Compte rendu Mini chat multi Client

Réalisé par : Belouch Mustapha

# Présentation :

Cette application « mini chat multi client » permet de faire communiquer plusieurs clients entre eux, à l’aide du protocole de communication TCP.

L’application contient deux dossiers :

* Serveur : contient quatres classes :
* Serveur.java : qui permet de créer et de lancer un Thread de type socketServer

Ecoutant sur un port spécifié.

* ClientServeur.java : permet d’établir la connexion avec une socket client.
* Emission.java : permet d’envoyé des messages à partir du serveur vers un client.
* Reception.java : permet de recevoir les messages envoyés par les clients.
* Client : contient trois classes :
* Client : est une fenêtre qui permet de connecter le client au serveur et aussi d’envoyer des messages à d’autres clients.
* Emission.java : permet d’envoyé des messages au serveur pour qu’il les transmette au client destinataire.
* Reception.java : permet de recevoir les messages envoyés par le serveur.

# Test et description de l’application :

1. Lancement du Serveur :

Tous d’abord on lance le serveur avec un numéro de port (2019) et dans cette classe on appelle la classe ClientServeur.java qui se charge de communiquer le client avec le serveur via les Threads, lorsque un client est connecté à la socket serveur on lance un autre Thread pour écouter d’autres Clients, en lance deux Threads de type Emission et Reception pour communiquer avec le client connecté.

Les classes du dossier Serveurs sont définit comme suit :

Serveur.java

/\* serveur.java

\*dans cette classe qui joue le rôle du serveur

\*on declare une ServerSocket avec un numero de port

\*d'écoute.

\*/

import java.io.\*;

import java.net.\*;

public class Serveur {

public static ServerSocket serverSocket = null;

public static Thread t;

public static PrintWriter[] out = new PrintWriter[10];

public static int nbr=0;

public static String[] nomClient = new String[10];

public static void main(String[] args) {

try {

//creation d'une socket serveur avec un numero de port

serverSocket = new ServerSocket(2019);

System.out.println("Le serveur est à l'écoute du port "+serverSocket.getLocalPort());

t = new Thread(new ClientServeur(serverSocket));

t.start();

} catch (IOException e) {

System.err.println("Le port "+serverSocket.getLocalPort()+" est déjà utilisé !");

}

}

public static void ajoutPrintWriter(PrintWriter sortie,String adress){

//un nouveau client vient de se connecté on garde son adresse

//et le flux de sotire qui est lui associé

out[nbr]=sortie;

nomClient[nbr] = adress;

//on informe les autre qu'un nouveau client s'est connecté

for(int i=0;i<nbr;i++){

Thread e = new Thread(new Emission(out[i],adress+"vient de se connecté "));

e.start();

}

nbr++;

}

public static void redirectMessage(String adrDestin,String msg){

//on cherche le flux de sortie associé a cette adresse pour qu'on puise

//l'envoyé le message

for(int i=0;i<nbr;i++){

if(nomClient[i].equalsIgnoreCase("/"+adrDestin)){

Thread t1 = new Thread(new Emission(out[i],msg));

System.out.println("dddddd "+adrDestin);

t1.start();

}

}

}

}

Emission.java

import java.io.IOException;

import java.io.PrintWriter;

public class Emission implements Runnable {

private PrintWriter out;

private String message = null;

//le constucteur prend en charge le flux de sortie lié au client destinataire

//et le message

public Emission(PrintWriter out,String message) {

this.out = out;

this.message = message;

}

public void run() {

//envoi du message au client

out.println(message);

out.flush();

}

}

Reception.java

import java.io.BufferedReader;

import java.io.IOException;

import java.net.Socket;

import java.io.InputStreamReader;

import java.net.InetAddress;

import java.io.PrintWriter;

public class Reception implements Runnable {

private BufferedReader in;

private PrintWriter out;

private String message = null, login = null;

private Socket socket;

public Reception(Socket socket){

this.socket = socket;

}

public void run() {

try{

//on enregistre le flux de sortie lié et on l'ajoute à la table

//pour le récupérer lorsqu'un autre client l'envoie un message

out = new PrintWriter(socket.getOutputStream());

InetAddress adr = socket.getInetAddress();

String adress = adr.toString();

Serveur.ajoutPrintWriter(out,adress);

while(true){

try {

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

message = in.readLine();

//on lit le message du client et on prend l'adresse du destinataire

//car le message contient l'adresse du destinataire suivi d'un delimiteursuivi du message à envoyer.

String[] data= message.split("°");

System.out.println("port ="+socket.getLocalPort()+" : "+message);

//une fonction de la classe Serveur se charge de l'émission du message à sa destination

Serveur.redirectMessage(data[0],data[1]);

} catch (IOException e) {

e.printStackTrace();

}

}

} catch(IOException ex){

ex.printStackTrace();

}

}

}

ClientServeur.java

import java.io.IOException;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class ClientServeur implements Runnable {

private Socket socket = null;

private ServerSocket socketServer;

private PrintWriter out = null;

private Thread t1, t2;

public ClientServeur(ServerSocket socketServer){

this.socketServer = socketServer;

}

public void run() {

try {

while(true){

//on accepte la demande de conexion d'un client

socket = socketServer.accept();

// puis on lance un thread pour recevoir les messages de ce client

t1 = new Thread(new Reception(socket));

t1.start();

// et un autre pour envoyer des messages a ce client

out = new PrintWriter(socket.getOutputStream());

t2 = new Thread(new Emission(out,"vous etes connecté au serveur"));

t2.start();

}

}catch (IOException e) {

System.err.println("le client est déconnecté ");

}

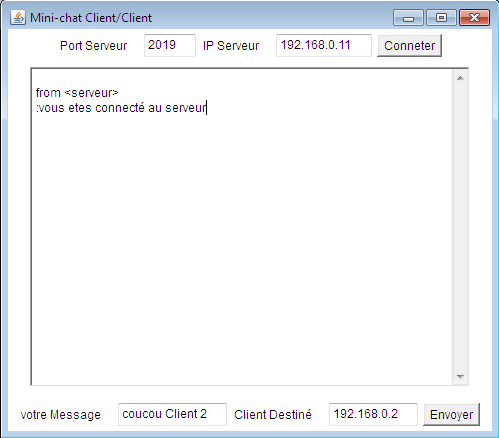
}

}

1. Lancement d’un client :

Le lancement de l’interface du client nous permet de connecter à un serveur, puis de communiquer avec un autre client connecté via son adresse ip.

L’interface client est la suivante :



Quand un nouveau client se connecte au serveur un message contenant l’adresse ip de ce client est diffusé pour informer tous les clients qui sont déjà connectés qu’un nouveau client est connecté.

Via cette adresse ip les clients peuvent se communiquer.

Les classes de dossier client sont définit comme suit :

Client.java

/\*\*

\* @(#)Client.java

\* @author : Belouch Mustapha

\* @version 1.00 2010/12/14

\*/

/\*on a utilisé les interfaces graphique du package awt

\*et a l'aide de l'action listner on connecte le client a un serveur avec la methode connectActionPerformed

\*et on envoie des messages à l'aide de la methode envoyerActionPerformed

\*

\*

\*/

import java.net.UnknownHostException;

import java.io.IOException;

import java.net.SocketException;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import java.net.Socket;

public class Client extends Frame{

public static Client client;

public static Socket socket = null;

public static Thread t1,t2;

public Client() {

initComponents();

setTitle("Mini-chat Client/Client" );

setSize(500,440);

client=this;

}

public void initComponents(){

//initialisation des composantes de l'interface graphiques

connect = new Button();

envoyer = new Button();

portClient = new TextField();

portServeur = new TextField();

ipServeur = new TextField();

nomClient = new TextField();

nomClientDes = new TextField();

text = new TextArea();

scroll = new ScrollPane();

message = new TextField();

label1 = new Label();

label2 = new Label();

label3 = new Label();

label4 = new Label();

label5 = new Label();

label6 = new Label();

label7 = new Label();

label8 = new Label();

label1.setText("Port Client");

label2.setText("Nom");

label3.setText("Port Serveur");

label4.setText("IP Serveur");

label5.setText("Client Destiné");

label6.setText("Messages Reçus");

label7.setText("votre Message");

label8.setText("");

text.setRows(20);

text.setColumns(60);

scroll.setComponentZOrder(text,0);

envoyer.setLabel("Envoyer");

connect.setLabel("Conneter");

//ajout d'un action listner lié au bouton envoyer

envoyer.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ev){

envoyerActionPerformed(ev);

}

});

//ajout d'un action listner lié au bouton connecter

connect.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ev){

connectActionPerformed(ev);

}

});

//ajout d'un action pour fermer la fenetre

Closer close = new Closer();

addWindowListener(close);

//on declare ici un groupeLayout pour groupé tous les zones de textes et labels

Panel p1 = new Panel();

Panel p2 = new Panel();

Panel p3 = new Panel();

Panel p4 = new Panel();

Panel p5 = new Panel();

p1.setLayout(new FlowLayout());

portServeur.setText("2018");

ipServeur.setText("192.168.0.11");

message.setText("coucou Serveur");

nomClientDes.setText("192.168.0.2");

// ajout des panel qui contient des composants à la fenetre

p1.add(label1);

p1.add(portClient);

p1.add(label2);

p1.add(nomClient);

p2.add(label3);

p2.add(portServeur);

p2.add(label4);

p2.add(ipServeur);

p2.add(connect);

p3.add(text);

p4.add(label7);

p4.add(message);

p4.add(label5);

p4.add(nomClientDes);

p4.add(envoyer);

add("North",p2);

add("Center",p3);

add("South",p4);

pack();

show();

}

//definition du code qui vas se generer lors de l'appuit sur connect

public void connectActionPerformed(ActionEvent ev){

String adrServer = ipServeur.getText();

int portServer = Integer.parseInt(portServeur.getText());

String localNom = nomClient.getText();

System.out.println("port "+portServer+" est destiné a : "+adrServer);

try {

System.out.println("Demande de connexion");

//on lance une socket client demandant une connection avec la socket serveur

socket = new Socket(adrServer,portServer);

System.out.println("Connexion établie avec le serveur, authentification :"); // Si le message s'affiche c'est que je suis connecté

// on lance un thread qui s'occupe de recevoir les messages venants du serveur

t2 = new Thread(new Reception(socket));

t2.start();

} catch (UnknownHostException e) {

System.err.println("Impossible de se connecter à l'adresse "+socket.getLocalAddress());

} catch (IOException e) {

System.err.println("Aucun serveur à l'écoute du port "+socket.getLocalPort());

}

}

//definition du code qui vas se generer lors de l'appuit sur envoyer

private void envoyerActionPerformed(ActionEvent ev){

//lorsqu'on click sur on envoyer on recupère les données des zones de textes

// et on les transmettres aux serveurs

String nomDestine = nomClientDes.getText();

String msg = " "+message.getText();

String data =nomDestine+"°"+msg;

message.setText("");

t1 = new Thread(new Emission(socket,data));

t1.start();

}

//la fonction principal main

public static void main (String[] args) {

EventQueue.invokeLater(new Runnable(){

public void run(){

new Client().setVisible(true);

}

});

}

class Closer extends WindowAdapter {

public void windowClosing (WindowEvent event) {

System.exit (0);

}

}

public static void receptionMessage(String hoste,String msg){

//on affiche le message recu dans la zone textField

client.text.append("\n from <"+hoste+">\n :"+msg);

}

//ici on declare les variables de l'interface graphique

private Button connect;

private TextField portServeur;

private TextField portClient;

private TextField ipServeur;

private TextField nomClient;

private TextField nomClientDes;

private TextField message;

private Label label1;

private Label label2;

private Label label3;

private Label label4;

private Label label5;

private Label label6;

private Label label7;

private Label label8;

private TextArea text;

private Button envoyer;

private ScrollPane scroll;

}

Emission.java

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Scanner;

import java.net.Socket;

public class Emission implements Runnable {

private PrintWriter out;

private String login = null, message = null;

private Scanner sc = null;

private Socket socket;

public Emission(Socket socket,String message) {

this.socket = socket;

this.message = message;

}

public void run() {

try{

out = new PrintWriter(socket.getOutputStream());

System.out.println("Votre message :");

out.println(message);

out.flush();

}catch(IOException ex){

System.err.println("Le serveur distant s'est déconnecté !");

}

// sc = new Scanner(System.in);

// while(true){

// }

}

}

Reception.java

import java.io.\*;

import java.io.IOException;

import java.net.Socket;

public class Reception implements Runnable {

private BufferedReader in;

private String message = null;

private Socket socket;

public Reception(Socket socket){

this.socket = socket;

}

public void run() {

try{

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

while(true){

try {

message = in.readLine();

Client.receptionMessage("serveur",message);

System.out.println("Le serveur vous dit :" +message);

} catch (IOException e) {

e.printStackTrace();

}

}

}catch(IOException ex){

System.err.println("erreur");

}

}

}

