Tugas MK Sistem Terdistribusi

Aplikasi Chatting Server Client dengan Pemrograman Socket Java

Oleh:

Ninan Kara Gicha Nasution 07/250814/TK/32408
Ega Reti Effendhy 07/252241/TK/32820

UNIVERSITAS GADJAH MADA
YOGYAKARTA
2009

PENGANTAR

Biasanya, server berjalan pada komputer tertentu dan memiliki sebuah socket yang terikat pada nomor port tertentu. Server hanya menunggu, mendengarkan socket untuk seorang klien untuk membuat permintaan sambungan.
Di sisi klien: klien mengetahui nama host dari mesin server yang sedang berjalan dan nomor port di mana server mendengarkan. Untuk membuat permintaan sambungan, klien mencoba untuk bertemu dengan server pada mesin server dan port. Klien juga perlu mengidentifikasi dirinya ke server sehingga mengikat ke nomor port lokal yang akan digunakan selama hubungan ini. Hal ini biasanya diberikan oleh sistem.
Jika semuanya berjalan dengan baik, server menerima koneksi. Setelah diterima, server mendapatkan soket baru terikat port lokal yang sama dan juga memiliki titik akhir yang di atur ke alamat dan port dari klien. Diperlukan soket baru sehingga dapat terus mendengarkan permintaan sambungan socket sementara mengurus kebutuhan klien yang terhubung.
Di sisi client, jika koneksi diterima, socket berhasil dibuat dan klien dapat menggunakan soket untuk berkomunikasi dengan server. Klien dan server dapat berkomunikasi dengan menulis atau membaca dari soket

PENJELASAN

Berikut ini adalah source code, penjelasan dan hasil eksekusi aplikasi ini.

server.java

Pada server, pertama dilakukan pembukaan port 9999 dan menunggu koneksi.

```
try
            {
                  serverSocket = new ServerSocket(9999);
            }
            catch(IOException e)
            {
                  System.out.println("IO "+e);
            }
```

Kemudian dibuatlah objek dari kelas cThread sehingga kelas tersebut dapat dijalankan dengan method start().

```
while (true)
            {
                  try
                  {

                        clientSocket = serverSocket.accept();
                        cThread s = new cThread(clientSocket);

                        clients.add(s);
```

```
                        s.start();
                }
                catch (IOException e)
                {
                        System.out.println("IOaccept "+e);

                }
        }
```

Jika koneksi sudah terbentuk dari permintaan login, server akan memeriksa nickname karena harus unik. Jika nickname yang sudah dipakai, server akan mengirim permintaan nickname yang baru.
Jika nickname tersedia , server akan mengirim daftarnya ke semua client. Selanjutnya, jika banyak client yang terhubung dengan server dan ada yang mengirim pesan umum, maka server akan meneruskannya ke semua client. Sedangkan jika client menerima pesan privat, maka server hanya akan meneruskannya ke tujuannya.
Jika server menerima perintah log out, maka dihapuslah nickname tesebut dari list dan mengirim kembali up date list client terbaru ke semua client. Server bekerja dengan Thread, sehingga dapat menangani banyak client.

cThread.java

Pada program di bawah, cThread membuat soket yang digunakan untuk berhubungan dengan server. Input dari user akan dibaca berupa standart output stream lalu meneruskan pesan tersebut ke server dengan menuliskannya pada soket. Server akan menampilkan input dari soket ke client. Program client akan membaca dan menampilkan pesan dari server.

Selanjutnya terdapat perintah untuk mendapatkan output stream dari soket dan membuka *PrintWriter* di dalamnya. Kemudian dilakukan perintah untuk mendapatkan input stream dari soket dan membuka *BufferedReader* di dalamnya. Untuk mengirim data melaluii soket ke server, cThread cukup menuliskannya pada *PrintWriter*. Untuk mendapat respon dari server, cThread membacanya dari *BufferedReader*.

```
try
        {

                out = new PrintWriter(clientSocket.getOutputStream(),
true);
                in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        }
        catch (Exception e)
        {
                System.out.println(e);
        }
    }
```

Perintah-perintah selanjutnya merupakan perintah untuk merespon tindakan user yang melakukan login, log out, menulis pesan public dan privat. Dimulai dengan kode di bawah ini yang melakukan pengecekan akan adanya pesan (msg) pada *BufferedReader* (i). Jika benar , lalu dicek juga apakah pesan yang diterima diawali dengan ”Login”, jika ya, maka akan dipanggil method login().

```
while (true)
                    {
                        String msg = in.readLine();
                          System.out.println(msg);


                    if (msg.startsWith("Login"))
                    {
                                login(msg);
                    }
```

Jika pesan yang diterima adalah pesan log out maka koneksi dengan client akan diputus dengan perintah close().

```
else if (msg.equals("Logout")){
                          if (connected)                {
                                connected = false;
                                int k = server.clients.indexOf(this);
                                server.clients.remove(this);
                                      sendList();
                                out.println("OK");
                                out.close();
                                in.close();
                                clientSocket.close();
                                return;                        }
```

Jika pesan diawali dengan post maka server akan mengirim ke pesan ke semua client dan jika diawali dengan privatepost maka pesan hanya akan dikirim ke client tujuan.

```
else if (msg.startsWith("Post "))              {

                        for (int i = 0; i < server.clients.size() ; i ++)
            {

                              cThread t = (cThread)server.clients.get(i);

                              if (t.connected)                          {

                                    t.send("Recieve     "+     nick+":     "
+msg.substring(5, msg.length()));

                              }                     }               }

                  else if (msg.startsWith("PrivatePost "))            {

                        StringTokenizer          st          =          new
StringTokenizer(msg.substring(12,msg.length()),", ");

                        String message = st.nextToken();

                        String to = st.nextToken();

                  boolean success = false;

                        for (int i = 0; i < server.clients.size() ; i ++)
            {

                              cThread t = (cThread)server.clients.get(i);

                              if (t.nick.equals(to))
      {

                                    t.send("PrivateRecieve  "+  nick+":  "  +
message);

                                    success = true;

                                    break;
```

Kelas ini memiliki method bernama login() yang digunakan untuk memeriksa nickname yang digunakan client agar tetap unik dengan perulangan berikut.

```
for (int i = 0;i<server.clients.size();i++)        {

            if (server.clients.get(i) != null)         {

                        System.out.println(msg.substring(7, msg.length()));
```

```
                        cThread temp = (cThread)server.clients.get(i);

                  if ((temp.nick).equals(msg.substring(7, msg.length())))
      {

                  exists = true;

                        break;
```

Kelas ini juga memiliki methode send() yang digunakan untuk mengirim list client yang terkoneksi.

```
for (int i = 0; i < server.clients.size() ; i ++)         {

            cThread t = (cThread)server.clients.get(i);

            if (t.connected)        {

                  t.send(list);           }
```

Pada sisi client, ada 4 kelas yang dibentuk:

1. readFromServer.java – membaca koneksi dengan server
2. client.java – kelas untuk membuat GUI dari aplikasi chat ini dengan menggunakan JFrame milik Netbeans
3. userInput.java – untuk menuliskan pesan chatting
4. clientRunner.java – untuk menjalankan GUI dari client.java, karena kelas ini bertugas untuk membuat objek baru dari kelas client tersebut.

readFromServer.java

Progam ini berisi pembacaan client terhadap server. Di antaranya adalah, menerima pesan umum, pesan privat, mengetahui list terakhir client yang terhubung ke server, dan peringatan bahwa nickname yang diinginkan sudah dipakai sehingga harus menggunakan nickname lain. Hampir sama dengan perintah-perintah pada cThread.java., kelas ini juga menampilkan list client, pesan yang diterima baik yang public maupun yang privat dengan perintah di bawah ini.

```
System.out.print("List updated: New names: ");
                        for (int i = 0; i < client.list.size();i++)
                        {
                              System.out.print(client.list.get(i) + " ");
                        }
                        System.out.println();
                  }

                  else if (s.startsWith("Recieve"))
                  {
                        client.mainText.setText(client.mainText.getText() +
"\n" + s.substring(8,s.length()));
```

```
	client.mainText.setCaretPosition(client.mainText.getText().length());
			}

			else if (s.startsWith("PrivateRecieve"))
			{
				client.mainText.setText(client.mainText.getText() +
"\n" + "Privat Messages: " + s.substring(14,s.length()));

	client.mainText.setCaretPosition(client.mainText.getText().length());
			}
```

Untuk menjalankan client, langkah pertama yang harus dilakukan adalah menjalankan servernya dulu, setelah itu run clientRunner.java untuk membuat objek baru. Jika ingin membuka lebih dari 1 client, run lagi clientRunner.java untuk membuat objek lagi.

clientRunner.java

Kelas ini hanya bertugas untuk membuat objek baru dari kelas client.java. Jika ketika ingin mencoba memunculkan lebih dari 1 client, kita tinggal mengompile kelas ini. Untuk menjalankan GUI dari program chat ini, kita harus membuat objek terlebih dahulu, otomatis kelas inilah yang di run terlebih dahulu setelah menjalankan server.

userInput.java

Kelas ini merupakan turunan dari kelas Thread. Seperti tugas yang lalu mengenai fungsi run, dalam kelas ini dideklarasikan method bernama run(). Tujuan dari method ini adalah mengatur input dan output text dari ke server. Dalam method run() ini berisi pemanggilan fungsi read() dan send(), 2 method ini dideklarasikan pada kelas client.java. Tugas dari method send() adalah mengeluarkan input message dari client.

client.java

Bisa dibilang kelas ini merupakan program dari client yang utama. Pada kelas ini dibuat GUI sebagai interaksi untuk melakukan chatting dengan antar client. GUI dari program ini dibuat dengan bantuan Netbeans dengan JFrame-nya. Jadi kebanyakan dari tampilan source code merupakan hasil dari generate Java Swing tersebut. Sedangkan logicnya hanya berada pada aksi button, input text dan menu-menu terkait. Yang akan dibahas hanya bagian-bagian pentingnya saja, berikut penjelasannya

```
	private void nickListMouseClicked(java.awt.event.MouseEvent evt)
	{
		if (connected && (!nickList.getSelectedValue().equals(nick)))
		{
			String msg =  JOptionPane.showInputDialog(null, "server:");
			if (msg != null)
			{
			send("PrivatePost"+msg+","+nickList.getSelectedValue());
			}
			System.out.println(nickList.getSelectedValue());
		}
```

```
      }
```

Perintah diatas merupakan perintah disaat kita melakukan private message kepada client tertentu. Ketika event bernilai true, maka akan muncul dialog yang bertanya server mana yang dituju. Jika benar, akan tertulis Private Post dan pesannya pada server.

```
private void jMenuItem1ActionPerformed2(java.awt.event.ActionEvent evt)
    {

      send("Logout");
            client.logout = true;
            System.exit(0);
    }
```

Perintah diatas untuk action ketika menu Disconnect diklik, akan muncul tulisan Logout pada server dan system client langsung berhenti.

```
    private void jMenuItem1ActionPerformed3(java.awt.event.ActionEvent evt)
    {
      String s;
      s = "Login and Connect. Connect to localhost then input your
username\n";
      JOptionPane.showMessageDialog(             this,             s,"Usage",
JOptionPane.INFORMATION_MESSAGE );
    }
```

Perintah diatas ketika menu Help lalu Usage diklik. Muncul popup message yang berisi seperti string yang ada dalam variable s diatas.

```
  static String server;
      private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt)
    {
            setTitle("Connecting ...");
      logout = false;
            sinput = new readFromServer(this);
            uinput = new userInput();
            cSocket = null;
            out = null;
            in = null;
            boolean error;
            error = false;
            server = JOptionPane.showInputDialog("Server: ", "localhost");
            try
            {
                  cSocket = new Socket(server,9999);
                  out = new PrintWriter(cSocket.getOutputStream(), true);
```

```
                  in            =            new            BufferedReader(new
InputStreamReader(cSocket.getInputStream()));
                  uinput.start();
                  sinput.start();
            }
            catch(UnknownHostException e)
            {
            JOptionPane.showMessageDialog(     this,     "No     server     is
found","ERROR", JOptionPane.ERROR_MESSAGE );
                  System.out.println("Host Error" + e);
                  setTitle("Simple  Java  Chat  -  Cannot  Connect  Please  Try
another server");
                  error = true;
            }
            catch (IOException e)
            {
                  System.out.println("IOException" + e);
            }
            if (!error)
            {
            nick = null;
                  nick = JOptionPane.showInputDialog(null, "NickName: ");

                  while(nick.contains(";"))
                  {
                        nick     =     JOptionPane.showInputDialog(null,"Input
Nickname\";\".Cannot be null.");
                  }
                  send("Login: "+nick);
                  if (nick != null)
                  {
                        jMenuItem1.setEnabled(false);
                        jMenuItem2.setEnabled(true);
                  }
            }
    }
```

Sedangkan perintah diatas mengatur client ketika akan connect. Dimulai ketika menu Connect pada menubar Login di klik. Awalnya client harus membaca dulu, apakah server sudah jalan atau belum, jika belum tidak akan bisa tersambung. Jika sudah berjalan servernya, akan muncul popup yang menanyakan alamat server, dalam hal ini kita isi dengan localhost. Lalu akan ditanyakan nickname untuk login. Saat kita mencoba tersambung dengan server, cSocket akan membuat objek baru yang mengakses port 9999. Namun, ketika server tidak ditemukan, akan muncul popup message yang meminta kita untuk menuliskan alamat server yang tepat.

```
void sendInput()
    {
    if (!connected)
      {
```

```
            JOptionPane.showMessageDialog(  this,  "Not  connected!  Actions  -
Connect","Error", JOptionPane.ERROR_MESSAGE );
            inputText.setText("");
      }
            else             if(inputText.getText().equals("")             ||
inputText.getText().equals("\n") ||  inputText.getText()== null  )
      {
            inputText.setText("");
      }
      else
      {
       send("Post " + replace(inputText.getText(),"\n"," "));
       inputText.setText("");
      }
    }
```

Fungsi diatas merupakan method sendInput() yang merupakan method untuk membaca input yang diinput oleh client. Method ini digunakan untuk memasukkan text ke server, lalu server melempar string tersebut ke client yang dikehendaki.

PasswordDemo,java

Pada kelas ini dilakukan request password kepada client. Password yang dimasukkan client akan dicek benar atau tidak dengan method isPasswordCorrect. Jika password salah, maka client tidak dapat melakukan koneksi ke server.

```
private static boolean isPasswordCorrect(char[] input) {
        boolean isCorrect = true;
        char[] correctPassword = { 't', 'e', 's', 't', 'i', 'n', 'g' };

        if (input.length != correctPassword.length) {
            isCorrect = false;
        } else {
            isCorrect = Arrays.equals (input, correctPassword);
        }

        //Zero out the password.
        Arrays.fill(correctPassword,'0');

        return isCorrect;
```

SOURCE CODE

server.java

```
package components;

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




class server
{

	static Vector clients;
	static Socket clientSocket;

	public static void main(String args[])
	{

		clients = new Vector();

		clientSocket = null;


		ServerSocket serverSocket = null;

		try
		{
			serverSocket = new ServerSocket(9999);
		}
		catch(IOException e)
		{
			System.out.println("IO "+e);
		}


		while (true)
		{
			try
			{

				clientSocket = serverSocket.accept();
				cThread s = new cThread(clientSocket);

				clients.add(s);
				s.start();
```

```
                  }
                  catch (IOException e)
                  {
                        System.out.println("IOaccept "+e);

                  }
            }
      }
}
```

cThread.java

```
package components;

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


public class cThread extends Thread
{

      String nick;
      Boolean connected;
      Socket socket;
      PrintWriter out;
      BufferedReader in;
      Socket clientSocket;

      cThread(Socket s)
      {
            super("cThread");

            connected = false;

            nick = "";


            clientSocket = s;
            try
            {

                  out = new PrintWriter(clientSocket.getOutputStream(),
true);
                  in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            }
            catch (Exception e)
            {
```

```
            System.out.println(e);
      }
}


public boolean equals(cThread c)
{
      return (c.nick.equals(this.nick));
}


synchronized void send(String msg)
{
            out.println(msg);
}



void listen()
{
      try
      {

            while (true)
            {
                String msg = in.readLine();
                  System.out.println(msg);

            if (msg.startsWith("Login"))
            {
                        login(msg);
            }

            else if (msg.equals("Logout"))
            {


                  if (connected)
                  {
                        connected = false;
                        int k = server.clients.indexOf(this);
                        server.clients.remove(this);
                              sendList();
                        out.println("OK");
                        out.close();
                        in.close();
                        clientSocket.close();
                        return;
                  }
```

```
                        else
                        {
                              send("Not Logged in !!");
                        }

                  }


                  else if (msg.startsWith("Post "))
                  {

                        for (int i = 0; i < server.clients.size() ; i ++)
                        {
                              cThread t = (cThread)server.clients.get(i);
                              if (t.connected)

                              {
                                    t.send("Recieve "+ nick+": "
+msg.substring(5, msg.length()));
                              }
                        }
                  }


                  else if (msg.startsWith("PrivatePost "))
                  {

                        StringTokenizer st = new
StringTokenizer(msg.substring(12,msg.length()),", ");

                        String message = st.nextToken();
                        String to = st.nextToken();


                              boolean success = false;


                        for (int i = 0; i < server.clients.size() ; i ++)
                        {
                              cThread t = (cThread)server.clients.get(i);
                              if (t.nick.equals(to))
                              {
                                    t.send("PrivateRecieve "+ nick+": " +
message);
                                    success = true;
                                    break;
                              }
                        }
```

```
                  if (!success)
                  {
                        send("Error!");
                  }
            }

            else
            {
                  send(msg);
            }
            }
      }
      catch (SocketException e)
      {

                  if (connected)
                  {
                        try
                        {
                              connected = false;
                              int k = server.clients.indexOf(this);
                              server.clients.remove(this);
                                    sendList();
                                    out.close();
                                    in.close();
                              clientSocket.close();
                              return;
                              }
                        catch (Exception d)
                        {
                              return;
                        }
                  }
      }
      catch (Exception e)
      {
            System.out.println(e);
      }
}


public void run()
{
      listen();
}

boolean login(String msg)
```

```
{

    if (connected)
    {
      out.println("Already Connected!");
      return true;
    }

      boolean exists = false;
      System.out.println("Login" + msg.substring(5, msg.length()));

    for (int i = 0;i<server.clients.size();i++)
    {
      if (server.clients.get(i) != null)
        {

                System.out.println(msg.substring(7, msg.length()));
                cThread temp = (cThread)server.clients.get(i);
            if ((temp.nick).equals(msg.substring(7, msg.length())))
            {
                        exists = true;
                break;
            }

        }
      }


      if (exists)
      {
            out.println("New Nick");
      }
      else
      {
            connected = true;
            nick = msg.substring(7,msg.length());
        sendList();
      }
    return true;
}


void sendList()
{

      String list = "";
      System.out.println(server.clients.size());
      if (server.clients.size() == 0)
      {
```

```
                return;
            }


          for (int i = 0;i<server.clients.size();i++)
          {
            cThread temp = (cThread)server.clients.get(i);
            if (server.clients.get(i) != null)
              {
                        if (connected)
                        {
                              list =temp.nick + "," + list  ;
                        }
              }
            }

            list = "List " +list.substring(0,list.length() -1) +";";


          for (int i = 0; i < server.clients.size() ; i ++)
          {
            cThread t = (cThread)server.clients.get(i);
            if (t.connected)
            {
                  t.send(list);
            }
          }
      }


      static String replace(String str, String pattern, String replace)
      {
            int s = 0;
            int e = 0;
            StringBuffer result = new StringBuffer();
      while ((e = str.indexOf(pattern, s)) >= 0)
      {
            result.append(str.substring(s, e));
                result.append(replace);
                s = e+pattern.length();
      }
      result.append(str.substring(s));
      return result.toString();
    }
}
```

readFromServer.java

```
package components;
```

```
import java.util.*;
import javax.swing.*;


public class readFromServer extends Thread
{

	client c;

	readFromServer(client cc)
	{
		c = cc;
	}

	public void run()
	{
		String s;
		while (true)
		{

			if (client.logout)
			{
				return;
			}


			s = client.read();

			if (s.startsWith("List"))
			{

				client.mainText.setText("Connected as " +
client.nick);
				c.setTitle("Simple Java Chat - " + c.nick + " -
Connected to " + c.server);

				client.connected = true;
				client.list.clear();
				String nextNick = "";


				StringTokenizer st = new
StringTokenizer(s.substring(5,s.length()),", ");

				String temp = null;
				while(st.hasMoreTokens())
				{
					temp = st.nextToken();
```

```
                        client.list.addElement(replace(temp,";",""));
                  }


                  System.out.print("List updated: New names: ");
                  for (int i = 0; i < client.list.size();i++)
                  {
                        System.out.print(client.list.get(i) + " ");
                  }
                  System.out.println();
            }

            else if (s.startsWith("Recieve"))
            {
                  client.mainText.setText(client.mainText.getText() +
"\n" + s.substring(8,s.length()));


      client.mainText.setCaretPosition(client.mainText.getText().length());
            }

            else if (s.startsWith("PrivateRecieve"))
            {
                  client.mainText.setText(client.mainText.getText() +
"\n" + "Privat Messages: " + s.substring(14,s.length()));

      client.mainText.setCaretPosition(client.mainText.getText().length());
            }

            else if (s.startsWith("NewNick"))

            {
                  client.mainText.setText("");
                  String newnick =  JOptionPane.showInputDialog(null,
"New nick:");
                  client.connected = false;

                  client.jMenuItem1.setEnabled(true);
                  client.jMenuItem2.setEnabled(false);

                  if (newnick != null)
                  {

                        client.nick = newnick;
                        client.jMenuItem1.setEnabled(false);
                        client.jMenuItem2.setEnabled(true);
                        client.send("Login: "+newnick);
                  }
            }
```

```
                System.out.println(s);
            }
      }

      static String replace(String str, String pattern, String replace)
      {
            int s = 0;
            int e = 0;
            StringBuffer result = new StringBuffer();
      while ((e = str.indexOf(pattern, s)) >= 0)
      {
            result.append(str.substring(s, e));
                result.append(replace);
                s = e+pattern.length();
      }
      result.append(str.substring(s));
      return result.toString();
    }
}
```

userInput.java

```
import java.io.*;

public class userInput extends Thread
{
      public void run()
      {
            BufferedReader kin = new BufferedReader(new
InputStreamReader(System.in));
            while(true)
            {
                  if (client.logout)
                  {
                        return;
                  }

                  try
                  {
                        String command = kin.readLine();
                        if (command.equals("Logout"))
                        {
                              client.send(command);

                              String response = client.read();
                              client.logout = true;
                              return;
                        }
                        else
```

```
                        {
                              client.send(command);
                        }
                  }
                  catch (Exception e)
                  {
                  }
            }
      }
}
```

clientRunner.java

```
class clientRunner
{

      public static void main(String args[])
      {
            client c = new client();
            c.run();
      }
}
```

client.java

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


class client extends JFrame
{
      static boolean connected;
      static boolean logout;
      static Socket cSocket;
      static PrintWriter out;
      static BufferedReader in;
      static userInput uinput;
      static readFromServer sinput;
      static DefaultListModel list;


      void run()
      {

            setTitle("Simple Java Chat");
```

```
            enter = false;
            connected = false;

            jScrollPane1 = new
javax.swing.JScrollPane(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.
HORIZONTAL_SCROLLBAR_NEVER);
        inputText = new javax.swing.JTextArea();
        sendButton = new javax.swing.JButton();
        jScrollPane2 = new
javax.swing.JScrollPane(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.
HORIZONTAL_SCROLLBAR_NEVER);
        mainText = new javax.swing.JTextArea();
        jScrollPane3 = new
javax.swing.JScrollPane(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.
HORIZONTAL_SCROLLBAR_NEVER);


            list = new DefaultListModel();
            list.addElement("Not Connected");


            nickList = new JList(list);
        jMenuBar1 = new javax.swing.JMenuBar();


        jMenu1 = new javax.swing.JMenu();
        jMenu2 = new javax.swing.JMenu();
        jMenuItem1 = new javax.swing.JMenuItem();
            jMenuItem2 = new javax.swing.JMenuItem();
            jMenuItem3 = new javax.swing.JMenuItem();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        inputText.setColumns(20);
        inputText.setRows(5);
        jScrollPane1.setViewportView(inputText);


        sendButton.setText("Send");



        sendButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                sendButtonActionPerformed(evt);
            }
        });
```

```
inputText.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        inputTextKeyReleased(evt);
    }
});


nickList.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        nickListMouseClicked(evt);
    }
});


inputText.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        inputTextKeyReleased(evt);
    }
});
mainText.setColumns(20);
mainText.setEditable(false);
mainText.setRows(5);
mainText.setLineWrap(true);
inputText.setLineWrap(true);
jScrollPane2.setViewportView(mainText);


    jScrollPane3.setViewportView(nickList);
    //menu
jMenu1.setText("Login");
    jMenu2.setText("Help");
jMenuItem1.setText("Connect");
jMenuItem2.setText("Disconnect");
jMenuItem3.setText("Usage");
    jMenuItem2.setEnabled(false);
jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem1ActionPerformed(evt);
    }
});
jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem1ActionPerformed3(evt);
    }
});
jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem1ActionPerformed2(evt);
    }
```

```
        });
        jMenu1.add(jMenuItem1);
            jMenu1.add(jMenuItem2);
            jMenu2.add(jMenuItem3);

            jMenuBar1.add(jMenu1);
        jMenuBar1.add(jMenu2);

        setJMenuBar(jMenuBar1);

            make();


      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
            mainText.setFont(new Font("Serif", Font.ITALIC, 16));
      }

      void make()
      {
                    javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILI
NG)
                    .addComponent(jScrollPane2,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE)
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G, false)
                    .addComponent(jScrollPane3)
                    .addComponent(sendButton,
javax.swing.GroupLayout.DEFAULT_SIZE, 141, Short.MAX_VALUE))
                .addContainerGap())
        );
        layout.setVerticalGroup(
```

```
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addComponent(jScrollPane3,
javax.swing.GroupLayout.DEFAULT_SIZE, 565, Short.MAX_VALUE)
                    .addComponent(jScrollPane2,
javax.swing.GroupLayout.DEFAULT_SIZE, 565, Short.MAX_VALUE))
                .addGap(9, 9, 9)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G, false)
                    .addComponent(sendButton,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.Alignment.TRAILING))
                .addContainerGap())
        );

        pack();


        setVisible(true);
      }


      private void nickListMouseClicked(java.awt.event.MouseEvent evt)
      {

            if (connected && (!nickList.getSelectedValue().equals(nick)))
            {

                  String msg =  JOptionPane.showInputDialog(null, "server:
");
                  if (msg != null)
                  {

                        send("PrivatePost " + msg + ",
"+nickList.getSelectedValue());
                  }

                  System.out.println(nickList.getSelectedValue());
            }
      }
```

```
      static boolean enter;
    private void inputTextKeyReleased(java.awt.event.KeyEvent evt)
    {

        if(evt.getKeyCode() == 10)
        {
            if (enter)
            {

                  sendInput();
                  enter = false;
            }
            else
            {

                  enter = true;
            }
        }
    }
    private void jMenuItem1ActionPerformed2(java.awt.event.ActionEvent evt)
    {

      send("Logout");
            client.logout = true;
            System.exit(0);
    }

    private void jMenuItem1ActionPerformed3(java.awt.event.ActionEvent evt)
    {
      String s;
      s = "Login and Connect. Connect to localhost then input your
username\n";
      JOptionPane.showMessageDialog( this, s,"Usage",
JOptionPane.INFORMATION_MESSAGE );
    }

    static String server;

      private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt)
    {

      setTitle("Connecting ...");
      logout = false;

            sinput = new readFromServer(this);
            uinput = new userInput();

            cSocket = null;
```

```
            out = null;
            in = null;

            boolean error;
            error = false;

            server = JOptionPane.showInputDialog("Server: ", "localhost");
            try
            {

                  cSocket = new Socket(server,9999);
                  out = new PrintWriter(cSocket.getOutputStream(), true);
                  in = new BufferedReader(new
InputStreamReader(cSocket.getInputStream()));
                  uinput.start();
                  sinput.start();
            }
            catch(UnknownHostException e)
            {

                  JOptionPane.showMessageDialog( this, "No server is
found","ERROR", JOptionPane.ERROR_MESSAGE );
                  System.out.println("Host Error" + e);
                  setTitle("Simple Java Chat - Cannot Connect Please Try
another server");
                  error = true;
            }
            catch (IOException e)
            {
                  System.out.println("IOException" + e);
            }
            if (!error)
            {

                  nick = null;
                  nick = JOptionPane.showInputDialog(null, "NickName: ");

                  while(nick.contains(";"))
                  {
                        nick = JOptionPane.showInputDialog(null, "Input
Nickname \";\".Cannot be null .");
                  }

                  send("Login: "+nick);

                  if (nick != null)
                  {
                        jMenuItem1.setEnabled(false);
                        jMenuItem2.setEnabled(true);
```

```
                }
        }
}


private void sendButtonActionPerformed(java.awt.event.ActionEvent evt)
{
        sendInput();
}


  static void send(String msg)
  {
        out.println(msg);
  }

  static String read()
  {

        String s = null;
        try
        {

                s = in.readLine();
        }
        catch (Exception e)
        {
                System.out.println(e);
        }

        return s;
  }


  static String replace(String str, String pattern, String replace)
  {
        int s = 0;
        int e = 0;
        StringBuffer result = new StringBuffer();
  while ((e = str.indexOf(pattern, s)) >= 0)
  {
        result.append(str.substring(s, e));
            result.append(replace);
            s = e+pattern.length();
  }
  result.append(str.substring(s));
  return result.toString();
}
```

```
    void sendInput()
    {

      if (!connected)
      {
            JOptionPane.showMessageDialog( this, "Not connected! Actions -
Connect","Error", JOptionPane.ERROR_MESSAGE );
            inputText.setText("");
      }

      else if(inputText.getText().equals("") ||
inputText.getText().equals("\n") ||  inputText.getText()== null  )
      {
            inputText.setText("");
      }
      else
      {

            send("Post " + replace(inputText.getText(),"\n"," "));
            inputText.setText("");
      }
    }

    public static String nick;
    private javax.swing.JTextArea inputText;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    static public javax.swing.JMenuItem jMenuItem1;
    static public javax.swing.JMenuItem jMenuItem2;
    static public javax.swing.JMenuItem jMenuItem3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    public static javax.swing.JTextArea mainText;
    private javax.swing.JList nickList;
    private javax.swing.JButton sendButton;

}
```

PasswordDemo.java

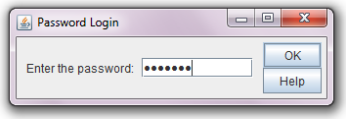
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Arrays;

public class PasswordDemo extends JPanel
                          implements ActionListener {
```

```
private static String OK = "ok";
private static String HELP = "help";

private JFrame controllingFrame;
private JPasswordField passwordField;

public PasswordDemo(JFrame f) {
    controllingFrame = f;

    passwordField = new JPasswordField(10);
    passwordField.setActionCommand(OK);
    passwordField.addActionListener(this);

    JLabel label = new JLabel("Enter the password: ");
    label.setLabelFor(passwordField);

    JComponent buttonPane = createButtonPanel();

    JPanel textPane = new JPanel(new FlowLayout(FlowLayout.TRAILING));
    textPane.add(label);
    textPane.add(passwordField);

    add(textPane);
    add(buttonPane);
}
protected JComponent createButtonPanel() {
    JPanel p = new JPanel(new GridLayout(0,1));
    JButton okButton = new JButton("OK");
    JButton helpButton = new JButton("Help");

    okButton.setActionCommand(OK);
    helpButton.setActionCommand(HELP);
    okButton.addActionListener(this);
    helpButton.addActionListener(this);

    p.add(okButton);
    p.add(helpButton);

    return p;
}

public void actionPerformed(ActionEvent e) {
    String cmd = e.getActionCommand();

    if (OK.equals(cmd)) { //Process the password.
        char[] input = passwordField.getPassword();
        if (isPasswordCorrect(input)) {
            JOptionPane.showMessageDialog(controllingFrame,
                "Success! You typed the right password.");
```

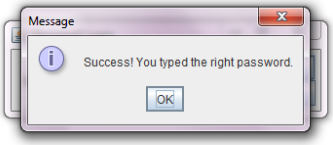
```
                client c = new client();
                c.run();


            } else {
                JOptionPane.showMessageDialog(controllingFrame,
                    "Invalid password. Try again.",
                    "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }

            //Zero out the possible password, for security.
            Arrays.fill(input, '0');

            passwordField.selectAll();
            resetFocus();
        } else { //The user has asked for help.
            JOptionPane.showMessageDialog(controllingFrame,
                "You can get the password by requesting to the
administrator");
        }
    }

//declaring password
    private static boolean isPasswordCorrect(char[] input) {
        boolean isCorrect = true;
        char[] correctPassword = { 't', 'e', 's', 't', 'i', 'n', 'g' };

        if (input.length != correctPassword.length) {
            isCorrect = false;
        } else {
            isCorrect = Arrays.equals (input, correctPassword);
        }

        //Zero out the password.
        Arrays.fill(correctPassword,'0');

        return isCorrect;
    }

    //Must be called from the event dispatch thread.
    protected void resetFocus() {
        passwordField.requestFocusInWindow();
    }

        private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("Password Login");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        //Create and set up the content pane.
        final PasswordDemo newContentPane = new PasswordDemo(frame);
        newContentPane.setOpaque(true); //content panes must be opaque
        frame.setContentPane(newContentPane);

        frame.addWindowListener(new WindowAdapter() {
            public void windowActivated(WindowEvent e) {
                newContentPane.resetFocus();
            }
        });

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event dispatch thread:
        //creating and showing this application's GUI.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                //Turn off metal's use of bold fonts
            UIManager.put("swing.boldMetal", Boolean.FALSE);
            createAndShowGUI();
            }
        });
    }
}
```

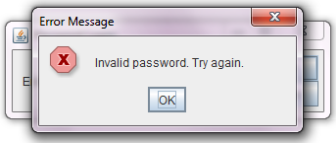
SCREENSHOT



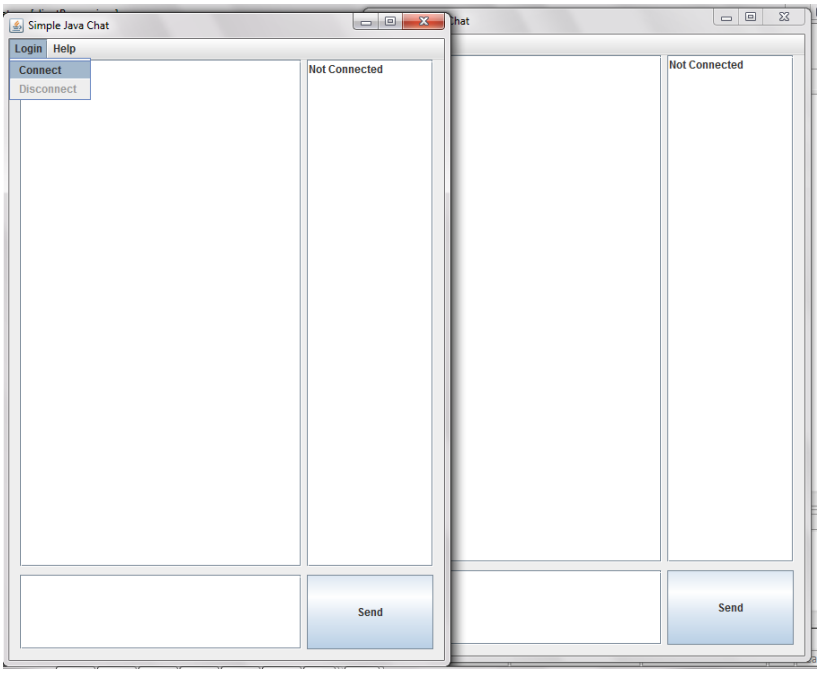
Request Password



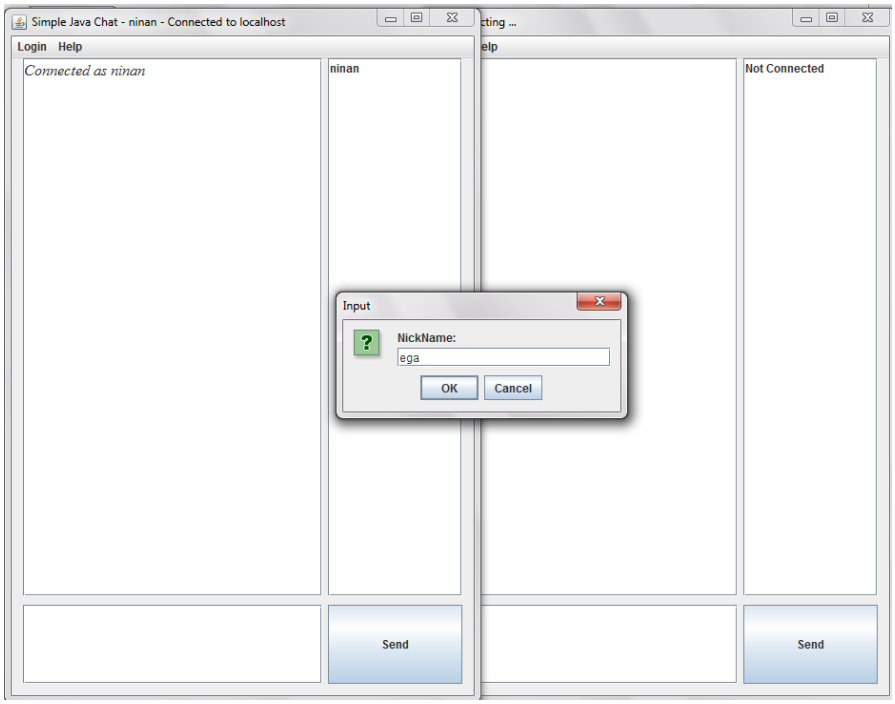
Sukses memasukkan password



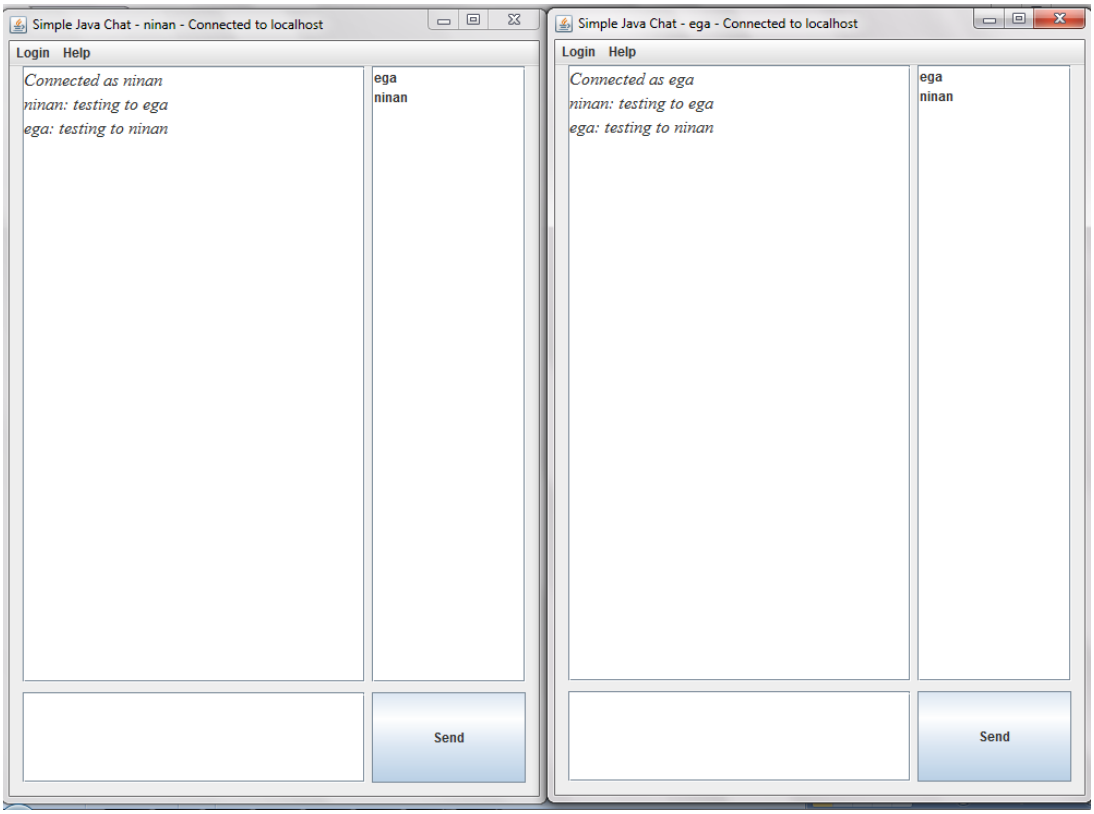
Pesan bila password yang dimasukkan salah



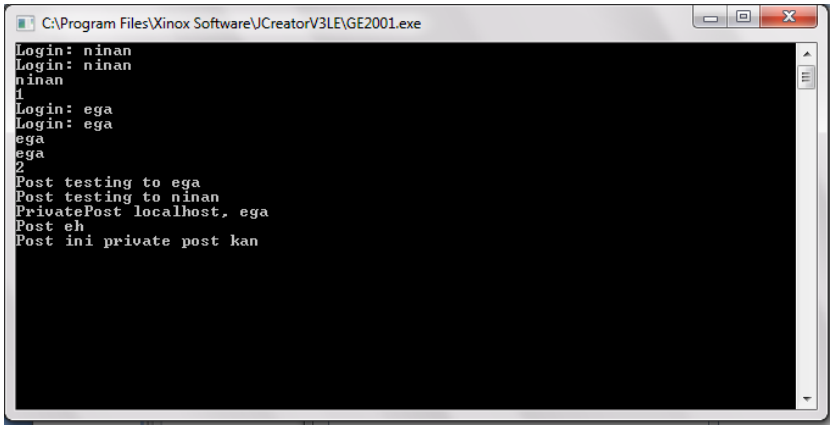
Client login



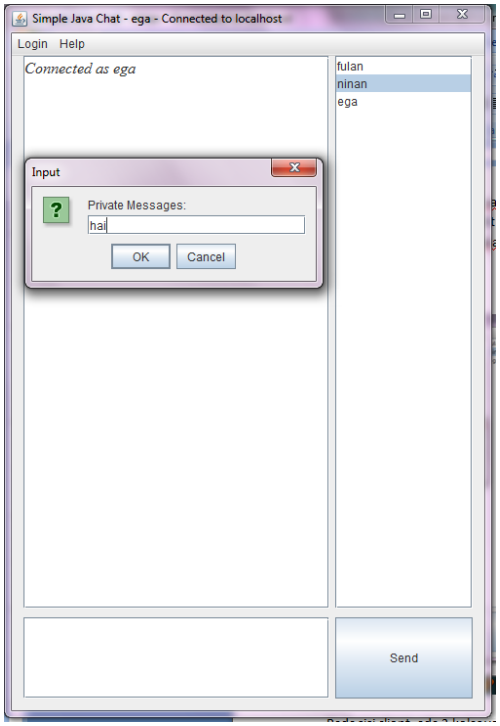
Input nickname



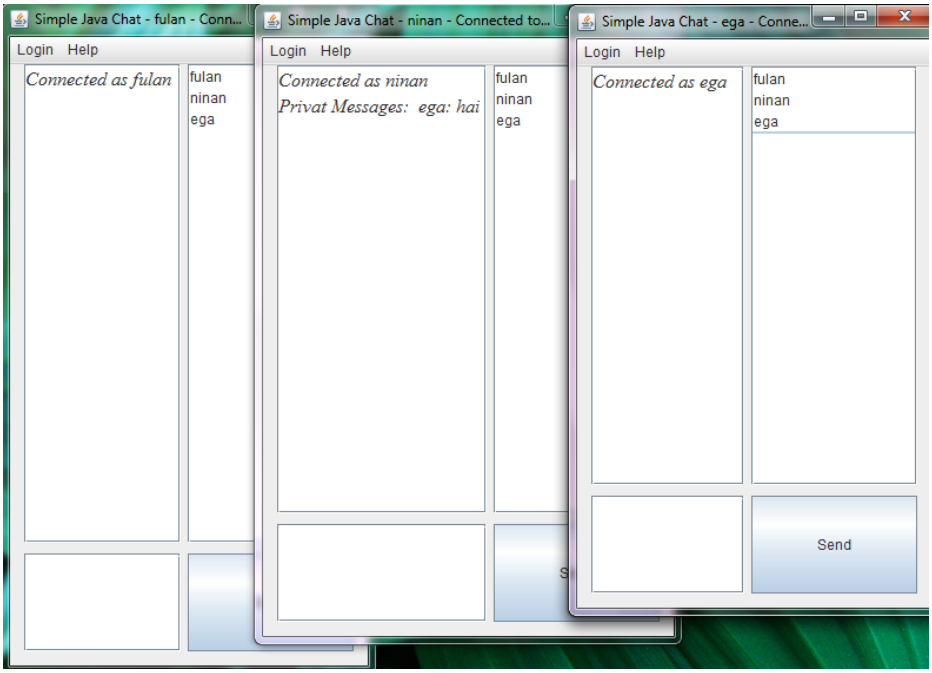
Test chatting



server



Private Message oleh Ega ke Ninan



Private Message yang diterima Ninan tidak diterima Fulan