Chat Server Programming Assignment

3/26/2018

Vinod Thorat

Chat Application

Problem Statement

Chat Server Programming Assignment
==========
Introduction

You have to write a program based on provided requirements. You are free to use a programming language of your choice, preferably Java. Please state all assumptions when implementing a solution. It is important to deliver elegant object-oriented code, apply design patterns and consider performance of the implementation. In addition, please provide JUnit test cases and comments for the implemented classes. Your delivered program should compile (provide all required jars) and run.

Chat Server

Develop a TCP/IP based multi-party chat server and client. Application should include appropriate unit test, error logging, and design documentation.

Server Requirements:
- Server should accept 1 to 10 simultaneous client connections
- A message received from any client should be echoed to all clients by default
- The server should log all client connect and disconnect events
- The listening IP and port address should be configurable

Client Requirements:
- The server IP and port address should be configurable
- The client should attempt to connect to the server and gracefully handle connection failures and disconnect events
- At start-up, the client should ask the user to provide a unique username
- The client should allow the user to enter an alpha-numeric text message
- Entered text messages should be transmitted to the server
- The client should display received text messages
- The client should be able to filter messages by user
- (Optional) The client should be able to send a message to a selected user

Chat Application

Solution

The solution for given assignment is in two phases.

1. Screenshots of all over application.
2. Source code for given application.
3. Junit testing.
4. Logging file.
5. Two applications data.

Source code

Java Code:

ServerCode.java

```
package chat_server;

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

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

/**
 *
 * @author vinod thorat :
 *
 * A TCP server that runs on port 2222.  When a client connects, it
 * sends the client the current date and time, then closes the
 * connection with that client.  Arguably just about the simplest
 * server you can write.
 */
public class ServerCode extends javax.swing.JFrame
{

   /**
       * serialVersionUID for Server
       */
      private static final long serialVersionUID = 1L;
```

```
      final static Logger logger = Logger.getLogger(ServerCode.class);

      int port = 2222;

      Boolean connection_Done = false;


      ArrayList userDataOutputStreams;
      ArrayList<String> users_Info;

      HashMap<String, String> hashmap = new HashMap<>();


public class CommunicationPhase implements Runnable
   {
        BufferedReader reader;
        Socket socket;
        PrintWriter clientInformation;

        public CommunicationPhase(Socket clientSocket, PrintWriter
user)
        {
             clientInformation = user;
             try
             {
                 socket = clientSocket;
                 InputStreamReader isReader = new
InputStreamReader(socket.getInputStream());
                 reader = new BufferedReader(isReader);
             }
             catch (Exception ex)
             {
                 ta_chat.append("Unexpected error... \n");
                 logger.error("ERROR : Error Information while getting
information about user using getInputStream");
             }

        }

        @Override
        public void run()
        {
             String information;
             String connect = "Connect", disconnect = "Disconnect";
```

```
            String chat = "Chat" ;
            String[] dataStream;

            try
            {
                while ((information = reader.readLine()) != null)
                {
                    ta_chat.append("Getting The data : " + information
+ "\n");
                    dataStream = information.split(":");
                    logger.error("This is error for  : " +
dataStream[0]);
                    for (String single_Data:dataStream)
                    {
                        ta_chat.append(single_Data + "\n");
                    }

                    if (dataStream[2].equals(connect))
                    {
                        broadcastMessage((dataStream[0] + ":" +
dataStream[1] + ":" + chat));
                        addUserData(dataStream[0]);
                        logger.info("INFO : On Connect: Num clients
after connect: " + userDataOutputStreams.size());

                    }
                    else if (dataStream[2].equals(disconnect))
                    {
                      logger.info("INFO : dataStream   On Disconnect:
Num clients after disconnect: " + dataStream[2]);

                        broadcastMessage((dataStream[0] + ":has
disconnected." + ":" + chat));
                        removeUserData(dataStream[0]);

userDataOutputStreams.remove(clientInformation);
                        logger.info("INFO : On Disconnect: Num clients
after disconnect: " + userDataOutputStreams.size());
                    }
                    else if (dataStream[2].equals(chat))
                    {
                        broadcastMessage(information);
                        logger.info("INFO : On Chat Information: Num
clients after disconnect: " + userDataOutputStreams.size());
```

```
                    }
                    else
                    {
                        ta_chat.append("No Conditions were met for
this, Please check it once. \n");
                    }
                }
            }
            catch (Exception ex)
            {
                ta_chat.append("Connection Lost. \n");
                ex.printStackTrace();
                logger.error("ERROR : This is error for Lost
Connection  : " + ex.getMessage());

                userDataOutputStreams.remove(clientInformation);
                logger.info("INFO : Remove operation on given data
userDataOutputStreams");

            }
      }
    }

    public ServerCode()
    {
        initializingUIcomponentsServer();
    }

    private void initializingUIcomponentsServer() {

        jScrollPane1 = new javax.swing.JScrollPane();
        ta_chat = new javax.swing.JTextArea();
        b_start = new javax.swing.JButton();
        b_end = new javax.swing.JButton();
        b_users = new javax.swing.JButton();
        b_clear = new javax.swing.JButton();
        lb_name = new javax.swing.JLabel();
        lb_port = new javax.swing.JLabel();
        tf_port = new javax.swing.JTextField();

        lb_port.setText("(Port)");

        tf_port.setText("2222");
```

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



setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Chat Application - Server  Window");
        setName("server");
        setResizable(false);

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

        b_start.setText("START SERVER");
        b_start.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                startButtonAction(evt);
            }
        });

        b_end.setText("END SERVER");
        b_end.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                endButtonAction(evt);
            }
        });

        b_users.setText("Online Client");
        b_users.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                onlineUserButtonAction(evt);
            }
```

```
        });

        b_clear.setText("Clear Text");
        b_clear.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                clearButtonAction(evt);
            }
        });

        lb_name.setText("vthorat1");

lb_name.setBorder(javax.swing.BorderFactory.createLineBorder(new
java.awt.Color(0, 0, 0)));

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

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                    .addComponent(jScrollPane1)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                            .addComponent(b_end,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(b_start,
javax.swing.GroupLayout.DEFAULT_SIZE, 75, Short.MAX_VALUE)

                            )

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
291, Short.MAX_VALUE)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                                .addComponent(b_clear,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(b_users,
javax.swing.GroupLayout.DEFAULT_SIZE, 103, Short.MAX_VALUE)))
                        .addComponent(tf_port,
javax.swing.GroupLayout.DEFAULT_SIZE, 50, Short.MAX_VALUE))
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                                    .addComponent(lb_port,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(tf_port,
javax.swing.GroupLayout.DEFAULT_SIZE, 10, Short.MAX_VALUE))

        )
                    .addContainerGap())
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(lb_name)
                    .addGap(209, 209, 209))
            );
            layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 340, Short.MAX_VALUE)
                    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                        .addComponent(b_start)
                        .addComponent(b_users)
                        )
                    .addGap(10,10,10)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                        .addComponent(lb_port)
                          .addComponent(tf_port,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                          )
                .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                    .addComponent(b_clear)
                    .addComponent(b_end))
                .addGap(4, 4, 4)
                .addComponent(lb_name))
        );

        pack();
    }
    private void endButtonAction(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_b_endActionPerformed
        try
        {
            Thread.sleep(6000);//6000 milliseconds is Six second.

            tf_port.setEditable(true);
            logger.info("INFO : Here Thread is sleeping for five
seconds.");
        }
        catch(InterruptedException ex)
{Thread.currentThread().interrupt();}

        broadcastMessage("Server:is stopping and all users will be
disconnected.\n:Chat");
        logger.error("ERROR : Server:is stopping and all users will be
disconnected.\\n:Chat and Server stopping");

        ta_chat.append("Server stopping... \n");

        ta_chat.setText("");
    }
```

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

           tf_port.setEditable(false);
           logger.info("INFO : Here Thread Start Begin.");

           Thread starter = new Thread(new ServerStart());
        starter.start();
        if(connection_Done == false) {
        ta_chat.append("Server started Successfully and Ready for
communication.\n");
        }
    }

    private void onlineUserButtonAction(java.awt.event.ActionEvent
evt) {
        ta_chat.append("\n Online users : \n");
        logger.info("INFO : Getting online users or clients
information");

        for (String presentUser : users_Info)
        {
            ta_chat.append(presentUser);
            ta_chat.append("\n");
           }
    }

    private void portNumberAction(java.awt.event.ActionEvent evt) {
      logger.info("INFO : Action happened on Port number");

    }

    private void clearButtonAction(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_b_clearActionPerformed
        ta_chat.setText("");
        logger.info("INFO : Action happened on Clear Button");

    }


    public class ServerStart implements Runnable
    {
        @Override
        public void run()
        {
```

```
                connection_Done = true;

            userDataOutputStreams = new ArrayList();
            users_Info = new ArrayList();
            port = Integer.parseInt(tf_port.getText());

            logger.info("INFO : Information about User and port number
: " + port);

            try
            {
                ServerSocket serverSock = new ServerSocket(port);

                while (true)
                {
                      Socket clientSock = serverSock.accept();
                      PrintWriter writer = new
PrintWriter(clientSock.getOutputStream());

                      logger.info("INFO : Adding information into the
userDataOutputStreams from socket getOutputStream");
                      userDataOutputStreams.add(writer);

                      logger.info("INFO : num clients: " +
userDataOutputStreams.size());

                      Thread listener = new Thread(new
CommunicationPhase(clientSock, writer));

                      listener.start();
                      ta_chat.append("Got a connection. \n");
                       }
            }
            catch (Exception ex)
            {
                ta_chat.append("Error while making a connection.
Please check if connection exist.\n");
            }
            finally {
                logger.warn("WARN : Please do close all the
cooresponding connections");

                }
        }
```

```
    }

    /**
     * Adding User Data into the list to access online Users in
future.
     * @param data
     */
    public void addUserData (String data)
    {
        String message, add = ": :Connect", done = "Server: :Done",
name = data;
        ta_chat.append("Before adding user now  " + name + " added.
\n");
        users_Info.add(name);
        ta_chat.append("After adding user now  " + name + " added.
\n");
        String[] tempList = new String[(users_Info.size())];
        users_Info.toArray(tempList);

        for (String token:tempList)
        {
            message = (token + add);
            broadcastMessage(message);
        }
        broadcastMessage(done);
    }

    /**
     * Removing User Data from the list to access online Users in
future.
     * @param data
     */

    public void removeUserData (String data)
    {
        String message, add = ": :Disconnect", done = "Server: :Done",
name = data;
        users_Info.remove(name);
        String[] tempList = new String[(users_Info.size())];
        users_Info.toArray(tempList);

        for (String token:tempList)
        {
            message = (token + add);
```

```
            broadcastMessage(message);
        }
        broadcastMessage(done);
    }

    /**
     * Send the given message to all the users means send all the
messages to the clients (Broadcasting all the information related to
the each other.)
     * @param message
     */
    public void broadcastMessage(String message)
    {
           Iterator it = userDataOutputStreams.iterator();

        while (it.hasNext())
        {
            try
            {
                PrintWriter writer = (PrintWriter) it.next();
                writer.println(message);
                logger.info("INFO : Sending messages :" + message);
                ta_chat.append("Sending: " + message + "\n");
                writer.flush();

ta_chat.setCaretPosition(ta_chat.getDocument().getLength());

            }
            catch (Exception ex)
            {
                        ta_chat.append("Error : While Broadcasting
Message to each client. \n");
                        logger.error("Error : While Broadcasting Message
to each client.");
            }
        }
    }

    // Component declaration which is used during this Server phase.
    //This variables used for the giving best look and working
functionality of this application.
    private javax.swing.JButton b_clear;
    private javax.swing.JButton b_end;
    private javax.swing.JButton b_start;
```

```
    private javax.swing.JButton b_users;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lb_name;
    private javax.swing.JTextArea ta_chat;
    private javax.swing.JLabel lb_port;
    private javax.swing.JTextField tf_port;


    /**
     * Runs the server.
     */
    public static void main(String args[])
    {

            String log4jConfigFile = System.getProperty("user.dir")
                  + File.separator + "src" + File.separator +
"log4j.properties";
          PropertyConfigurator.configure(log4jConfigFile);
          logger.info("START : this is a information log message");
          logger.info("INFORMATION : Client Application Starting");


          java.awt.EventQueue.invokeLater(new Runnable()
         {
             @Override
             public void run() {
                 new ServerCode().setVisible(true);

             }
         });
    }


}
```

ClientCode.java

```
package chat_server;

import java.net.*;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.io.*;
import java.util.*;

import javax.swing.JComboBox;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

/**
 *
 * @author vinod thorat :
 *
 *This class is for client which will obtain the connection with
server.
 *When we run this class client user will get added to server after
successful connection with given port and IP address.
 */
public class ClientCode extends javax.swing.JFrame
{


    /**
       * serialVersionUID for client
       */
      private static final long serialVersionUID = 1L;

      final static Logger logger = Logger.getLogger(ClientCode.class);

      String username;
    String address_IP = "localhost";
    ArrayList<String> clientUserList = new ArrayList<String>();
    int port = 2222;
    Boolean connection_Done = false;
    String[] data_Values;
    Socket socket;
    BufferedReader readerInfo,readerInfo123;
    PrintWriter writerInfo;
```

```
/**
 * ListenThread method :
 *
 * This method is for listening thread.
 */
public void ThreadListening()
{
       logger.info("CLIENT INFO : Called ThreadListening Method");
     Thread ThreadReader = new Thread(new ThreadReader());

     logger.info("CLIENT INFO : Startig Thread");
     ThreadReader.start();
}

/**
 * addUser :
 * This method is for adding online users in a given list.
 *
 * @param info
 *
 *
 */
public void addUser(String info)
{
       logger.info("CLIENT INFO : Adding User information");
    clientUserList.add(info);
}

/**
 * removeUser :
 * This method is for removing users in a given list.
 *
 * @param info
 *
 *
 */
public void removeUser(String info)
{
       logger.info("CLIENT INFO : Removing User information");
     ta_chat.append(info + " this is disconnected now.\n");
}

/**
 * userInformationWrite :
```

```
     *
     * Contains writing logic for given list of user.
     */
    public void userInformationWrite()
    {
            logger.info("CLIENT INFO : userInformation Write User
information");

          String[] tempList = new String[(clientUserList.size())];
          clientUserList.toArray(tempList);
//          for (String token:tempList)
//          {
//              //users.append(token + "\n");
//          }
    }

    /**
     * sendingDisconnectInformation :
     *
     * User disconnect Information.
     */
    public void sendingDisconnectInformation()
    {
            logger.info("CLIENT INFO : Now sendingDisconnectInformation
");

        String bye = (username + ": :Disconnect");
        try
        {
            writerInfo.println(bye);
            writerInfo.flush();
            data_Values = new String[0];
        } catch (Exception e)
        {
            ta_chat.append("Could not send Disconnect message.\n");
            logger.error("CLIENT INFO : Could not send Disconnect
message. ");

        }
    }

    /**
     * disconnectClient :
     *
```

```
     * ADD information to the chat text that user disconnected
     * and call sendingDisconnectInformation()
     */
    public void disconnectClient()
    {
            logger.info("CLIENT INFO : Now disconnectClient");

        try
        {
            ta_chat.append("Disconnected.\n");
            socket.close();
            logger.info("CLIENT INFO : Connection with socket Closed
Successfully.");

        } catch(Exception ex) {
            ta_chat.append("Failed to disconnect. \n");
            logger.error("CLIENT INFO : Failed to disconnect, Please
check given information. ");

        }
        connection_Done = false;
        tf_username.setEditable(true);

    }


    public ClientCode()
    {
            logger.info("CLIENT INFO : Initializing All user defined
components.");
        initializingAllUIVariables();
    }


    public class ThreadReader implements Runnable
    {
        @Override
        public void run()
        {
                logger.info("CLIENT INFO : calling run() method from
ThreadReader.");

            String[] data;
            String streamReaderInfo;
```

```
            String done = "Done";
                String connect = "Connect", disconnect = "Disconnect",
chat = "Chat";

            try
            {
                     // Get messages from the client, line by line;
return them
                while ((streamReaderInfo = readerInfo.readLine()) !=
null)
                {
                     data = streamReaderInfo.split(":");

                     System.out.println("DATATATATATTAA 2222222222 : "
+ data[2]);
                     if (data[2].equals(chat))
                     {
                           logger.info("CLIENT INFO : stream data
matched with chat option");

                        ta_chat.append(data[0] + ": " + data[1] +
"\n");

ta_chat.setCaretPosition(ta_chat.getDocument().getLength());
                     }
                     else if (data[2].equals(connect))
                     {
                           logger.info("CLIENT INFO : stream data
matched with connect option");
                        ta_chat.removeAll();
                        addUser(data[0]); // add each extracted detail
from the text file that was stored in the list of the stuff object

                        if(cbox1.getSelectedIndex() == -1) {
                                    cbox1.addItem("All");

                        }
                          cbox1.addItem(data[0]);

                          Object selected = cbox1.getSelectedItem();
                            System.out.println("Selected Item  = " +
selected);
                            if("All".equals(selected)) {
```

```
                                        //System.out.println("Action
Command =  ALLLLLLLLLLLLLLLL");
                                }

                        }
                        else if (data[2].equals(disconnect))
                        {
                                logger.info("CLIENT INFO : stream data
matched with disconnect option and calling remove user method");

                            removeUser(data[0]);
                            cbox1.removeItem(data[0]);
                        }
                        else if (data[2].equals(done))
                        {
                                logger.info("CLIENT INFO : Done with this
information so clearing data from user list before it calling
userInformationWrite method.");
                                cbox1.removeItem(data[0]);

                           userInformationWrite();
                           clientUserList.clear();
                        }
                    }
                }catch(Exception ex) { }
            }
        }

        @SuppressWarnings("unchecked")
        private void initializingAllUIVariables() {

                logger.info("CLIENT INFO : Initializing All the UI
components.");

            lb_address = new javax.swing.JLabel();
            tf_address = new javax.swing.JTextField();
            lb_port = new javax.swing.JLabel();
            tf_port = new javax.swing.JTextField();
            lb_username = new javax.swing.JLabel();
            tf_username = new javax.swing.JTextField();
            b_connect = new javax.swing.JButton();
            b_disconnect = new javax.swing.JButton();
            b_anonymous = new javax.swing.JButton();
            jScrollPane1 = new javax.swing.JScrollPane();
```

```
        ta_chat = new javax.swing.JTextArea();
        tf_chat = new javax.swing.JTextField();
        b_send = new javax.swing.JButton();
        lb_name = new javax.swing.JLabel();

        cbox = new javax.swing.JLabel();
        cbox.setText("Sort by client Name: ");
        cbox1  = new javax.swing.JComboBox();
        cbox1.setSelectedIndex(-1);


        cbox1 = new JComboBox(clientUserList.toArray());  // a
constructor which accepts an array of string items unlike AWT Choice
component


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Chat Application - Client Window");
        setName("client");
        setResizable(true);

        lb_address.setText("IP Address : ");

        tf_address.setText("localhost");
        tf_address.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                ipAddressAction(evt);
            }
        });

        lb_port.setText("Port :");

        tf_port.setText("2222");
        tf_port.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                portNumberAction(evt);
            }
        });

        cbox1.addActionListener(new java.awt.event.ActionListener() {
```

```
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                comboBoxUserDataAction(evt);
            }
        });


        lb_username.setText("User Name :");

        tf_username.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                userDataNameAction(evt);
            }
        });


        b_connect.setText("Connect");
        b_connect.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                connectAction(evt);
            }
        });

        b_disconnect.setText("Detach");
        b_disconnect.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                disconnectAction(evt);
            }
        });

        b_anonymous.setText("Guest Login");
        b_anonymous.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                guestAction(evt);
            }
        });
```

```
        ta_chat.setColumns(20);
        ta_chat.setRows(5);
        jScrollPane1.setViewportView(ta_chat);

        b_send.setText("SEND MESSAGE");
        b_send.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                sendMessageAction(evt);
            }
        });

        lb_name.setText("vthorat1");

lb_name.setBorder(javax.swing.BorderFactory.createLineBorder(new
java.awt.Color(0, 255, 0)));

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        getContentPane().setPreferredSize(new Dimension(700, 550));
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(tf_chat)

//.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(b_send))
                    .addComponent(jScrollPane1)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.TRAILING, false)
                            .addComponent(lb_username,
javax.swing.GroupLayout.DEFAULT_SIZE, 62, Short.MAX_VALUE)
```

```
                                .addComponent(lb_address,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                            .addGap(10, 10, 10)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                                .addComponent(tf_address,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(tf_username))
                            .addGap(10, 10, 10)


.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                               //.addComponent(lb_password,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

     //.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.TRAILING, false)
                                            .addComponent(cbox,
javax.swing.GroupLayout.DEFAULT_SIZE, 77, Short.MAX_VALUE)
                                            //.addComponent(cbox1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))


                                   //.addGap(18, 18, 18)

                                .addComponent(lb_port,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                                    .addComponent(cbox1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                                .addComponent(tf_port,
javax.swing.GroupLayout.DEFAULT_SIZE, 50, Short.MAX_VALUE))
```

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                                .addGroup(layout.createSequentialGroup()
                                    .addComponent(b_connect)
                                    .addGap(2, 2, 2)
                                    .addComponent(b_disconnect)
                                    .addGap(0, 0, Short.MAX_VALUE))
                                .addComponent(b_anonymous,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
                    .addContainerGap())
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(lb_name)
                    .addGap(201, 201, 201))
            );
            layout.setVerticalGroup(

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                        .addComponent(lb_address)
                        .addComponent(tf_address,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        //.addComponent(cbox)
                        //.addComponent(cbox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

                        .addComponent(lb_port)
                        .addComponent(tf_port,
javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(b_anonymous))

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                    .addComponent(tf_username)
                    .addComponent(cbox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)


.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                        .addComponent(lb_username)
                        .addComponent(cbox)
                        .addComponent(b_connect)
                        .addComponent(b_disconnect)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 310,
javax.swing.GroupLayout.PREFERRED_SIZE)

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                    .addComponent(tf_chat)
                    .addComponent(b_send))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(lb_name))
        );

        pack();
    }

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

    }
```

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

    }
    private void comboBoxUserDataAction(ActionEvent evt) {
           JComboBox<?> comboBox = (JComboBox<?>) evt.getSource();
           logger.info("CLIENT INFO : calling combo box action for
filtering data as per user.");

           Object selected = comboBox.getSelectedItem();
            System.out.println("Selected Item  = " + selected);
         //   ThreadListening();

          //  if("All".equals(selected)) {
           //    System.out.println("Action Command =
ALLLLLLLLLLLLLLLL");

           //    gellALlUsers();

           // }
          //  else{

             /*   String[] data;
                String streamReaderInfo;
                String done = "Done";
                String connect = "Connect", disconnect = "Disconnect",
chat = "Chat";

                try
                {

                System.out.println("Action Command =
USERRRRRRRRRRRRRRRRRRRRRRRRRRRR");

                      socket = new Socket("localhost", 2222);
                    InputStreamReader streamreader = new
InputStreamReader(socket.getInputStream());
                    //readerInfo = new BufferedReader(streamreader);
                   // InputStreamReader streamreader = new
InputStreamReader(socket.getInputStream());
                    readerInfo123 = new BufferedReader(streamreader);
                    writerInfo = new
PrintWriter(socket.getOutputStream());
                    writerInfo.println(username + ":has
connected.:Connect");
```

```
                    writerInfo.flush();
                    connection_Done = true;
                }
                catch (Exception ex)
                {
                    ta_chat.append("Cannot Connect! Try Again, check
port Number. \n");
                    tf_username.setEditable(true);
                    logger.error("CLIENT ERROR : Having Issue with
connection, Cannot Connect! Try Again.");
                }

                ThreadListening123(selected);

*/
       //     }


        }


    private void ThreadListening123(Object selected) {
           // TODO Auto-generated method stub

      String[] data;
         String streamReaderInfo;
         String done = "Done";
            String connect = "Connect", disconnect = "Disconnect", chat
= "Chat";

            // Get messages from the client, line by line; return them
         try {
                while ((streamReaderInfo = readerInfo123.readLine())
!= null)
                {
                     System.out.println("Action Command =
USERRRRRRRRRRRRRRRRRRRRRRRRRRRRR");

                    data = streamReaderInfo.split(":");


                    if (data[0].contains(selected.toString())) {
```

```
                                System.out.println("Action Command =
USERRRRRRRRRRRRRRRRRRRRRRRRRRRR");

                            ta_chat.removeAll();
                            ta_chat.append(data[0] + ": " + data[1] + "\n");

ta_chat.setCaretPosition(ta_chat.getDocument().getLength());

                        }
                    }
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
    }



      private void gellALlUsers() {

      }

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

     }

     private void connectAction(java.awt.event.ActionEvent evt) {
       logger.info("CLIENT INFO : checking connection.");

            if (connection_Done == false)
         {
                logger.info("CLIENT INFO : No previous connection
exists.");

             username = tf_username.getText();
             tf_username.setEditable(false);
             port = Integer.parseInt(tf_port.getText());
             address_IP = tf_address.getText();
             try
             {

                 if(username.isEmpty()) {
                     ta_chat.append("Please Connect with User Name for
Future References. \n");
```

```
            }


                logger.info("CLIENT INFO : Creating connection with
given information of IP and Port." + address_IP + "and port is" +
port);

                socket = new Socket(address_IP, port);
                InputStreamReader streamreader = new
InputStreamReader(socket.getInputStream());
                readerInfo = new BufferedReader(streamreader);
                writerInfo = new
PrintWriter(socket.getOutputStream());
                writerInfo.println(username + ":has
connected.:Connect");
                writerInfo.flush();
                connection_Done = true;
            }
            catch (Exception ex)
            {
                ta_chat.append("Cannot Connect! Try Again, check port
Number. \n");
                tf_username.setEditable(true);
                logger.error("CLIENT ERROR : Having Issue with
connection, Cannot Connect! Try Again.");
            }

            ThreadListening();

        } else if (connection_Done == true)
        {
            ta_chat.append("You are already connected with given port.
\n");
            logger.info("CLIENT INFO : You are already connected with
given port.");
        }
    }

    private void disconnectAction(java.awt.event.ActionEvent evt) {
           logger.info("calling disconnect event for disconnecting
client.");

        sendingDisconnectInformation();
```

```
        disconnectClient();
    }

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

           logger.info("CLIENT INFO : Creating Guest even for adding
guest using this action.");

        tf_username.setText("");
        if (connection_Done == false)
        {
                logger.info("CLIENT INFO : Creating Guest even for
adding guest using this action.");

            String anon="Guest ";
            Random generator = new Random();
            int i = generator.nextInt(999) + 1;
            String is=String.valueOf(i);
            anon=anon.concat(is);
            username=anon;

            tf_username.setText(anon);
            tf_username.setEditable(false);
            port = Integer.parseInt(tf_port.getText());
            address_IP = tf_address.getText();

            try
            {
                socket = new Socket(address_IP, port);
                InputStreamReader streamreader = new
InputStreamReader(socket.getInputStream());
                readerInfo = new BufferedReader(streamreader);
                writerInfo = new
PrintWriter(socket.getOutputStream());
                writerInfo.println(anon + ":has connected.:Connect");
                writerInfo.flush();
                connection_Done = true;
                logger.info("CLIENT INFO : Creating Guest connection
Successfully Done.");

            }
            catch (Exception ex)
            {
```

```
                ta_chat.append("Having Issue, Cannot Connect! Try
Again. \n");
                tf_username.setEditable(true);
                logger.error("CLIENT ERROR : Having Issue, Cannot
Connect! Try Again.");

            }

            ThreadListening();

        } else if (connection_Done == true)
        {
            ta_chat.append("Connection : You are already connected.
\n");
            logger.warn("CLIENT WARN : Connection : You are already
connected.");

        }
    }
    private void sendMessageAction(java.awt.event.ActionEvent evt) {

           logger.info("CLIENT INFO : Sending Message.");

           String nothing = "";
        if ((tf_chat.getText()).equals(nothing)) {
            tf_chat.setText("");
            tf_chat.requestFocus();
        } else {
            try {
               writerInfo.println(username + ":" + tf_chat.getText() +
":" + "Chat");
               writerInfo.flush(); // flushes the buffer
            } catch (Exception ex) {
                ta_chat.append("Message was not sent, due to some
issue, Please check logs. \n");
                logger.error("CLIENT ERROR : Message was not sent, due
to some issue, Please check logs.");

            }
            tf_chat.setText("");
            tf_chat.requestFocus();
        }

        tf_chat.setText("");
```

```
        tf_chat.requestFocus();
    }


    // Component declaration which is used during this Client phase.
    //This variables used for the giving best look and working
functionality of this application.
    private javax.swing.JButton b_anonymous;
    private javax.swing.JButton b_connect;
    private javax.swing.JButton b_disconnect;
    private javax.swing.JButton b_send;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lb_address;
    private javax.swing.JLabel lb_name;
    private javax.swing.JLabel lb_port;
    private javax.swing.JLabel lb_username;
    private javax.swing.JTextArea ta_chat;
    private javax.swing.JTextField tf_address;
    private javax.swing.JTextField tf_chat;
    private javax.swing.JTextField tf_port;
    private javax.swing.JTextField tf_username;
    private javax.swing.JComboBox<String> cbox1;
    private javax.swing.JLabel cbox;

    /**
     * Runs the client as an application.  First it displays a dialog
     * box asking for the IP address or host name of a host running
     * the date server, then connects to it and displays the date that
     * it serves.
     */
    public static void main(String args[])
    {

            String log4jConfigFile = System.getProperty("user.dir")
                    + File.separator + "src" + File.separator +
"log4j.properties";
            PropertyConfigurator.configure(log4jConfigFile);
            logger.info("CLIENT : START : this is a information log
message");
            logger.info("CLIENT INFORMATION : Client Application
Starting");

            java.awt.EventQueue.invokeLater(new Runnable()
            {
```

```
            @Override
            public void run()
            {
                new ClientCode().setVisible(true);

            }
        });
    }


}
```

This is Java code for client and server. Other files (regarding Junit testing and form data files) uploaded at GitHub account @vinodthorat121
Under ChatServerApplication.

Thank you,
Vinod Thorat