

# Université de Carthage

Institute National des Sciences Appliqués et de Technologie  
Année Universitaire 2018-2019

## COMPTE RENDU

### Applications Réparties

### TP4



Réalisée par :

SNOUSSI Anis

BARGHOUDA Mohamed Lamine



## Introduction

Les échanges avec le réseau sont devenus omniprésents dans les applications et entre les applications. Ils permettent notamment :

- Un accès à un serveur comme une base de données
- D'invoquer des services distants
- De développer des applications web
- D'échanger des données entre applications

La plupart de ces fonctionnalités sont mises en œuvre grâce à des API de haut niveau mais celles-ci utilisent généralement des API de bas niveau pour interagir avec le réseau.

Depuis son origine, Java fournit plusieurs classes et interfaces destinées à faciliter l'utilisation du réseau par programmation en reposant sur les sockets. Celles-ci peuvent être mises en œuvre pour réaliser des échanges utilisant le protocole réseau IP avec les protocoles de transport TCP ou UDP. Les échanges se font entre deux parties : un client et un serveur.

Il existe deux protocoles utilisant les sockets en Java :

- TCP : Stream Socket
  - Communication en mode connectés.
  - Les applications sont averties lors de la déconnexion.
- UDP : Datagram Socket
  - Communication en mode non connectés.
  - Plus rapide mais moins fiable que TCP.

## Serveur Simple

Le serveur ne traite qu'un client à la fois, le serveur et le client n'échangent qu'un message chacun.

Le programme serveur ouvre un socket d'écoute et attend une connexion. Dès qu'un client se connecte, il lui envoie une phrase, puis attend une réponse, puis ferme la connexion et se remet en attente.

```
import java.io.*;
import java.net.*;
class ServeurSimple {
    public static void main(String[] args) {
        int port = 1973;
        ServerSocket SocketServeur = null;
        Socket socket = null;
        int client = 0;
        if (args.length == 1) {
            port = Integer.parseInt(args[0]);
        }

        try { // 5 personnes en attente maxi
            SocketServeur = new ServerSocket(port, 5);
        } catch (IOException e) {
            System.err.println("Impossible de creer un ServerSocket");
            return;
        }

        System.out.println("Serveur l' écoute sur le port :" + port);
        while (true) {
            try {
                socket = SocketServeur.accept();

                client++; // un client s'est connect
                PrintWriter out = new PrintWriter(socket.getOutputStream());
                BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                out.println("Je suis le serveur : quelquechose declarer?");
                out.flush();
                String reply = in .readLine();
                System.out.println("Le client no. " + client + " m'a dit : " + reply);
            }
        }
    }
}
```

```
catch (IOException e) {  
    System.err.println("Erreur : " + e);  
} finally {  
    try {  
        socket.close();  
    } catch (IOException e) {}  
}  
}  
}
```

### ServeurSimple.java

Pour le client :

- Il reçoit en paramètre l'adresse IP et le port du serveur.
- Il crée un socket TCP pour se connecter sur la machine/port du serveur, puis affiche à l'écran le texte qu'il a reçu du serveur, puis envoie une réponse au serveur.

```
import java.io.*;  
import java.net.*;  
import java.util.Scanner;  
class ClientSimple {  
    public static void main(String[] args) {  
        InetAddress hote = null;  
        int port = 1973; // par défaut  
        Socket socket = null;  
        try {  
            if (args.length >= 1) hote = InetAddress.getByName(args[0]);  
            else hote = InetAddress.getLocalHost();  
            if (args.length == 2) port = Integer.parseInt(args[1]);  
        } catch (UnknownHostException e) {}  
  
        try {  
            socket = new Socket(hote, port);  
        }
```

```

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
String reply = in .readLine();
System.out.println(reply);

Scanner sc = new Scanner(System.in);
PrintWriter out = new PrintWriter(socket.getOutputStream());
System.out.println("ecrire un message au serveur:");
out.println(sc.nextLine());
out.flush();

socket.close();
} catch (IOException e) {}
}
}

```

### ClientSimple.java

### Résultat de l'exécution :

#### Client :

```

Anis SNOUSSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice1
$ java ClientSimple
Je suis le serveur : quelquechose declarer?
ecrire un message au serveur:
Hello GL2

```

#### Serveur :

```

Anis SNOUSSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice1
$ java ServeurSimple
Serveur l' coute sur le port :1973
Le client no. 1 m'a dit : Hello GL2

```

## Serveur acceptant plusieurs connexions simultanées

Chaque client a son propre socket de connexion et tourne dans un thread dédié.

Une boucle crée un nouveau Thread (classe ThreadClient) chaque nouvelle connexion et lui passe le socket de connexion. Créer cette classe de thread dédiée à chaque client.

Son programme principal sera de :

1. envoyer un premier message amical d'accueil
2. Attendre une phrase de réponse du client
3. Faire une pause d'une seconde
4. Envoyer une phrase au client en utilisant la méthode `message_suivant()`
5. Boucler en revenant à l'étape 2
6. Sortir de la boucle si la réponse du client est vide ou si le client s'est déconnecté ( on a une exception )
7. Penser à ajouter une méthode `protected void finalize()` qui fermera proprement socket et streams

```
import java.io.*;
import java.net.*;
public class ServeurMultiClient {
    public static void main(String[] args) {
        int port = 1973;
        ServerSocket SocketServeur = null;
        Socket socket = null;
        int client = 0;
        if (args.length == 1) {
            port = Integer.parseInt(args[0]);
        }

        try { // 5 personnes en attente maxi
            SocketServeur = new ServerSocket(port, 5);
        } catch (IOException e) {
            System.err.println("Impossible de creer un ServerSocket");
            return;
        }
    }
}
```

```

System.out.println("Serveur l' coute sur le port :" + port);
while (true) {
    try {
        socket = SocketServeur.accept();
        client++;
        //on le passe a un nouveau thread et on le demarre
        ThreadClient neo = new ThreadClient(socket, client);
        neo.start();
    } catch (IOException e) {
        System.err.println("Erreur : " + e);
    }
}
}
}
}

```

### ServeurMultiClient.java

```

import java.io.*;
import java.net.*;
import java.util.Scanner;
public class ClientExo2 {
    public static void main(String[] args) {
        InetAddress hote = null;
        int port = 1973; // par défaut
        Socket socket = null;
        try {
            if (args.length >= 1) hote = InetAddress.getByName(args[0]);
            else hote = InetAddress.getLocalHost();
            if (args.length == 2) port = Integer.parseInt(args[1]);
        } catch (UnknownHostException e) {}

        try {
            socket = new Socket(hote, port);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            int compteur = 0;
            while (compteur < 10) {
                String msg = in.readLine();
                System.out.println(msg);
                compteur++;
                out.println("Je suis le client " + hote + " et j'ai fait " + compteur + " appels");
            }
        }
    }
}

```

```

    try {
        System.out.println("pause de 2 secondes");
        Thread.sleep(2000);
    } catch (InterruptedException e) {}

    }
    out.println("");
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

### ClientExo2.java

```

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

public class ThreadClient extends Thread {
    private Socket socket = null;
    public int clientNo;
    public int reqcount = 0;
    PrintWriter out;
    BufferedReader in ;
    public void run() {
        System.out.println("thread started " + clientNo);
        try {
            out = new PrintWriter(socket.getOutputStream(), true); in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out.println("Je suis le serveur : quelque chose a declarer?");
            String reply = in .readLine();
            while (!reply.equals("")) {
                faireUnePause();
                out.println(message_suivant());
                reply = in .readLine();
                System.out.println(reply);
            }
        } catch (IOException e) {
            System.err.println("Erreur : " + e);}
    }
}

```



```
finally {
    try {
        socket.close();
    } catch (IOException e) {
        System.err.println("Erreur : " + e);
    }
}
}
public ThreadClient(Socket socket, int no) {
    super("ThreadClient");
    this.socket = socket;
    this.clientNo = no;
}
private String message_suivant() {
    reqcount++;
    switch (reqcount % 5) {
        case 0:
            return new String("Marrakech est une ville magnifique.");
        case 1:
            return new String("La medina de Fes est splendide au couchant.");
        case 2:
            return new String("Les montagnes de l'Atlas sont impressionnantes.");
        case 3:
            return new String("La place Jamaa alfna est au centre de la ville.");
        case 4:
            return new String("Les cotes du Maroc valent le coup d'oeil.");
    }
    return new String("ca n'arrive jamais");
}
void faireUnePause() {
    try {
        System.out.println("pause d'une seconde");
        Thread.currentThread().sleep(1000);
    } catch (InterruptedException e) {}
}
}
```

### ThreadClient.java

## Résultat de l'exécution :

### Serveur :

```

MINGW64:/c/Users/Anis SNOUSSI/Desktop/TP/Exercice2

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice2
$ java ServeurMultiClient
Serveur l'écoute sur le port :1973
thread started 1
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 2 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 3 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 4 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 5 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 6 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 7 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 8 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 9 appels
pause d'une seconde
Je suis le client Anis-Desktop/10.0.75.1 et j'ai fait 10 appels
pause d'une seconde

```

### Client :

```

MINGW64:/c/Users/Anis SNOUSSI/Desktop/TP/Exercice2

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice2
$ java ClientExo2
Je suis le serveur : quelque chose à déclarer?
pause de 2 secondes
La médina de Fes est splendide au couchant.
pause de 2 secondes
Les montagnes de l'Atlas sont impressionnantes.
pause de 2 secondes
La place Jamaa alfna est au centre de la ville.
pause de 2 secondes
Les côtes du Maroc valent le coup d'oeil.
pause de 2 secondes
Marrakech est une ville magnifique.
pause de 2 secondes
La médina de Fes est splendide au couchant.
pause de 2 secondes
Les montagnes de l'Atlas sont impressionnantes.
pause de 2 secondes
La place Jamaa alfna est au centre de la ville.
pause de 2 secondes
Les côtes du Maroc valent le coup d'oeil.
pause de 2 secondes

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice2
$ 

```

## Client avec interface graphique

On modifie le programme client pour qu'il affiche les réponses du serveur de l'exercice 2 dans une fenêtre AWT. Il donnera aussi la possibilité de saisir du texte dans une zone de saisie qu'il enverra au serveur.

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

class ClientFenetre extends Frame implements Runnable, ActionListener {
    TextArea Output;
    TextField Input;
    Socket socket = null;
    BufferedReader in ;
    PrintWriter out;

    public ClientFenetre(String hote, int port) {
        super("Client en fenetre");
        // mise en forme de la fenetre (frame)
        setSize(500, 700);
        setLayout(new BorderLayout());
        this.add(Output = new TextArea(), BorderLayout.CENTER);
        Output.setEditable(false);
        this.add(Input = new TextField(), BorderLayout.SOUTH);
        Input.addActionListener(this);
        pack();
        show();
        Input.requestFocus();
        // ajout d'un window adapter pour reagir si on ferme la fenetre
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                setVisible(false);
                dispose();
                System.exit(0);
            }
        });
        try {
            socket = new Socket(hote, port);
            out = new PrintWriter(socket.getOutputStream(), true); in = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));
        } catch (IOException e) {
```

```

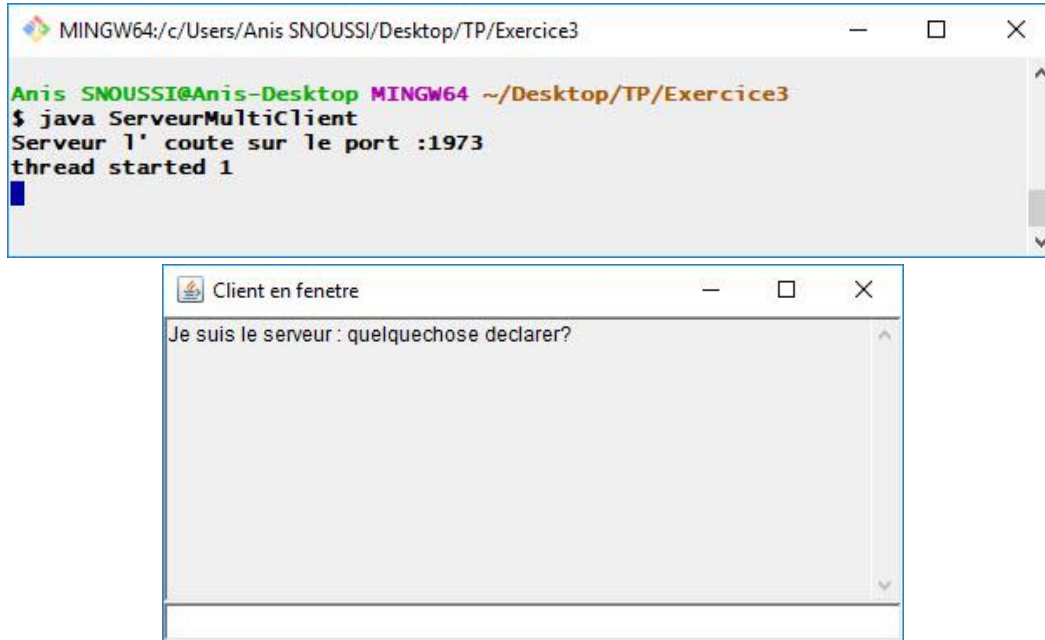
        e.printStackTrace();
    }
    Thread t = new Thread(this);
    t.start();
}
public void run() {
    try {
        int compteur = 0;
        while (compteur < 10) {
            String msg = in.readLine();
            Output.append(msg + "\n");
            compteur++;
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == Input) {
        String phrase = Input.getText();
        out.println(phrase);
        Input.setText("");
    }
}
protected void finalize() {
    try {
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public static void main(String[] args) {
    String hote = null;
    int port = 1973; // par d faut
    ClientFenetre chatwindow = new ClientFenetre(hote, port);
}
}

```

### ClientFenetre.java

## Résultat de l'exécution :

### Initialement

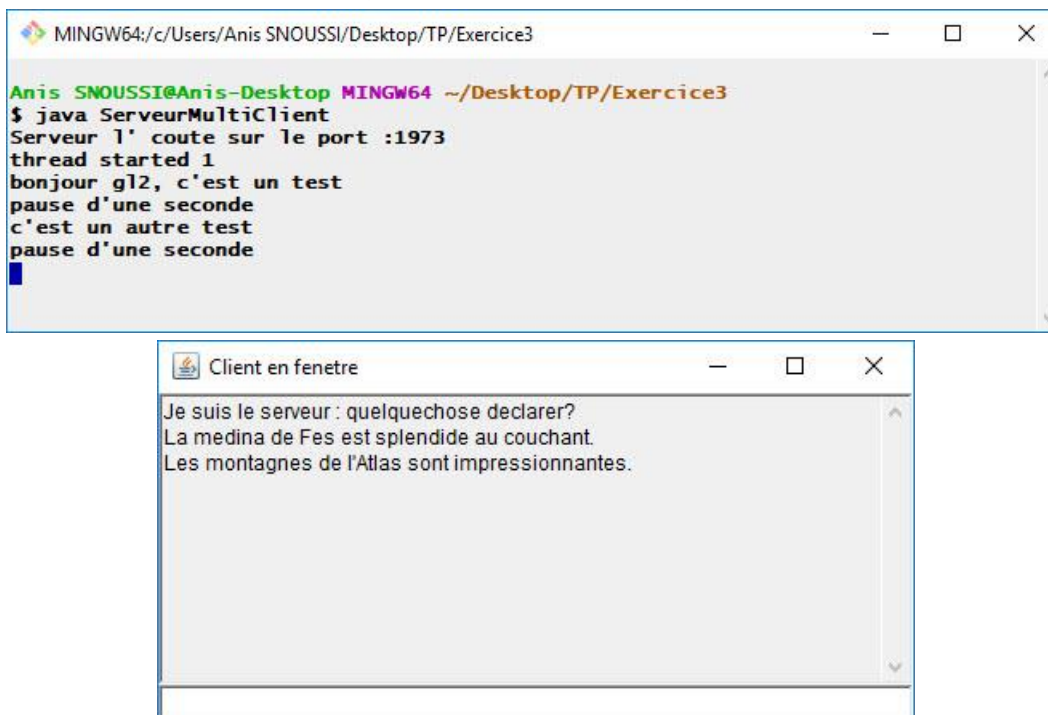


```
MINGW64:/c/Users/Anis SNOUSSI/Desktop/TP/Exercice3
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice3
$ java ServeurMultiClient
Serveur 1' coute sur le port :1973
thread started 1
```

Client en fenetre

Je suis le serveur : quelquechose declarer?

### Après envoi de messages



```
MINGW64:/c/Users/Anis SNOUSSI/Desktop/TP/Exercice3
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Exercice3
$ java ServeurMultiClient
Serveur 1' coute sur le port :1973
thread started 1
bonjour gl2, c'est un test
pause d'une seconde
c'est un autre test
pause d'une seconde
```

Client en fenetre

Je suis le serveur : quelquechose declarer?  
La medina de Fes est splendide au couchant.  
Les montagnes de l'Atlas sont impressionnantes.

## Un IRC

Le principe d'un IRC est qu'un ensemble de clients, tous connectés au même serveur, puisse dialoguer. C'est à dire que chaque client voit tout ce que disent les autres clients, et peut lui même parler. Pour distinguer les clients les uns des autres, chacun porte un pseudonyme, qui est repris en entête de chaque message. Exemple :

**PSEUDONYME> ceci est un message**

Le rôle du serveur IRC est donc de recevoir en simultanée les messages de tous les clients et de les renvoyer vers tous les autres clients. Convention : Chaque client doit fournir à la connexion au serveur un Pseudonyme (NickName dans le monde IRC).

Le serveur doit :

- Attendre des connexions
- Créer un nouveau Thread pour chaque nouveau client
- Créer et maintenir une liste des clients connectés (voir ci-dessous)
- Indiquer à tous les clients les nouveaux connectés et ceux qui se sont déconnectés

Le thread serveur (chargé d'un seul client) doit :

- Envoyer un message de bienvenue
- Envoyer la liste de tous les connectés
- Faire passer chaque message provenant du client au serveur, qui à son tour l'envoi à tous les clients connectés

```

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

class ServeurIRC {
    Vector V;
    public static void main(String args[]) {
        int port = 1973;
        if (args.length == 1)
            port = Integer.parseInt(args[0]);
        new ServeurIRC(port);
    }

    public ServeurIRC(int port) {
        V = new Vector();
        try {
            ServerSocket server = new ServerSocket(port);
            System.out.println("Server is listening...");
            while (true) {
                Socket socket = server.accept();
                ajouterClient(new ThreadClient(socket, this));
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    synchronized public void EnvoyerATous(String s) {
        for (int i = 0; i < V.size(); i++) {
            ThreadClient Client = (ThreadClient) V.elementAt(i);
            Client.Envoyer(s);
        }
    }

    public void Envoyer_p(String nom, String s) {
        String name = s.split(":", 2)[0];
        name = name.trim();
        s = s.split(":", 2)[1];
        for (int i = 0; i < V.size(); i++) {
            ThreadClient Client = (ThreadClient) V.elementAt(i);
            if (name.equals(Client.nom())) {
                Client.Envoyer(nom + "> " + s);
            }
        }
    }
}

```

```
public void ajouterClient(ThreadClient c) {
    V.addElement(c);
}

synchronized public void EnvoyerListeClients(PrintWriter out) {
    out.println("::list::");
    out.flush();
    out.println("Liste des connectés:");
    out.flush();
    for (int i = 0; i < V.size(); i++) {
        ThreadClient Client = (ThreadClient) V.elementAt(i);
        out.println("-" + Client.nom());
        out.flush();
    }
    out.println("::end_list::");
    out.flush();
}

synchronized public void SupprimerClient(ThreadClient c) {
    V.removeElement(c);
}

synchronized public boolean exist(String name) {
    for (int i = 0; i < V.size(); i++) {
        ThreadClient Client = (ThreadClient) V.elementAt(i);
        if (name.compareTo(Client.nom()) == 0) {
            return true;
        }
    }
    return false;
}
}
```

**ServeurIRC.java**



```

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

class ThreadClient extends Thread {
    BufferedReader In;
    PrintWriter Out;
    ServeurIRC serveur;
    Socket socket;
    String nom = "???";

    public ThreadClient(Socket socket, ServeurIRC s) {
        try {
            String msg;
            this.socket = socket;
            Out = new PrintWriter(socket.getOutputStream());
            In = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            serveur = s;

            while (true) {
                msg = In.readLine();
                if (msg.contains(" ")) Envoyer("Nom ne doit pas contenir d'espace !");
                else if (serveur.exist(msg) == false && msg.equals("") == false) break;
                else Envoyer("Nom deja existant !");
            }
            nom = msg;
            Envoyer("::ok::");
            Envoyer("Welcome " + nom);
            serveur.EnvoyerATous(nom + " connected");
        } catch (IOException e) {
            e.printStackTrace();
        }
        start();
    }

    public void run() {
        serveur.EnvoyerATous("::update::");
        System.out.println("Client " + nom + " connected");

        try {
            while (true) {
                String msg = In.readLine();
                if (msg.equalsIgnoreCase("list")) serveur.EnvoyerListeClients(Out);
                else if (msg.equalsIgnoreCase("exit")) break;
                else if (msg.contains(":")) serveur.Envoyer_p(nom, msg);
                else serveur.EnvoyerATous(nom + "> " + msg);
            }
        }
    }
}

```

```
Envoyer("::end::");
serveur.EnvoyerATous(nom + " disconnected");
serveur.EnvoyerATous("::update::");
System.out.println("Client " + nom + " disconnected");
serveur.SupprimerClient(this);
In.close();
Out.close();
socket.close();
} catch (IOException e) {
    e.printStackTrace();
}
}

public void Envoyer(String s) { // Envoie vers le client
    Out.println(s);
    Out.flush();
}

public String nom() {
    return nom;
}

}
```

**ThreadClient.java**

```

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

class ClientFenetre extends JFrame implements Runnable, ActionListener {
    TextArea Output;
    TextArea List;
    JTextField Input;
    Socket socket = null;
    BufferedReader in = null;
    PrintWriter out = null;
    boolean connected;
    JLabel welcome_label;
    JButton send_button;
    JButton clear_button;

    public ClientFenetre(InetAddress hote, int port) {
        // mise en forme de la fenetre (frame)
        super("Java Chat");
        setSize(500, 700);
        setLocation(400, 200);
        setLayout(new BorderLayout());
        Container frame_cont = getContentPane();
        // ajout d'un window adapter pour reagir si on ferme la fenetre
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                if (connected == true) {
                    out.println("exit");
                    out.flush();
                } else {
                    setVisible(false);
                    dispose();
                    System.exit(0);
                }
            }
        });
        //North Panel
        JPanel P_north = new JPanel(new FlowLayout(FlowLayout.CENTER));
        welcome_label = new JLabel("Welcome");
        P_north.add(welcome_label);
        P_north.setSize(500, 100);
        frame_cont.add(P_north, BorderLayout.NORTH);
        //South Panel
        JPanel P_south = new JPanel();
        JTextField Input = new JTextField(50);
    }
}

```

```

Input = new JTextField(50);
Input.addActionListener(this);
Input.disable();
send_button = new JButton("send");
send_button.setMargin(new Insets(0, 0, 0, 0));
send_button.addActionListener(this);
clear_button = new JButton("clear");
clear_button.setMargin(new Insets(0, 0, 0, 0));
clear_button.addActionListener(this);
P_south.add(Input);
P_south.add(clear_button);
P_south.add(send_button);
frame_cont.add(P_south, BorderLayout.SOUTH);
//Center Panel
frame_cont.add(List = new TextArea(10, 20), BorderLayout.EAST);
frame_cont.add(Output = new TextArea(), BorderLayout.CENTER);
Output.setEditable(false);
List.setEditable(false);
List.setFocusable(false);
Output.setBackground(new Color(255, 255, 255));
pack();
this.setMinimumSize(new Dimension(getWidth(), getHeight()));
setVisible(true);
Input.requestFocus();
//ouvrir la connexion sur le host/port du serveur
try {
    hote = InetAddress.getLocalHost();
    socket = new Socket(hote, port);
    out = new PrintWriter(socket.getOutputStream());
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    connected = true;
    Output.append("Connected to server...\n");
} catch (IOException e) {
    Output.append("Error: could not connect to server.\n");
    Input.setEnabled(false);
    connected = false;
    e.printStackTrace();
}
//run thread
Thread t = new Thread(this);
if (connected == true) t.start();
}

```

```

public void run() {
    String msg;
    // boucle qui receptionne les messages du serveur
    // et les affiche dans le textarea
    try {
        //Saisie du nom
        while (true) {
            get_name();
            msg = in .readLine();
            if (msg.equals("::ok::")) break;
            Output.append(msg + "\n");
        }
        //message d'accueil
        msg = in .readLine();
        welcome_label.setText(msg);
        Input.enable();
        //reception des messages
        while (true) {
            msg = in .readLine();
            if (msg.equals("::list::")) get_list();
            else if (msg.equals("::update::")) {
                out.println("list");
                out.flush();
            } else if (msg.equals("::end::")) break;
            else Output.append(msg + "\n");
        }
        setVisible(false);
        dispose();
        System.exit(0);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == Input || e.getSource() == send_button) {
        String phrase = Input.getText();
        // envoie au serveur
        // efface la zone de saisie
        if (phrase.equals("") == false) {
            out.println(phrase);
            out.flush();
        }
        Input.setText("");
    }
}

```

```

if (e.getSource() == clear_button) {
    Output.setText(" ");
    Output.setText("");
}
}

private void get_name() {
    String name;
    while (true) {
        name = JOptionPane.showInputDialog(this, "Enter your Name", "Input",
JOptionPane.QUESTION_MESSAGE);
        if (name != null)
            if (name.equals("") == false) break;
    }
    out.println(name);
    out.flush();
}

private void get_list() {
    try {
        List.setText("");
        while (true) {
            String msg = in.readLine();
            if (msg.equals("::end_list::")) break;
            List.append(msg + "\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

protected void finalize() {
    // Fermer ici toute les soquettes
    try {
        out.close(); in.close();
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

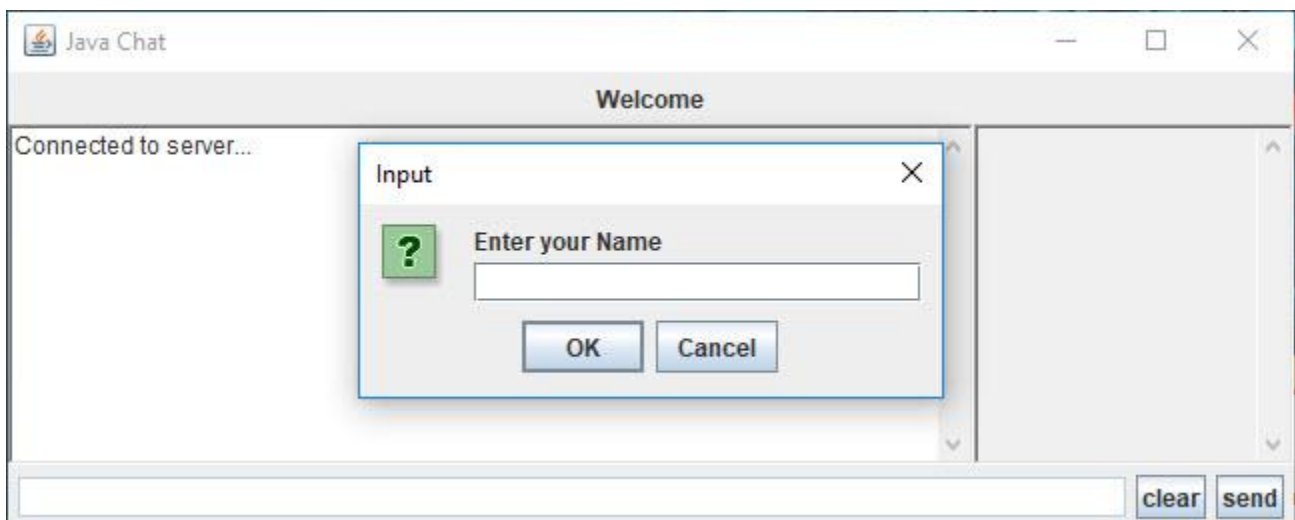
```
public static void main(String[] args) {  
    InetAddress hote = null;  
    int port = 1973; // par default  
    //... ( gestion des parametres )  
    ClientFenetre chatwindow = new ClientFenetre(hote, port);  
}  
  
}
```

### ClientFenetre.java

#### Résultat de l'exécution :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/apps repartis/TP/Exercice4-5  
$ java ServeurIRC  
Server is listening...
```

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/apps repartis/TP/Exercice4-5  
$ java ClientFenetre  
■
```





```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/apps repartis/TP/Exercice4-5
$ java ServeurIRC
Server is listening...
Client anis connected
Client lamine connected
Client anis disconnected
Client lamine disconnected
```