11/10/2015

COMPTE-RENDU DE TP5-JAVA

JAVA ET TCP/IP

Mark Kpamy
GROUPE B

Table des matières

Exercice 1 – Le serveur d'affichage

Il s'agit ici de créer un serveur affichant dans sa console et dans sa fenêtre IHM qui lui est liée les messages provenant d'un client pouvant envoyer autant de messages qu'il le souhaite.

a) Preparation des clients :

Classe Ecrivain :

```
package tp5;

/**
 * Titre :        TP6 Année
 * Description :  TP Réseau
 * Copyright :    Copyright (c) 2001
 * Société :
 * @author Thierry Champion
 * @version 1.0
 */

import java.io.*;
import java.net.*;
//classe permettant d'envoyer un message au serveur
public class Ecrivain extends Thread implements EcrivainInterface {

  private InetAddress adresse ;
  private Socket sock ;
  private PrintWriter out;
//les deux constructeurs
  //le premier est lancé si on spécifie un serveur en argument
  Ecrivain(String serveur) {
    System.out.println("Lancement d'un client vers : " + serveur);
    try {
      adresse = InetAddress.getByName(serveur) ;
      sock = new Socket(adresse, 6000) ;
      out = new PrintWriter(
                      new OutputStreamWriter(sock.getOutputStream()),true);
                      // true permet de faire de l'auto-flush.
    }
    catch (Exception e) { System.out.println(e) ; }
  }
//le deuxième est lancé si on ne précise pas d'arguments
  //il le lance avec comme serveur la machine locale
  Ecrivain() { this("localhost"); }

  //methode permettant l'envoi du message
  public void envoieMessage(String mess) {
    System.out.println("Emission du message : " + mess);
    try {
       //envoi du message
      out.println(mess) ;
    }
    catch (Exception e) { System.out.println(e) ; }
  }
//methode principale
  public static void main(String argv[]) {
    Ecrivain ecrivain ;
    String name ;
    //selon les cas
    switch (argv.length) {
```

```
      case 1 ://si on a 1 seul argument
        name=argv[0];
        ecrivain = new Ecrivain() ;
        break;
      case 2 :
         //si on a 2 argument
        name=argv[0];
        ecrivain = new Ecrivain( argv[1]) ;
        break;
      default ://à défaut
        name="Ecrivain";
        ecrivain = new Ecrivain() ;
        break;
    }
//on crée son IHM en passant en paramètre son nom et
    new EcrivainIHM(name, ecrivain);
  }
}
```

Classe EcrivainIHM :

```
package tp5;

/**
 * Titre :
 * Description :  TP Réseau
 * Copyright :    Copyright (c) 2001
 * Société :
 * @author Thierry Champion
 * @version 1.0
 */

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

public class EcrivainIHM extends Frame {
       // attributs pour la création de la fenêtre
       Panel panel1 = new Panel();
       BorderLayout borderLayout1 = new BorderLayout();
       Button bArret = new Button();
       Label texte = new Label();
       Panel panel2 = new Panel();
       Label blanc = new Label();
       TextField messageText = new TextField();

       Vector ecrivains = new Vector();

       // 1er constructeur
       public EcrivainIHM(String title) {
              super(title);
              try {
                     jbInit();
                     texte.setText(title);// on nomme le label
                     pack();
                     show();
              } catch (Exception ex) {
                     ex.printStackTrace();
              }
```

```
	}

	// 2eme constructeur
	public EcrivainIHM() {
		this("");
	}

	// 3eme constructeur
	public EcrivainIHM(String title, EcrivainInterface e) {
		this(title);
		this.addEcrivain(e);
	}

	//
	public void addEcrivain(EcrivainInterface e) {
		ecrivains.addElement(e);
	}

	// methode qui redefinit la methode envoieMessage de l'interface IHM
	void envoieMessage(String mess) {
		// elle permet d'envoyer un message précédé du nom de la fenetre
		for (Enumeration e = ecrivains.elements(); e.hasMoreElements();)
			((EcrivainInterface) e.nextElement()).envoieMessage(this.getTitle() + " : " +
mess);
	}

	// fonction de création de la fenêtre
	void jbInit() throws Exception {
		panel1.setLayout(borderLayout1);
		bArret.setLabel("Arret !");
		// ecouteur du bouton d'arrêt
		bArret.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(ActionEvent e) {
				bArret_actionPerformed(e);
			}
		});
		this.addWindowListener(new java.awt.event.WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				this_windowClosing(e);
			}
		});
		blanc.setText("                                                                ");
		messageText.setColumns(50);
		// ecouteur du text field
		messageText.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(ActionEvent e) {
				messageText_actionPerformed(e);
			}
		});
		this.add(panel1, BorderLayout.NORTH);
		panel1.add(panel2, BorderLayout.NORTH);
		panel2.add(texte, null);
		panel2.add(bArret, null);
		panel2.add(blanc, null);
		panel1.add(messageText, BorderLayout.SOUTH);
	}

	void bArret_actionPerformed(ActionEvent e) {
		// provoque la fermeture du programme lorsque l'on clique sur le bouton
```

```
			// d'arrêt
			System.exit(1);
		}

		void this_windowClosing(WindowEvent e) {
			// provoque la fermeture du programme lorsque l'on ferme la fenêtre
			System.exit(1);
		}

		// permet d'envoyer le texte saisi dans le text field
		void messageText_actionPerformed(ActionEvent e) {
			this.envoieMessage(messageText.getText());
			messageText.setText("");
		}
}
```

Classe EcrivainInterface :

```
package tp5 ;

/**
 * Titre :
 * Description :  TP Réseau
 * Copyright :    Copyright (c) 2001
 * Société :
 * @author Thierry Champion
 * @version 1.0
 */
//interface qui declare la methode envoieMessage()
public interface EcrivainInterface {
  public void envoieMessage(String mess);
}
```

b) Ecrire le serveur : classe ServeurMess

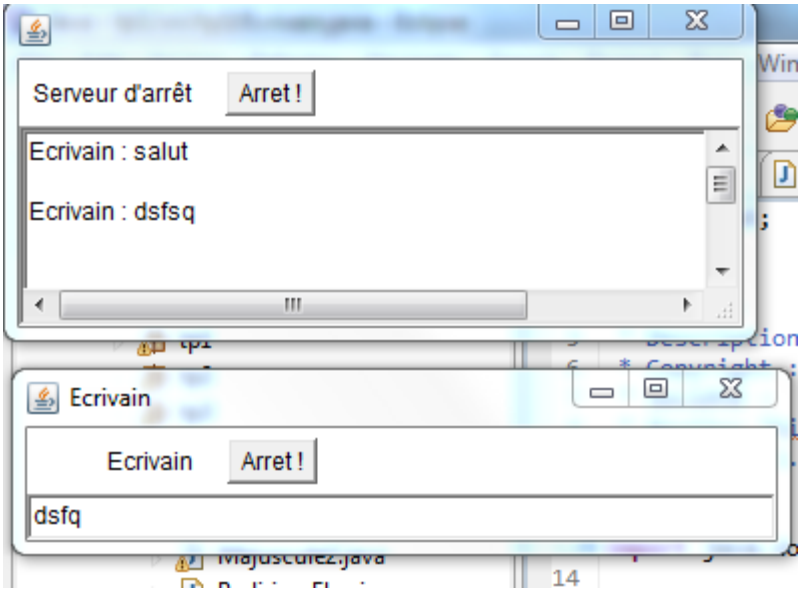
Dans cette partie nous allons maintenant créer notre serveur nous permettant de recevoir les messages de la part d'un client :

Classe ServeurMess :

```
package tp5;

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

public class ServeurMess {
	private ServerSocket sock;
	private Socket client;
	String mess = "";
	ServIHM serv;
	boolean stop = false;

	// méthode implémentant le fonctionnement du serveur
	public void run() {

		try {
			sock = new ServerSocket(6000);
```

```
                    // Attente d'une connexion entrante
                    client = sock.accept();
                    // Ouverture du port de réception des demandes de connexions
                    // Attente infinie des connexions entrantes
                    while (!stop) {


                          // 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()));
                          // Protocole de communication
                          mess = in.readLine();
                          System.out.println(mess);
                          serv.affiche(mess + '\n');
                          // Fermeture de la connexion

                    }
              } // Traitement des erreurs d'entrées / sorties
              catch (IOException ioe) {
              } finally {
                    try {
                          // fermeture du socket avant la fin du traitement du client
                          client.close();
                    } catch (IOException ioe) {
                    }
              }
        }

        // Un constructeur qui recevra en argument un objet de la classe ServIHM
        public ServeurMess(ServIHM serv) {
              this.serv = serv;
        }

        // méthode principale permettant de lancer ce serveur accompagné de son IHM
        public static void main(String argv[]) {
              ServIHM fen = new ServIHM("Serveur d'arrêt");
              new ServeurMess(fen).run();
        }
}
```

Test du serveur avec un client puis 2 clients:



On voit bien qu'il reçoit et affiche les messages du seul client. Lorsque l'on lance un autre un autre client, ses messages ne sont pas reçus car le serveur n'est pas mullti-threadé. Pour donc pallier à ce problème, on va multi-threader le serveur. Il s'agira de lancer à chaque fois un thread pour gérer les requêtes de chaque client indépendamment. On fera tous les traitements dans une nouvelle classe que l'on nommera ServeurRequete . Lors la connexion, on créera une instance de ServeurRequete que l'on passera ensuite en paramètre au thread. Pour cela, ServeurRequete devra être Runnable et donc implémenter la méthode run() .

c) Multi-threading du serveur

Après modification, voici les classes obtenues :

Classe ServeurMess :

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

public class ServeurMess {
	private static Vector tabClients = new Vector<ServeurRequete>(); // contiendra tous les flux de sortie vers les clients
	 private static int nbClients=0;
	static ServIHM serv;
	public void run() throws IOException {
		Socket client;
		// Ouverture du port de réception des demandes de connexions
		ServerSocket sock = new ServerSocket(6000);
		boolean stop = false;

			// Attente infinie des connexions entrantes
			while (!stop) {
				client = sock.accept();
				ServeurRequete s =new ServeurRequete(client);
				Thread threadService = new Thread(s);
				threadService.start();

				}
			sock.close();
```

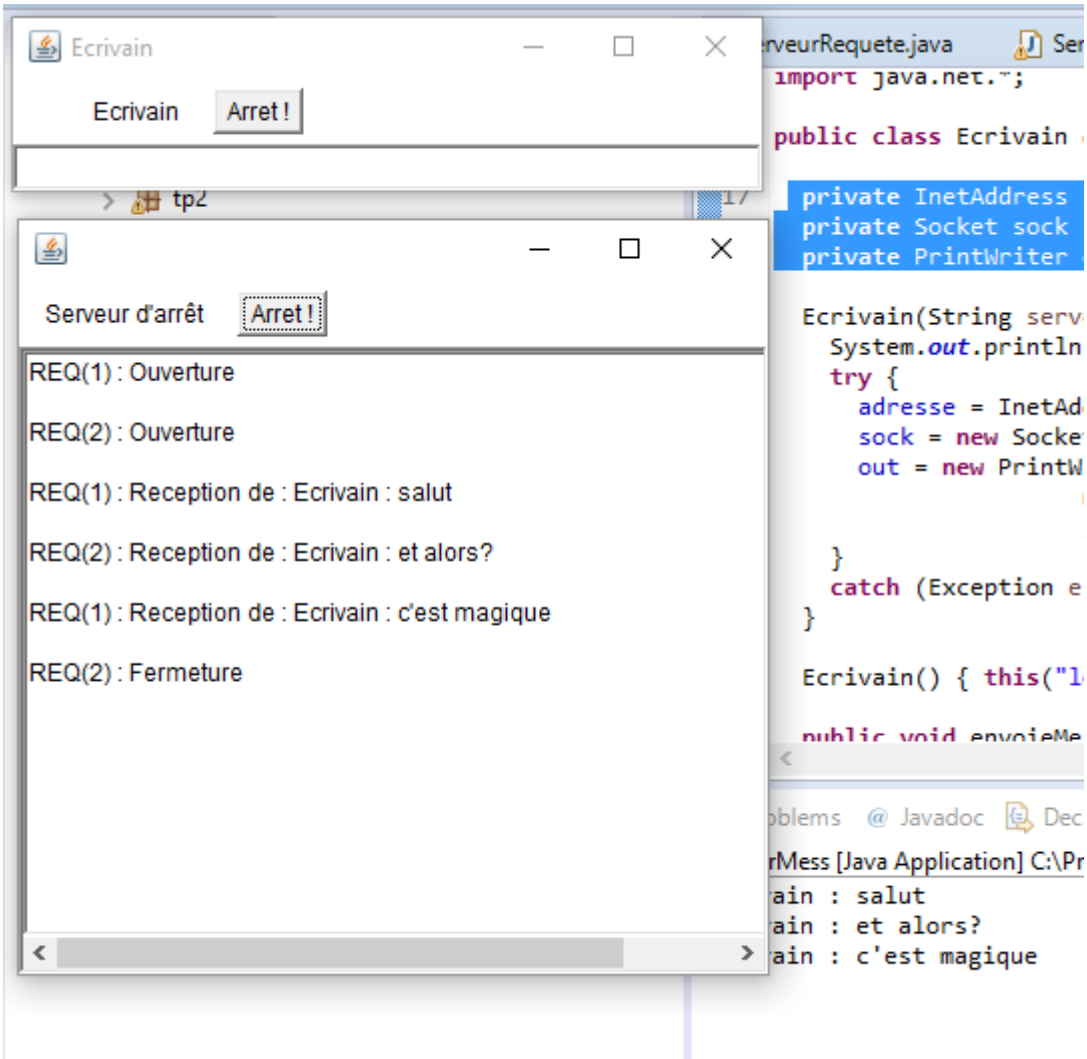
```
	}
	// Un constructeur qui recevra en argument un objet de la classe ServIHM
	synchronized public static int addClient(PrintWriter out)
	  {
	    nbClients++; // un client de plus
	    tabClients.addElement(out); // on ajoute le nouveau flux de sortie au tableau
	    return tabClients.size(); // on retourne le numéro du client ajouté
	  }
	public ServeurMess(ServIHM serv) {
		this.serv = serv;
	}

	// méthode principale permettant de lancer ce serveur accompagné de son IHM
	public static void main(String argv[]) throws IOException{
		ServIHM fen = new ServIHM("Serveur d'arrêt");
		new ServeurMess(fen).run();
	}
}
```

Classe ServeurRequete :

```
package tp5;

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

public class ServeurRequete implements Runnable {
	private int numClient = 0;
	private Socket client;
	String mess = "";
	boolean stop = false;
	BufferedReader in;
	PrintWriter out;

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

	// méthode implémentant le fonctionnement du serveur
	public void run() {

		try {
			// Attente infinie des connexions entrantes
			ServeurMess.serv.affiche("REQ(" + numClient + ") : Ouverture" + '\n');
			while (!stop) {
				// Protocole de communication
				mess = in.readLine();
				System.out.println(mess);
				ServeurMess.serv.affiche("REQ(" + numClient + ") : Reception de : " +
mess + '\n');
				// Fermeture de la connexion

			}
		} // Traitement des erreurs d'entées / sorties
```

```
            catch (IOException ioe) {
            } finally {
                  try {
                        // fermeture du socket avant la fin du traitement du client
                        client.close();
                        ServeurMess.serv.affiche("REQ(" + numClient + ") : Fermeture");
                  } catch (IOException ioe) {
                  }
            }
      }

}
```

Test :



On constate bien que le serveur peut à présent gérer plusieurs clients.

Exercice 2 : Le serveur de messages

On décide de créer un serveur TCP/IP permettant de recevoir des messages de la part de clients Ecrivain, les afficher dans sa console et sa fenêtre et aussi les rediriger vers les clients Lecteur.

Classe Lecteur :

Cette classe est à l'image de la classe Ecrivain sauf qu'elle écouter les messages envoyer par le serveur et les afficher.

```
package tp5;

import java.io.*;
import java.net.InetAddress;
import java.net.Socket;

public class Lecteur implements Runnable {
	private InetAddress adresse;
	private Socket sock;
	private BufferedReader in;
	String mess;
	ServIHM ihm;

	Lecteur(String serveur, ServIHM ihm) {
		this.ihm = ihm;
		try {
			adresse = InetAddress.getByName(serveur);
			sock = new Socket(adresse, 6001);
			in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
		} catch (Exception e) {
			System.out.println(e);
		}
	}

	Lecteur(ServIHM ihm) {
		this("localhost", ihm);
		this.ihm = ihm;
	}

	public void run() {
	//ecoute du message
		try {
			while (true) {
				mess = in.readLine();
				System.out.println(mess);
				ihm.affiche(mess);
			}
		}

		catch (Exception e) {
			System.out.println(e);
			try {
				sock.close();
			} catch (IOException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
		}
	}

	public static void main(String argv[]) {
		ServIHM fen = new ServIHM("Lecteur");
		new Lecteur(fen).run();
	}
}
```

Classe ServeurDouble :

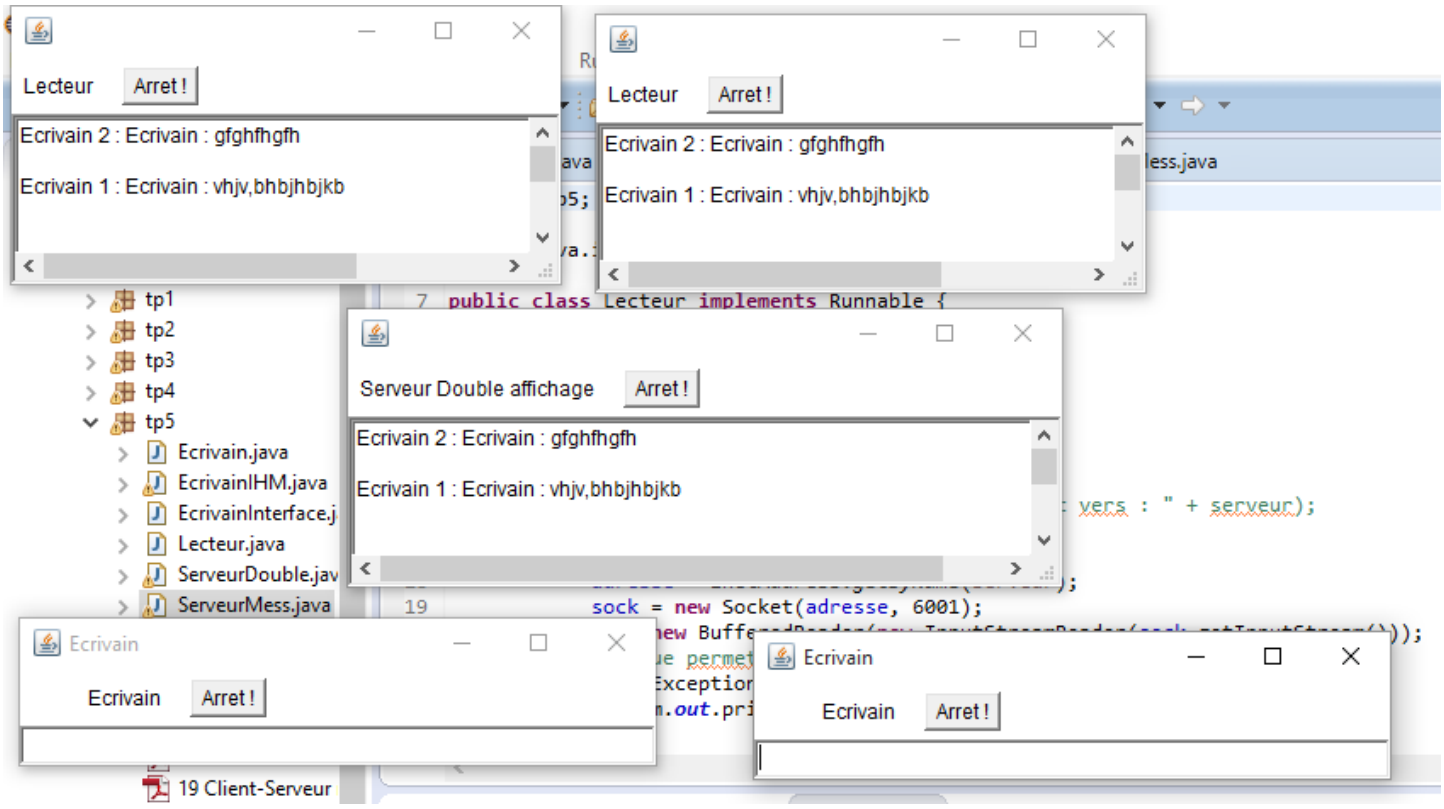
```
package tp5;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;

public class ServeurDouble {
	ServIHM ihm;
	String mess;
	BufferedReader in;
	PrintStream out;
	Vector<PrintStream> listeLecteurs = new Vector<PrintStream>();

	Socket client1;
	Socket client2;
	boolean stop = false;
	private Vector<BufferedReader> tabEcrivain = new Vector<BufferedReader>(); //

	synchronized public int addClient(PrintStream out) {
		listeLecteurs.addElement(out); // on ajoute le nouveau flux de sortie au
		                               // tableau
		return listeLecteurs.size(); // on retourne le numéro du client ajouté
	}

	synchronized public int addClient(BufferedReader in) {
		tabEcrivain.addElement(in); // on ajoute le nouveau flux de sortie au
		                            // tableau
		return tabEcrivain.size(); // on retourne le numéro du client ajouté
	}

	public void runEmission() throws IOException {
		int numLecteur = 0;
		out = new PrintStream((client1.getOutputStream()));
		;
		numLecteur = addClient(out);

	}

	public void runReception() throws IOException {
		int numEcrivain = 0;
		in = new BufferedReader(new InputStreamReader(client2.getInputStream()));
		numEcrivain = addClient(in);
		while (true) {
			mess = in.readLine();
			System.out.println("Ecrivain " + numEcrivain + " : " + mess + '\n');
			ihm.affiche("Ecrivain " + numEcrivain + " : " + mess + '\n');

			for (int i = 0; i < listeLecteurs.size(); i++) {
				PrintStream ps = listeLecteurs.elementAt(i);
				ps.println("Ecrivain " + numEcrivain + " : " + mess + '\n');
				ps.flush();
			}
		}
```

```
	}

	public void exec() throws IOException {
		while (!stop) {
//thead des ecrivains
			new Thread() {
				public void run() {
					try {
						ServerSocket sock1 = new ServerSocket(6000);
						while (!stop) {
							client2 = sock1.accept();
							//thread de chaque ecrivain
							new Thread() {
								public void run() {
									try {
										runReception();
									} catch (IOException e) {
										//	TODO	Auto-generated	catch
block

										e.printStackTrace();
									}
								}
							}.start();
						}
						sock1.close();
					} catch (IOException e1) {
						// TODO Auto-generated catch block
						e1.printStackTrace();
					}
				}
			}.start();
//thread des lecteurs
			new Thread() {
				public void run() {
					try {
						ServerSocket sock2 = new ServerSocket(6001);
						while (!stop) {
							client1 = sock2.accept();
							//thread de chaque lecteur
							new Thread() {
								public void run() {
									try {
										runEmission();
									} catch (IOException e) {
										//	TODO	Auto-generated	catch
block

										e.printStackTrace();
									}
								}
							}.start();
						}
						sock2.close();
					} catch (IOException e1) {
						// TODO Auto-generated catch block
						e1.printStackTrace();
					}
				}
			}.start();
		}
```

```
            // fermeture du socket avant la fin du traitement du client
            //
            //

      }

      public ServeurDouble(ServIHM ihm) {
            this.ihm = ihm;
      }

      public static void main(String[] args) throws IOException {
            ServIHM fenetre = new ServIHM("Serveur Double affichage");
            new ServeurDouble(fenetre).exec();
      }
}
```

Test :

Ce code génère affiche un bon nombre d'erreurs dans la console dont j'arrive pas à connaitre la cause. Il marche néanmoins pour les fenêtres.



Conclusion :

Ce TP comme le précédent nous a permis d'approfondir nos connaissances sur l'interaction entre Java et les protocoles TCP/IP et surtout le multi-threading.