Page de Jean-Michel Richer

Maître de Conférences en Informatique

<<<M1 - UE2

2 Communication par Socket

Note : on pourra se référer à l'API JavaSE 7 pour plus de détails.

Note :

1. Représentation et traitement des adresses IP

La classe `InetAddress` permet de représenter des adresses Internet. Elle comporte deux champs :

- `address`
- `hostname`

Parmi les méthodes les plus utiles on trouve :

- `byte[] getAddress()`
- `String getHostAddress()`
- `String getHostName()`
- `static InetAddress getByName(String host)`
- `static InetAddress[] getAllByName(String host)`
- `static InetAddress getLocalHost()`

API InetAdress

Il existe également `Inet4Address` et `Inet6Address`.

1.1. Exemple 1 : getByName

On cherche à obtenir l'adresse IP de différents hôtes :

```
1.  import java.net.*;
2.
3.  public class Code1 {
4.    public static void main(String args[]) {
5.      String hostName1 = "yahoo.fr";
6.      String hostName2 = "172.20.41.8";
7.      String hostName3 = "www.xtg9yhfr.fr";
8.
9.      try {
10.       InetAddress ad1 = InetAddress.getByName(hostName1);
11.       System.out.println(ad1 + " # " + ad1.getHostName());
12.
13.       InetAddress ad2 = InetAddress.getByName(hostName2);
14.       System.out.println(ad2 + " # " + ad2.getHostName());
15.
16.       InetAddress ad3 = InetAddress.getByName(hostName3);
17.       System.out.println(ad3 + " # " + ad3.getHostName());
18.
19.     } catch(UnknownHostException exc1) {
20.       System.err.println("Error: " + exc1);
21.     } catch(Exception exc2) {
22.       exc2.printStackTrace(System.err);
23.     }
24.   }
25. }
26.
```

Le résultat en sortie est par exemple :

```
yahoo.fr/87.248.120.148 # yahoo.fr
/172.20.41.8 # janus.info-ua
Error: java.net.UnknownHostException: www.xtg9yhfr.fr
```

1.2. Exemple 2 : getAllByName

On cherche à obtenir l'ensemble des adresses IP de www.yahoo.com

```
1.  import java.net.*;
2.
3.  public class Code2 {
4.    public static void main(String args[]) {
5.      String hostName1 = "www.yahoo.com";
6.
7.      try {
8.        InetAddress tAddresses[] = InetAddress.getAllByName(hostName1);
9.        for (InetAddress ad : tAddresses) {
10.         System.out.println(ad + " # " + ad.getHostName());
11.       }
12.
13.     } catch(UnknownHostException exc1) {
14.       System.err.println("Error: " + exc1);
15.     } catch(Exception exc2) {
16.       exc2.printStackTrace(System.err);
17.     }
18.   }
19. }
20.
```

Le résultat en sortie est par exemple :

```
www.yahoo.com/87.248.122.122 # www.yahoo.com
www.yahoo.com/87.248.112.181 # www.yahoo.com
www.yahoo.com/2a00:1288:f006:1fe:0:0:0:3000 # www.yahoo.com
www.yahoo.com/2a00:1288:f00e:1fe:0:0:0:3000 # www.yahoo.com
www.yahoo.com/2a00:1288:f00e:1fe:0:0:0:3001 # www.yahoo.com
www.yahoo.com/2a00:1288:f006:1fe:0:0:0:3001 # www.yahoo.com
```

1.3. Exemple 3 : getLocalHost

On cherche à obtenir l'adresse IP de la machine sur laquelle on opère :

```java
import java.net.*;

public class Code3 {
  public static void main(String args[]) {

    try {
      InetAddress ad = InetAddress.getLocalHost();
      System.out.println(ad);

    } catch(Exception exc2) {
      exc2.printStackTrace(System.err);
    }
  }
}

```

Le résultat en sortie est par exemple :

```
richer/172.20.41.171
```

1.4. Exemple 4 : byte[] getAddress

On cherche à obtenir l'adresse IP de la machine sur laquelle on opère :

```
1.  import java.net.*;
2.
3.  public class Code4 {
4.    public static void main(String args[]) {
5.
6.      try {
7.        InetAddress ad = InetAddress.getLocalHost();
8.        byte bytes[] = ad.getAddress();
9.        for (int i = 0; i < bytes.length; ++i) {
10.         byte b = bytes[i];
11.         int unsigned = (b < 0) ? b + 256 : b;
12.         System.out.print(unsigned);
13.         if (i < (bytes.length - 1)) System.out.print(".");
14.       }
15.       System.out.println();
16.
17.     } catch(Exception exc) {
18.       exc.printStackTrace(System.err);
19.     }
20.   }
21. }
22.
```

Le résultat en sortie est par exemple :

```
172.20.41.171
```

2. Gestion des URL

La classe URL permet de représenter un *Unified Resource Locator*, c'est à dire un lien sur une ressource internet. On utilise une classe (plutôt qu'une chaîne) afin d'identifier les différents éléments qui composent l'URL. Il existe différents constructeurs :

- `URL(String spec)`
- `URL(String protocol, String host, int port, String file)`
- `URL(String protocol, String host, int port, String file, URLStreamHandler handler)` : on spécifie un gestionnaire de protocole
- `URL(URL context, String spec)`

Les méthodes les plus utilisées sont :

- `String getHost()`
- `String getPath()`
- `int getPort()`
- `URLConnection openConnection()`
- `InputStream openStream()`, équivalent à `openConnection().getInputStream()`

2.1. créer une URL

```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  public class URLCreate {
5.      public static void main(String[] args) {
6.
7.      try {
8.          URL url = new URL("http://www.info.univ-
    angers.fr:80/membre_fiche.php?us_id=23");
9.          System.out.println("protocol: " + url.getProtocol());
10.         System.out.println("host: " + url.getHost());
11.         System.out.println("port: " + url.getPort());
```

```
12.         System.out.println("path: " + url.getPath());
13.         System.out.println("query: " + url.getQuery());
14.         System.out.println("file: " + url.getFile());
15.
16.       } catch(MalformedURLException e) {
17.         System.err.println(e);
18.       }
19.     }
20. }
21.
```

```
protocol: http
host: www.info.univ-angers.fr
port: 80
path: /membre_fiche.php
query: us_id=23
file: /membre_fiche.php?us_id=23
```

2.2. lire depuis une URL

```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  public class URLReader {
5.     public static void main(String[] args) throws Exception {
6.
7.       URL url = new URL("http://www.info.univ-
    angers.fr:80/membre_fiche.php?us_id=14");
8.       BufferedReader in = new BufferedReader(
9.      new InputStreamReader(url.openStream())
10.    );
```

```
11.
12.       String inputLine;
13.       while ((inputLine = in.readLine()) != null) {
14.          System.out.println(inputLine);
15.       }
16.       in.close();
17.    }
18. }
19.
```

2.3. lire une image depuis une URL

```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  public class URLContent {
5.     public static void main(String[] args) throws Exception {
6.     Object obj = null;
7.
8.        URL url = new URL("http://www.info.univ-
    angers.fr:80/img/logo_dpt_info.png");
9.        obj = url.getContent();
10.    System.err.println(obj.getClass().getName());
11.    }
12. }
13.
```

```
sun.awt.image.URLImageSource
```

2.4. ecrire vers une URL

On utilise la classe `URLConnection` obtenue depuis `openConnection()`.

```
1.  import java.io.*;
2.  import java.net.*;
3.
4.  public class URLWriter {
5.     public static void main(String[] args) throws Exception {
6.
7.        String name = URLEncoder.encode("Jean-Michel Richer", "UTF-8");
8.
9.        URL url = new URL("http://www.info.univ-angers.fr/"+
10.         "pub/richer/ens/m1/info/ue2/post_parameters.php");
11.       URLConnection connection = url.openConnection();
12.       connection.setDoOutput(true);
13.
14.       OutputStreamWriter out = new OutputStreamWriter(
15.         connection.getOutputStream());
16.       out.write("name=" + name);
17.       out.close();
18.
19.       BufferedReader in = new BufferedReader(
20.      new InputStreamReader(
21.            connection.getInputStream()
22.          )
23.       );
24.       String decodedString;
25.       while ((decodedString = in.readLine()) != null) {
26.          System.out.println(decodedString);
27.       }
```

```
28.         in.close();
29.     }
30. }
31.
```

```
POST: Hello Jean-Michel Richer
```

2.5. URLEncoder et URLDecoder

Les classes `URLEncoder` et `URLDecoder` ont pour but de coder et décoder les URL. Ces classes possèdent des méthodes statiques :

- `static String URLEncoder.encode(String s)`
- `static String URLEncoder.encode(String s, String enc)` avec description de l'encodage
- `static String URLDecoder.decode(String s)`
- `static String URLDecoder.decode(String s, String enc)`

2.6. URLConnection

Il s'agit d'une class abstraite qui permet d'ouvrir une connexion associée à une URL. Plusieurs méthodes lui sont associées.

```
1. import java.net.*;
2. import java.io.*;
3.
4. public class URLConnectionRetriever {
5.
6.   public static void main(String args[]) {
7.     if (args.length == 0) {
8.       System.err.println("require one argument");
9.       System.exit(0);
10.    }
11.
```

```
12.    try {
13.      URL url = new URL(args[0]);
14.      URLConnection uc = url.openConnection();
15.
16.      System.out.println("Content type: "+uc.getContentType());
17.      System.out.println("Content encoding: "+uc.getContentEncoding());
18.      System.out.println("Content length: "+uc.getContentLength());
19.      System.out.println("Date: "+uc.getDate());
20.
21.      System.out.println(uc.getHeaderField("content-type"));
22.      System.out.println(uc.getHeaderField("content-encoding"));
23.
24.      System.out.println(uc.guessContentTypeFromName(".tgz"));
25.    } catch(Exception e) {
26.      System.err.println("Exception: "+e);
27.    }
28.
29.  }
30. }
31.
```

```
java URLConnectionRetriever http://www.info.univ-angers.fr
Content type: text/html
Content encoding: null
Content length: 6186
Date: 1350615476000
text/html
null
null
```

On peut également utiliser cette classe pour envoyer des données à un formulaire :

```

```

```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  public class URLPostForm {
5.
6.    public static void main(String args[]) {
7.     try {
8.       URL url = new URL(args[0]);
9.       URLPostForm.submitData(url);
10.
11.    } catch(Exception e) {
12.      System.out.println("Error: "+e);
13.    }
14.
15.   }
16.
17.   public static void submitData(URL url) throws Exception {
18.    String query = "first_name=" +
19.      URLEncoder.encode("Jean-Michel", "UTF-8")
20.      + "&last_name=" +
21.      URLEncoder.encode("Richer", "UTF-8");
22.
23.    int qLength = query.length();
24.
25.    URLConnection uc = url.openConnection();
26.    uc.setDoOutput(true);
27.    uc.setDoInput(true);
28.    uc.setAllowUserInteraction(false);
29.    DataOutputStream dos = new
   DataOutputStream(uc.getOutputStream());
30.    dos.writeBytes(query);
```

```
31.     dos.close();
32.
33.     BufferedReader br = new BufferedReader(new
    InputStreamReader(new DataInputStream(uc.getInputStream())));
34.     String line;
35.
36.     while ((line = br.readLine()) != null) {
37.       System.out.println(line);
38.     }
39.
40.
41.   }
42. }
43.
```

```
java URLPostForm http://localhost/pub/richer/person_form.php
<html>
<head>
<title>person form</title>
</head>
<body>
<p>Hello Jean-Michel Richer</p></body>
</html>
```

3. Sockets

Un socket est une des extrémités d'un canal de communication entre deux interlocuteurs dans un dialogue de type client / serveur. On dispose donc de deux classes pour représenter le client et le serveur :

- `Socket` pour le client
- `ServerSocket` pour le serveur

3.1. la partie client : socket

Il s'agit de la partie la plus simple, on se connecte au serveur en spécifiant son adresse, puis on lit les messages envoyés par le serveur, ou on écrit au serveur.

```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  class SocketClient {
5.    public static void main(String args[]) {
6.     PrintWriter out = null;
7.        BufferedReader in = null;
8.
9.     try {
10.      Socket s = new Socket("localhost", 30970);
11.      out = new PrintWriter(s.getOutputStream(), true);
12.      in = new BufferedReader(new InputStreamReader(
13.        s.getInputStream())
14.      );
15.
16.      out.println("hello server");
17.      System.out.println("server reply> " + in.readLine());
18.      Thread.sleep(5000);
19.      out.println("bye");
20.
21.      out.close();
22.      in.close();
23.      s.close();
24.
25.    } catch(Exception e) {
26.      System.err.println("Client: "+e);
27.    }
28.  }
```

```
}

```

3.2. la partie serveur : ServerSocket

Le serveur ouvre un port de communication et attend qu'un client se connecte.

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

class SocketServer {
  public static void main(String args[]) {
    PrintWriter out = null;
        BufferedReader in = null;

    try {
      ServerSocket server = new ServerSocket(30970);

      Socket client = server.accept();

      out = new PrintWriter(client.getOutputStream(), true);
          in = new BufferedReader(new InputStreamReader(
            client.getInputStream())
          );

      String input = in.readLine();
        System.out.println("client send> " + input);
      out.println("hello client");

      out.close();
      in.close();
```

```
      client.close();

      server.close();

    } catch(Exception e) {
      System.err.println(e);
    }
  }
}

```

Le serveur précédent n'est pas très intéressant car il n'accepte qu'un client, on peut mettre en place un serveur qui répond à plusieurs clients, en utilisant une boucle while :

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

class MultiSocketServer {
  public static void main(String args[]) {
    PrintWriter out = null;
        BufferedReader in = null;

    try {
      ServerSocket server = new ServerSocket(30970);

      while (true) {
        Socket client = server.accept();

        out = new PrintWriter(client.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
```

```
            client.getInputStream())
          );

      String line;
      while (true) {
        line = in.readLine();
        if (line.equals("bye")) break;
          System.out.println("client send> " + line);
        out.println("hello client");
      }

      out.close();
      in.close();
      client.close();
    }

    //server.close();

  } catch(Exception e) {
    System.err.println(e);
  }
 }
}

```

Le serveur multi-client est plus intéressant mais souffre d'un défaut : les requêtes des clients sont traitées l'une après l'autre : si le traitement prend du temps, alors les autres clients doivent attendre. Pour résoudre ce problème on crée une classe qui sera un thread et qui se charge du dialogue avec le client, ainsi on peut prendre en charge rapidement un autre client.

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

```
2.  import java.io.*;
3.
4.  class MultiThreadSocketServer {
5.    public static void main(String args[]) {
6.
7.      try {
8.        ServerSocket server = new ServerSocket(30970);
9.
10.       while (true) {
11.         Socket client = server.accept();
12.
13.         ClientDialogThread t = new ClientDialogThread(client);
14.         t.start();
15.       }
16.
17.       //server.close();
18.
19.     } catch(Exception e) {
20.       System.err.println(e);
21.     }
22.   }
23. }
24.
```

```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  public class ClientDialogThread extends Thread {
5.
6.    Socket socket;
```

```
7.   PrintWriter out = null;
8.    BufferedReader in = null;
9.
10.  public ClientDialogThread(Socket s) {
11.   socket = s;
12.  }
13.
14.  public void run() {
15.   try {
16.    out = new PrintWriter(socket.getOutputStream(), true);
17.       in = new BufferedReader(new InputStreamReader(
18.        socket.getInputStream())
19.       );
20.
21.       String line;
22.       while (true) {
23.        line = in.readLine();
24.          System.out.println("received> "+line);
25.          out.println("ack");
26.        if (line.equals("bye")) break;
27.       }
28.
29.       in.close();
30.       out.close();
31.       socket.close();
32.
33.   } catch(Exception e) {
34.    System.err.println(e);
35.   }
36.  }
37. }
```

```
38.
```

4. Datagrammes UDP

Les sockets vus précédemment utilisent TCP, un protocole fiable qui réexpédie les données perdues et réarrange les trames de manière à obtenir l'ordre initial de la transmission.

UDP (User Datagram Protocol) est quant à lui un protocole de transfert rapide mais non fiable.

Java implante UDP grâce à deux classes :

- `DatagramSocket` envoi et réception de datagrammes
- `DatagramPacket` représente les paquets de données à envoyer ou recevoir

Contrairement aux Sockets, c'est le paquet qui contient les informations de connexion.

4.1. la classe DatagramPacket

Celle-ci prend deux formes lors de la construction en fonction que l'on envoie ou reçoit les données :

- `DatagramPacket(byte buffer[], int length, InetAddress ia, int port)` : permet de coder les données à envoyer
- `DatagramPacket(byte buffer[], int length)` : permet de définir le tampon qui reçoit les données ainsi que sa longueur

4.2. la classe DatagramSocket

Même principe que le DatagramPacket :

- `DatagramSocket()` : permet de définir un socket pour l'envoi
- `DatagramSocket(int port)` : permet de définir un socket de réception sur le port indiqué

4.3. Exemple

Voici un exemple de client / serveur en UDP où le serveur renvoie le message reçu en majuscule.

4.3.1 le client

```
1. import java.net.*;
```

```java
import java.io.*;

public class UDPClient {

  public static void main(String args[]) {
    String host = "localhost";
    int port = 7030;
    String message;
    int n = 1;

    try {
      DatagramPacket packet;
      DatagramSocket clientSocket = new DatagramSocket();
      InetAddress ipServer = InetAddress.getByName(host);

      while (true) {
        message = "ligne " + n;
        byte data[] = message.getBytes();

        System.out.println("send: " + message);
        packet = new DatagramPacket(data, data.length, ipServer, port);
        clientSocket.send(packet);

        Thread.sleep(1000);
        ++n;
      }
    } catch(Exception e) {
      System.err.println(e);
    }
  }
}
```

```
33.
```

4.3.2 le serveur

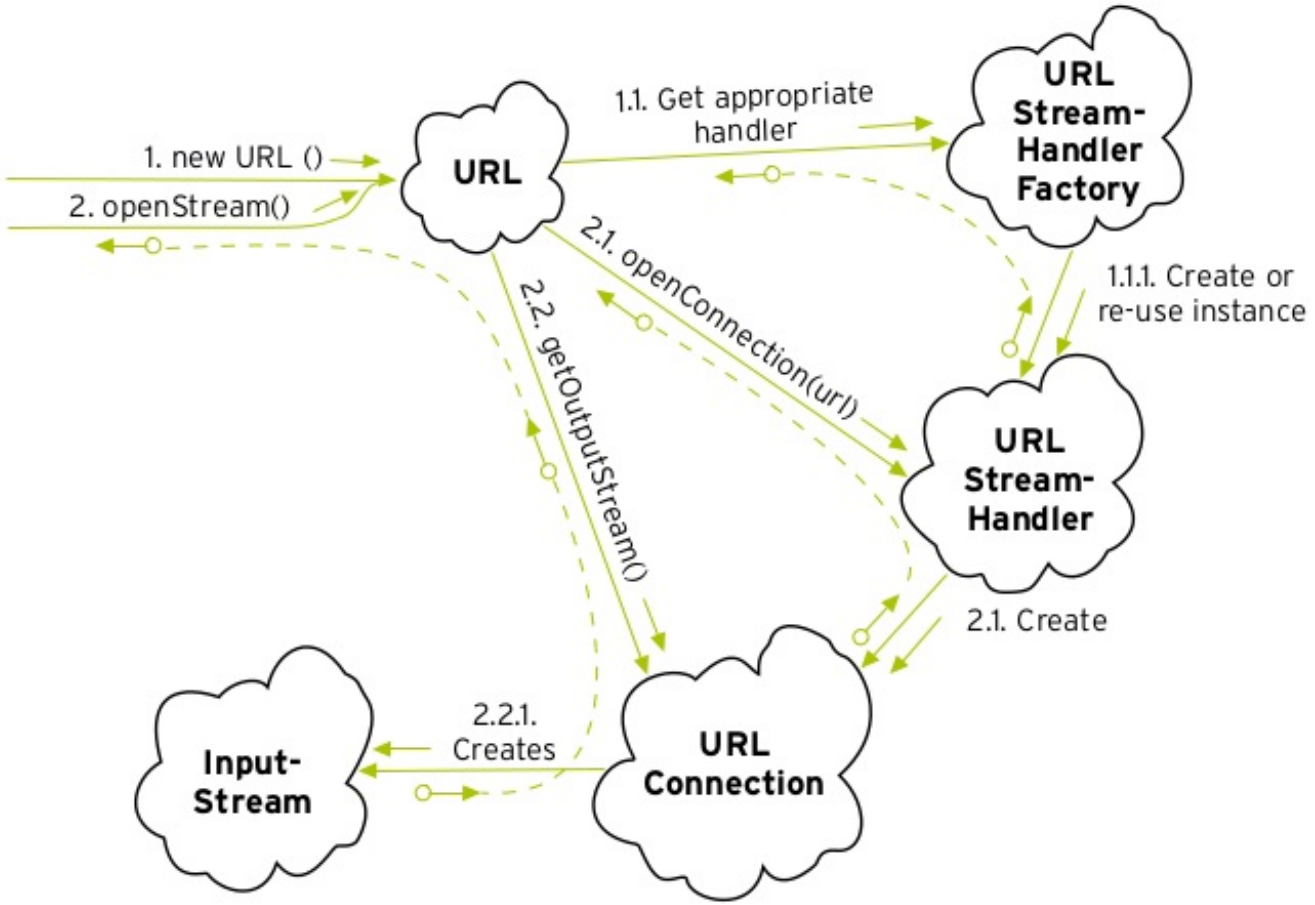
```
1.  import java.net.*;
2.  import java.io.*;
3.
4.  public class UDPServer {
5.
6.   static byte buffer[] = new byte[ 1024 ];
7.
8.   public static void main(String args[]) {
9.    String host = "localhost";
10.   int port = 7030;
11.
12.   String message;
13.   int n = 1;
14.
15.   try {
16.
17.    DatagramSocket serverSocket = new DatagramSocket(port);
18.
19.
20.    while (true) {
21.     DatagramPacket packet = new DatagramPacket(buffer,
    buffer.length);
22.     serverSocket.receive(packet);
23.     message = new String(packet.getData(), "UTF-8");
24.     System.out.println(packet.getAddress() + " on port " +
25.      packet.getPort() + " send: " + message);
```

```
26.
27.
28.       }
29.     } catch(Exception e) {
30.       System.err.println(e);
31.     }
32.   }
33. }
34.
```

5. Gestion de protocole avec URLStreamhandler

La gestion de protocole consiste permet l'accès et la récupération de données, ceci en fonction du protocole choisi : ftp, http, ... Quatre classes sont dévolues à ce type de traitement :

- URL
- URLConnection
- URLStreamHandler
- URLStreamHandlerFactory



(Extrait d'un white paper from developMentor)

```
1.
2.  package com.richer.net.www.protocol.daytime;
3.
4.  import java.net.*;
5.  import java.io.*;
6.
7.  public class DaytimeURLConnection extends URLConnection {
8.
```

```
 9.   private Socket connection = null;
10.   public final static int DEFAULT_PORT = 13;
11.
12.   public DaytimeURLConnection (URL u) {
13.     super(u);
14.   }
15.
16.   public synchronized InputStream getInputStream( ) throws
    IOException {
17.     if (!connected) connect( );
18.
19.     String header = "<html><head><title>The Time at "
20.       + url.getHost( ) + "</title></head><body><h1>";
21.     String footer = "</h1></body></html>";
22.     InputStream in1 = new ByteArrayInputStream(header.getBytes("UTF-
    8"));
23.     InputStream in2 = this.connection.getInputStream( );
24.     InputStream in3 = new ByteArrayInputStream(footer.getBytes("UTF-
    8"));
25.     SequenceInputStream result = new SequenceInputStream(in1, in2);
26.     result = new SequenceInputStream(result, in3);
27.     return result;
28.   }
29.
30.   public String getContentType( ) {
31.     return "text/html";
32.   }
33.
34.   public synchronized void connect( ) throws IOException {
35.     if (!connected) {
36.       int port = url.getPort( );
37.       if ( port <= 0 || port > 65535) {
```

```
      port = DEFAULT_PORT;
    }
    this.connection = new Socket(url.getHost( ), port);
    this.connected = true;
   }
  }
}
```

```
package com.richer.net.www.protocol.daytime;

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

public class Handler extends URLStreamHandler {

  public int getDefaultPort( ) {
    return 13;
  }

  protected URLConnection openConnection(URL u) throws IOException
{
    return new DaytimeURLConnection(u);
  }
}
```

```
1.  package com.richer.net.www.protocol;
2.
3.  import java.net.*;
4.
5.  public class NewFactory implements URLStreamHandlerFactory {
6.
7.    public URLStreamHandler createURLStreamHandler(String protocol) {
8.      if (protocol.equalsIgnoreCase("finger")) {
9.        return new com.richer.net.www.protocol.finger.Handler( );
10.     } else if (protocol.equalsIgnoreCase("daytime")) {
11.       return new com.richer.net.www.protocol.daytime.Handler( );
12.     } else {
13.       return null;
14.     }
15.   }
16. }
17.
18.
```

```
1.  /**
2.   * usage: java HandlerUser daytime://richer
3.   * open ports by installing xinet
4.   * and modify /etc/xinetd.d/daytime and set
5.   *       disable         = no
6.   */
7.  import java.net.*;
8.  import java.io.*;
9.  import com.richer.net.www.protocol.*;
```

```
10.
11. public class UserHandler {
12.
13.   public static void main (String[] args) {
14.     URL.setURLStreamHandlerFactory(new NewFactory( ));
15.     if (args.length > 0) {
16.       try {
17.         //Open the URL for reading
18.         URL u = new URL(args[0]);
19.         InputStream in = new BufferedInputStream(u.openStream( ));
20.         // chain the InputStream to a Reader
21.         Reader r = new InputStreamReader(in);
22.         int c;
23.
24.         while ((c = r.read( )) != -1) {
25.           System.out.print((char) c);
26.         }
27.       } catch (MalformedURLException ex) {
28.         System.err.println(args[0] + " is not a parseable URL");
29.       } catch (IOException ex) {
30.         System.err.println(ex);
31.       }
32.     } // end if
33.   } // end main
34. }
35.
36.
```

Pour exécuter le programme, il faut ouvrir les ports correspondants grâce à xinetd

- installer le paquet **xinetd**
- modifier le fichier `/etc/xinetd.d/echo` :

```
service echo
{
 disable  = no
 type  = INTERNAL
 id  = echo-stream
 socket_type = stream
 protocol = tcp
 user  = nobody
 wait  = no
}
```

- modifier le fichier `/etc/xinetd.d/daytime` :

```
service daytime
{
 disable  = no
 type  = INTERNAL
 id  = daytime-stream
 socket_type = stream
 protocol = tcp
 user  = root
 wait  = no
}
```

- pour finger ajouter le fichier `/etc/xinetd.d/finger`

```
service finger
{
 disable  = no
 socket_type = stream
 user  = nobody
 wait  = no
 server   = /usr/bin/finger
}
```

- relancer xinetd : `sudo service xinetd restart`
- on peut tester dans un premier temps grâce à telnet :

```
telnet localhost 13
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
18 OCT 2012 14:38:46 CEST
Connection closed by foreign host.
telnet localhost 79
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Login     Name                 Tty      Idle  Login Time   Office     Office Phone
richer    Jean-Michel Richer   pts/2       2  Oct 18 09:15 (:0)
richer    Jean-Michel Richer   pts/4          Oct 18 14:20 (:0)
Connection closed by foreign host.
```

On peut ensuite tester avec le UserHandler :

```
java UserHandler daytime://richer
```