCP/UDP

Attention!

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

Réseau et Java

- sockets

Classes

- java.net.**DatagramPacket**
- java.net.**DatagramSocket**
  - java.net.**MulticastSocket**
- java.net.**ServerSocket**
  - javax.net.ssl.**SSLServerSocket**
- java.net.**Socket**
  - javax.net.ssl.**SSLSocket**
- Classe abstraite:

  java.net.**SocketAddress** (implements java.io.Serializable)
  - java.net.**InetSocketAddress**

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 TCP 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

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

Obtenir des infos…

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

Exemple

```
        try {
            Socket theSocket = new Socket(InetAddress.getByName("81.194.27.171"), 443);
            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 ");
        } catch (SocketException ex) {
            System.err.println("Connection impossible ");
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

Connecté sur /81.194.27.171 port 443 depuis port 53828 de /192.168.1.99

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

    try {
        Socket theSocket = new Socket(hostname, port);
        System.out.println("Client: Connecté au serveur d'echo "+ theSocket);

        networkIn = new BufferedReader( new InputStreamReader(theSocket.getInputStream()));
        out = new PrintWriter(theSocket.getOutputStream());

        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_SNDBUF
- SO_RCVBUF
- SO_KEEPALIVE
- OOBINLINE
- SO_REUSEADDR

Côté serveur

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

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