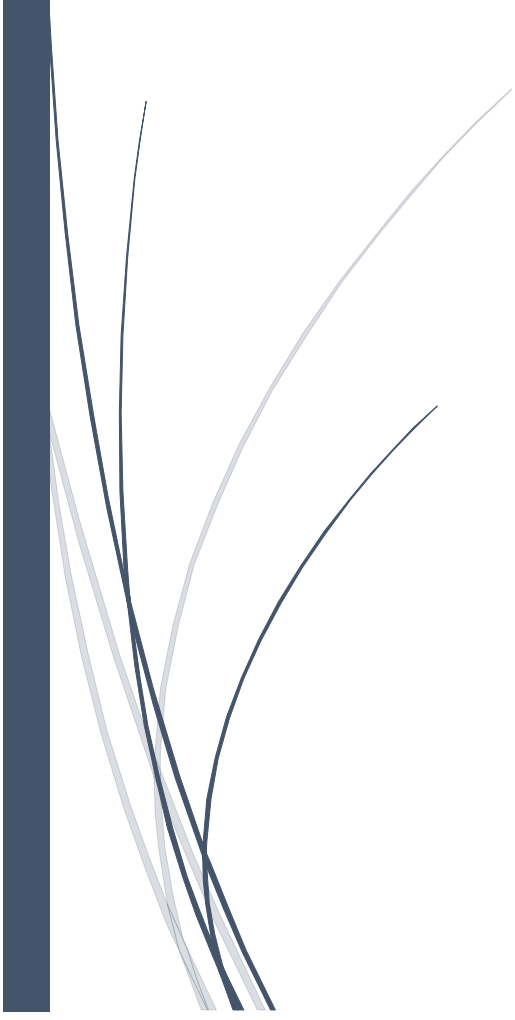
02/10/2015

COMPTE-RENDU DE TP4-JAVA

JAVA ET TCP/IP



Mark Kpamy
GROUPE B

Table des matières

Exercice 1 – Un client Telnet simplifié

Il s'agit ici d'implémenter l'exemple de client Telnet vu en cours:

```
Classe RedirigerFlux :
package tp4;
import java.net.*;
public class RedirigerFlux extends Thread{
//initialisation des variables
	protected java.io.BufferedReader fluxLecture = null;
	protected java.io.PrintStream fluxEcriture = null;

	public RedirigerFlux(java.io.InputStream fl, java.io.OutputStream fe) {
		fluxLecture = new java.io.BufferedReader(new java.io.InputStreamReader(fl));
		fluxEcriture = new java.io.PrintStream(fe);
		super.start();
	}
//fonction permettant de rediriger les flux
	public void run() {
		try {
			String s;
			while ((s = fluxLecture.readLine()) != null)
				fluxEcriture.println(s);
		} catch (java.io.IOException err) {
		}
	}

}
```

Classe Telnet :

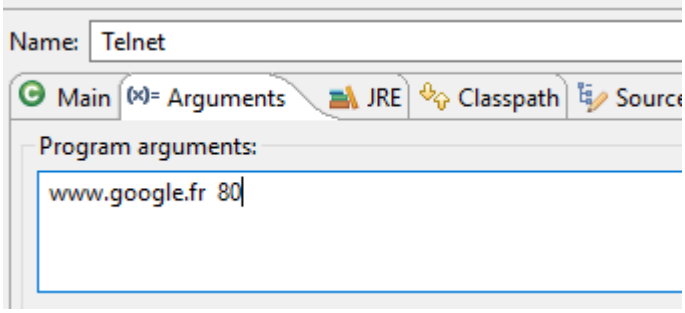
```
package tp4;
import java.net.*;
public class Telnet {

	public static void main(String arg[]) throws Exception {
	//String host="www.google.fr";
	InetAddress machine = InetAddress.getByName(arg[0]) ;
	int port= java.lang.Integer.parseInt(arg[1]);
	Socket socket = new Socket(machine, port);
	// Création du thread redirigeant le flux entrant du socket vers l'écran
	RedirigerFlux socketVersEcran = new RedirigerFlux(socket.getInputStream(), System.out) ;
	// Création du thread redirigeant le clavier vers le flux sortant du socket.
	RedirigerFlux clavierVersSocket = new RedirigerFlux(System.in, socket.getOutputStream()) ;
	// Attendre la fin de lecture sur la socket..
	socketVersEcran.join() ;
	// Arrêter de lire le clavier.
	clavierVersSocket.interrupt() ;
	}
}
```

1er Test :

On teste ce client en nous connectant sur le serveur www.google.fr, sur le port http (80).

Pour cela, on passe en argument dans Eclipse :



On obtient :

```
GET / HTTP/1.1

HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Location: http://www.google.fr/?gfe_rd=cr&ei=Lt8OVoCTK5H5iQbjsJPQCw
Content-Length: 258
Date: Fri, 02 Oct 2015 19:46:54 GMT
Server: GFE/2.0

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.fr/?gfe_rd=cr&amp;ei=Lt8OVoCTK5H5iQbjsJPQCw">here</A>.
</BODY></HTML>
```

2ème Test :

Avec le serveur smtp.orange.fr sur le port 25.

On obtient :

```
Telnet [Java Application] C:\Program Files\Java\jre1.8.0_31\bin\javaw.exe (2 oct. 2015 21:48:57)
220 mwinf5d68 ME ESMTP server ready
HELP
214-2.0.0 This is ME ESMTP service help
214-2.0.0 Topics:
214-2.0.0   HELO   EHLO   HELP   VRFY
214-2.0.0   MAIL   RCPT   DATA   NOOP
214-2.0.0   RSET   QUIT   AUTH
214-2.0.0 To contact postmaster send email to postmaster@orange.fr.
214-2.0.0 For local information send email to Postmaster at your site.
214 2.0.0 end of HELP info
```

Les résultats obtenus nous permettent de conclure qu'après l'ouverture de connexion, le serveur se met en écoute et répond effectivement à réception d'une commande.

Exercice 2 : Le serveur Majuscule

On décide de créer un serveur TCP dans la classe Majuscule et répondant aux spécifications suivantes :

- L'unique service du serveur sera de renvoyer en majuscule tous les messages qu'un client lui envoie.
- Un client se déconnectera du serveur par l'envoie du message "Fin" (ou "fin" ou "FIN" ou tout autre mélange de minuscules et majuscules).

- Le serveur répondra par défaut sur le port 10000, mais il sera possible de le lancer sur un autre port passé en paramètre au lancement du programme

Classe Majuscule1 :

```
package tp4;

import java.io.*;
import java.net.*;
//import java.lang.String.*;

// Serveur TCP ouvert sur le port 10000, permettant pour chaque client de recevoir puis envoyer un message texte.
public class Majuscule1 {
	private ServerSocket sock;
	private Socket client;
	String s1 = null;
	String message = null;

	public void exec() {

		try {
			// Ouverture du port de réception des demandes de connexions
			sock = new ServerSocket(10000);
			// Attente infinie des connexions entrantes
			client = sock.accept();
			while (true) {
				// Attente d'une connexion entrante

				// Une connexion entrante a été établie avec le socket client =>
				// Traitement
				// Ouverture des flux entrant et sortant
				BufferedReader in = new BufferedReader(new InputStreamReader(
						client.getInputStream()));
				PrintWriter out = new PrintWriter(new OutputStreamWriter(
						client.getOutputStream()), true);
				// Protocole de communication
				s1 = in.readLine();
				// Fermeture de la connexion
				if (s1.equalsIgnoreCase("fin")) {
					client.close();
				}
				System.out.println(s1);
				message = s1.toUpperCase();
				// réception
				out.println(message + '\n'+ '\n');// émission
				out.flush();

			}
		} // Traitement des erreurs d'entées / sorties
		catch (Exception e) {
			System.out.println(e);
		}
	}

	public static void main(String argv[]) {
		ServStop serv = new ServStop();
		(new Majuscule1()).exec();
	}
```

```
}
```

Test du client :

```
Telnet [Java Application] C:\Program Files\Java
hello
HELLO


mark
MARK
```

Exercice 3 : Le serveur « Majuscule » multi-clients

Le lancement de 2 clients simultanément n'est pas car le serveur ne peut gérer qu'une seule connexion à la fois.

Pour rémédier à cela, on doit multithreader le serveur. Cela nous amènera à créer une nouvelle classe que chaque thread pourra lancer indépendamment.

Classe Majuscule1 :

```
package tp4;

import java.io.*;
import java.net.*;
//import java.lang.String.*;

// Serveur TCP ouvert sur le port 10000, permettant pour chaque client de recevoir puis envoyer un message texte.
public class Majuscule1 {
	ServerSocket sock;
	Socket client;
	boolean stop = false;

	public void exec() {
		try {
			// Ouverture du port de réception des demandes de connexions
			sock = new ServerSocket(10000);
			// Attente infinie des connexions entrantes

			while (!stop) {
				client = sock.accept();
				// création du la classe qui traitera la connexion du client
				ServiceClient t = new ServiceClient(client);
				// Création du thread de gestion du client donné
				Thread threadService = new Thread(t);
				threadService.start();
			}
		} // Traitement des erreurs d'entées / sorties
		catch (Exception e) {
			System.out.println(e);
		}
	}

	public static void main(String argv[]) {
		ServStop serv = new ServStop();
		(new Majuscule1()).exec();
	}
}
```

Classe ServiceClient :

```
package tp4;

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

public class ServiceClient implements Runnable {
	private Socket s;
	private BufferedReader in;
	private PrintWriter out;
	String s1 = null;
	String message = null;
	 boolean termine = false;

	public ServiceClient(Socket s) throws IOException {
		this.s = s;
		// variables permettant de récuperer les flux entrants et sortants
		in = new BufferedReader(new InputStreamReader(s.getInputStream()));
		out = new PrintWriter(new OutputStreamWriter(s.getOutputStream()));
	}

	public void run() {
		try {
			while (!termine) {
				// Une connexion entrante a été établie avec le socket client =>
				s1 = in.readLine();
				// Fermeture de la connexion
				if (s1.equalsIgnoreCase("fin")) {
					s.close();
				}
				// réception
				System.out.println(s1);
				message = s1.toUpperCase();
				// émission
				out.println(message + '\n'+ '\n');
				out.flush();
				}
		} catch (IOException ioe) {
		} finally {
			try {
				//fermeture du socket avant la fin du traitement du client
				s.close();
			} catch (IOException ioe) {
			}
		}
	}
}
```

Conclusion :

Ce TP nous a permis de comprendre l'interaction entre Java et les protocoles TCP/IP. Tout cela appliqué à la fin à la notion de thread.