CS/CE 6390 – Advanced Computer Networks

Spring 2017

Project S3: Chat Program

Angaha Asok: 2021261798
axa151631

Madhumitha Shankar: 2021312870
mxs162530

04/26/2017

---

1. Introduction

The project discusses the development of a simple chat application in java using socket programming. A socket is one of the fundamental technology of computer network programming. Sockets allow network software applications to communicate using standard mechanisms built into network hardware and operating systems. Many popular network software applications rely on sockets. The application creates a socket and the socket type dictates the style of communication. Once configured the application can transmit data to the socket for network transmission receive data from the socket. Hence two machines can connect to each other. Inspired by the concepts in socket programming, we decided to do this project as it can provide a good learning experience in socket programming.

In Java the package java.net.Socket supports socket programing. The project demanded extensive learning in socket programming and advanced Java programming. It was decided to develop a GUI for the chat application and Java Swing was made use to do it. The project is developed in NetBeans and a ".jar" application file has been exported which can be ran in any PC with Java and launched.

The project aims at development of a chat application where server owns a "chat-room" and allows clients to connect to it and enter the room to chat with other users and share file.

2. Description

The project is aimed at the development of a chat application which can be ran in multiple machines and all can transmit and receive data and text files i.e. A Multi-Client Chat application. The envisioned design includes development of two applications, where one can be ran as a server and the other can be ran as a client in different machines. All the client applications should have the IP address and the port number used by the server. The server is the owner of a “chat-room” where clients and come and communicate with each other. All the communication happening in the chat room are directed through the server and is monitored by the same. The server allows any client who enters a name enter the chat room. The server does not allow two clients to connect with the same user name. When the second client connects with the same user name used by another client, the server broadcasts that the user name has been already used and prompts the client to disconnect and reconnect with a new name.

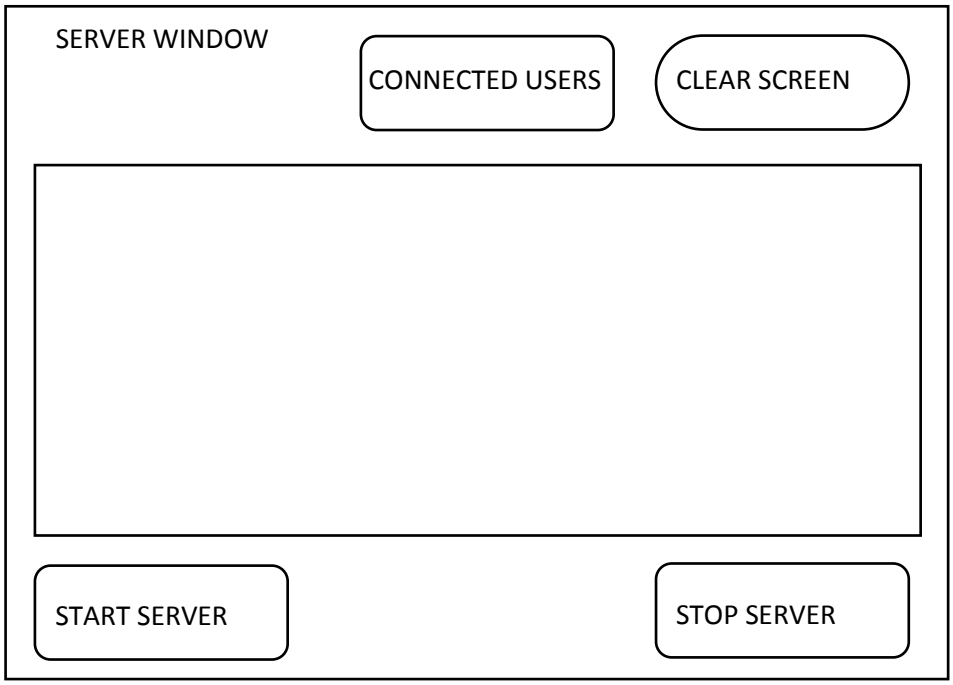
Any clients in the room can send a text file to all other clients. The sent file would be displayed on the chat screen of all the clients. Here server maintains a list of the connected users and every time a new client connects, it broadcasts the information that a new connection has been received from a particular client. It passes the information about all connected users whenever a new connection occurs that all users know who are all the users connected. If the client wants to connect as an anonymous user, the application would have a provision for that. It allows the client to connect as a “Anon_XX” user. Also, both the server and client can clear their screen and set the cursor to beginning at any given time on pressing the “clear screen” button. The client can disconnect and leave the chat room at any given time on pressing the disconnect button. The client who disconnects would get a message “Disconnected” and it would not be allowed to chat. The thread will be closed. The information that one client has disconnected and is offline would be broadcasted to all other connected users with the user name of the client who disconnected. The server has one more provision to display the list of all connected users, the given time. A button facilitates this. The server can stop managing the chat-room on pressing the “Stop Server” button and a message that the server is stopping and all users who are connected would be disconnected would be broadcasted and the server would be closed. The clients will be disconnected and all threads would be closed.

This project finds its application in a closed environment like an office or an institution where all employees can connect and chat together and share text files. The client can be installed in all machines and the server can be run by a centralized machine which can monitor all clients connects and the messages they exchange.

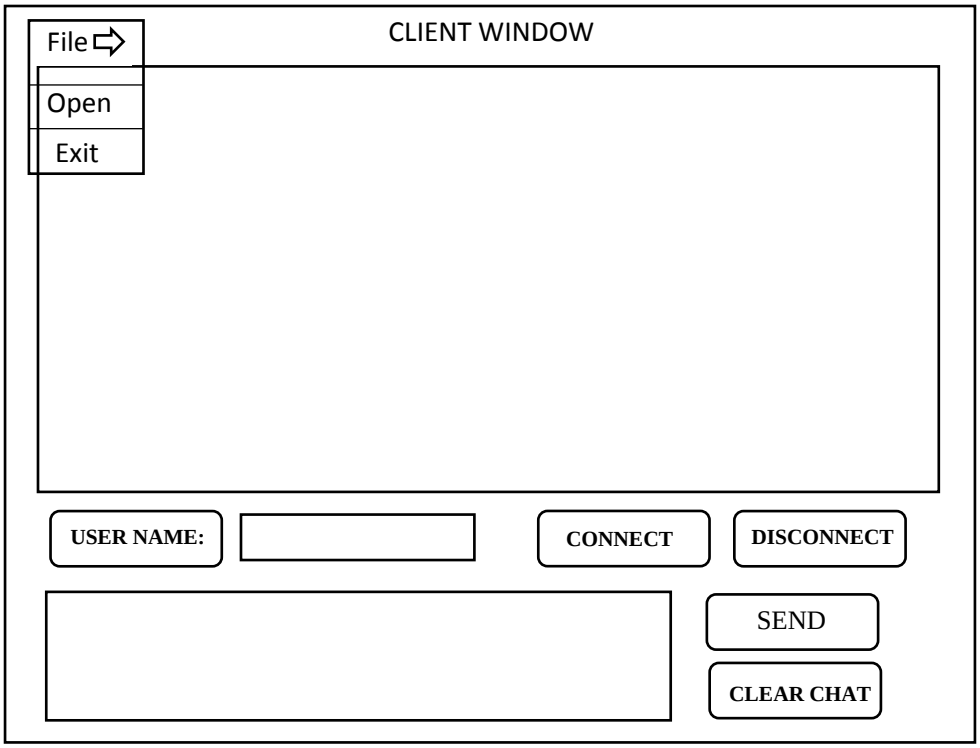
The application interface is designed as shown in the figure below:

Fig :1

Envisioned GUI: Server



Envisioned GUI: Client



3. Approach

Having the foundation laid, it was a major challenge to start with the project. A clear-cut idea about the application to be developed was as above. The problem was solved in different steps and attending individual concerns progressively. Following is our approach:

1. Socket Programming in Java was the first concern. To understand the concepts of socket programming a lot of articles were referred and sample codes were executed. At first a basic program where a client and a server exchanges simple Boolean data was written. Both applications were run in the same machine. Server address was declared as "localhost" and client and server communication was established.

2. While developing the above step, multi-threading in Java and concurrent access to global data structures by multiple clients were the major concern. Errors and logical flaws were handled step by step by reading about and referring to examples related to multi-threading in Java. Classes in java like Buffered Reader, Print Writer, FileOutputStream, Thread, InputStreamReader and FileReader were utilized in different steps.

3. Development of logic to make the server and client understand the information exchanged and channel it appropriately was another major challenge. The logic developed was to send data in a format "x:y:z" always between client and server where
   x is the user involved.
   y is the action it performed. (Left blank at times)
   z is the intended action.
   The message is always split into x, y and z up on reception and the receiver (server/client) decides what to do depending on the data received. For example if a server receives "Ana:has connected:connect", it would split it as "Ana", "has connected" and "connect". It identifies this stream as "User Ana has to connect" and hence it broadcasts the message "Ana: has connected. Available:chat" and adds "Ana" to the existing user list. All connected clients receive it and recognizes the new user and adds her to their list. All the information exchange between client and server were handled this way, where clients could connect, chat and disconnect. To transmit the information to all connected users, an iterator had to be used by the server. Implementing this part was slightly challenging.

4. Next step of the project was to develop the GUI. Java swings were used and a GUI was designed. It was developed with the help of various classes like jFrame, jButton etc. The switch cases in the code developed so far were converted to action performed on

various button clicks. This part of the project was really challenging as all the classes of swing had to be understood to design the Application Interface designed earlier.

5. A major problem incurred was the usage of class “File chooser”. Having developed the GUI for chatting, the next step was to allow user to send a file. In the application, the files which can be sent are limited to “.txt” files whose maximum size is 4 MB. A file chooser filter was used for this purpose. Reception of the file and its broadcast to other users demanded extensive research in usage of java classes related to thread communication. Even after reception of the file at the server broadcasting it to all connected users was problem. Knowledge gained from file handling and multithreading concepts in Java were put in use to debug the code.

6. Once a working chat application with file exchange was developed, there were minor challenges to facilitate options like ‘Clear Screen’, ‘Connected Users’ etc. Disconnecting the client and stopping the server were major challenges occurred here. Problems were solved by extensive debugging and hence finding out the logical flaws. Closing stream readers without closing sockets was a major issue. This was solved by defining readers outside the methods and passing them to each method when required.

4. Design and Flowchart

1. Client

The client has following functionalities:

1. Connect: Connect to the Server. This button accepts the name entered by user and send that to the server via socket defined in the code. It is broadcasted in a way that server understands a user is trying to connect. If there is no server connection available, it must display that it cannot connect. If it is pressed again after connection has established it must print that the user has already been connected.

2. Disconnect: Pressing this button should disconnect the user from server and close the thread and the socket. It must also display before closing that it has disconnected and send the information that the user has disconnected to server in a way server understands.

3. Anonymous Login: This button must facilitate the same functionality as the "connect" button. But it must provide an unknown user name.

4. Send: This button, on clicking must send the text entered on the text screen to server if the client is connected to the server. If there is no text entered it should do nothing.

5. Clear Screen: This button should clear the data in the chat screen and bring the cursor to the starting.

6. Select Text File to send -> open: This file chooser should let the user browse through the files in his PC and select a text file. On selecting it displays the selected file on the chat screen.

7. Send File: This button, on click, must send the file chosen to the server. It should not do anything if no file is chosen.

8. Chat screen: This is the area where the messages from server, necessary messaged displayed by client and the loaded file appears and become visible to the user.

9. Text Screen: This is the area where user enters the texts to be sent to the sever.

10. User box: This is the area where user must enter the user name. Once entered and established a connection, this area should not be available for data entry unless disconnected.

2. Server

Server has following functionalities:

1. Start Server: On click, the server must start and the socket should be opened for incoming connections.

2. Stop Server: On click, the server must broadcast to all connected users that it is going to stop, disconnect all users and close the socket.

3. Connected Users: This button must display the connected users on the screen.

4. Clear Screen: This button should clear the data in the chat screen and bring the cursor to the starting.

5. Chat screen: This is the area where all incoming messages from different clients and outgoing messages to all connected users appear with a time stamp.

Some of the other design criteria and strategies were as below:

1. All exceptions are caught and reported.
2. Methods has been developed for all functionalities and has been called from other methods.
3. Minimal amount of communication between server and client.
4. Code words to make client and server understand the purpose of communication.
5. Attractive colorful GUI.

5. Source Code

Client

```
package chat_client;
import java.util.*;
import java.net.*;
import java.io.*;
import java.text.SimpleDateFormat;
import javax.swing.JFileChooser;

public class Chat_client extends javax.swing.JFrame
{
    //Declarations
    String user_name, server_address = "localhost"; //Host IP
    int port_number = 4009;
    Boolean Connection_status = false; //To check if the current thread is connected
    public static Date date = new Date();
    public static SimpleDateFormat format = new SimpleDateFormat ("MM/dd/yy, h:mm a, ");
    public File file;
    Thread incoming_reader;
    ArrayList<String> users_list = new ArrayList();
    Socket socket;
    BufferedReader read_buffer;
    PrintWriter write_buffer;


    //The function which invokes the thread start
    public void thread_listen()
    {
        if(incoming_reader == null)
        {
            incoming_reader = new Thread(new incoming_reader());
            incoming_reader.start();
        }
    }

    //The function to add the connected user to the user lisr
    public void add_user(String data)
    {
        users_list.add(data);
    }

    //The function print the connected users on screen
    public void users_write()
    {
        String[] list_temp = new String[(users_list.size())];
        users_list.toArray(list_temp);
        for (String token:list_temp)
        {
            text_screen.append("                                                                    "+token + "\n");
        }
    }

    //Function to send server that the client is disconnecting.
    public void disconnect_send()
    {
        String exit1 = (user_name + ": :Disconnect");
        try
        {
            write_buffer.println(exit1);
            write_buffer.flush();
        }
        catch (Exception ex)
        {
            text_screen.append(format.format(new Date())+"Not Disconnected. Try again.\n");
        }
```

```
    }

    //Function to print on a client's screnn that it has been disconnected
    public void Disconnected()
    {
        try
        {
            text_screen.append(format.format(new Date())+"Disconnected.\n");
            socket.close();
        }  catch(Exception ex)
        {
            text_screen.append(format.format(new Date())+"Failed to disconnect. \n");
        }
        Connection_status = false;
        username_box.setEditable(true);
    }

    //Initilaise the GUi
    public Chat_client()
    {
        initComponents();
    }

    // Function Remove a user if disconnected
    public void userRemove(String data)
    {
         text_screen.append(data + " is now offline.\n");
    }
    //Thread run
    public class incoming_reader implements Runnable
    {
        @Override
        public void run()
        {
            while(incoming_reader !=null)
            {
                String[] split_data;
                String stream, connected_status = "Sending Connected Users", connect = "Connect", disconnect =
"disconnected", chat = "Chat";
                String red = "reduntant", file = "File", stop = "Stop";
                try
                {
                    while ((stream = read_buffer.readLine()) !=null)
                    {
                        text_screen.setCaretPosition(text_screen.getDocument().getLength());
                        split_data = stream.split(":");//Splits incomimg data at occurances of ":"
                        if(split_data[0].equals(red))//Checks if server told the name entered is already in use
                        {
                            text_screen.append(format.format(new Date())+" Illegal User  Entry. Disconnect and
Reconnect. User :"+ user_name +" : Please exit and connect again \n");
                        }
                        else
                        {
                            if (split_data[2].equals(chat)) // Checks if incoming data is a chat. If yes, displays
                            {
                                text_screen.append(format.format(new Date())+split_data[0] + ": " + split_data[1] +
"\n");
                                text_screen.setCaretPosition(text_screen.getDocument().getLength());
                            }
                            else if (split_data[2].equals(connect))//Check if incoming data is about anew connection
occured in the chat room. If yes, add the new user to user list
                            {
                                add_user(split_data[0]);
                            }
                            else if (split_data[2].equals(connected_status)) // If connected, displays all the connected
users.
                            {
                                text_screen.append(format.format(new Date())+"Currently Available Users : \n");
                                users_write();
                                users_list.clear();
                            }
```

```
                    else if (split_data[2].equals(disconnect)) // check if incomimng data is about a user
disconnecting. If yes, remove the user from the list
                    {
                       userRemove(split_data[0]);
                    }
                     else if (split_data[2].equals(stop)) //Check if server is stopping. If yes, exits.
                    {
                       Thread.sleep(600);
                       Disconnected();
                       System.exit(0);
                    }
                    else if (split_data[2].equals(file))//Check if incoming data is a file transfer. If yes, displays
the file.
                    {
                       text_screen.append(format.format(new Date())+"Received file from :
"+split_data[0]+"\n");
                       text_screen.setCaretPosition(text_screen.getDocument().getLength());
                       text_screen.append(format.format(new Date())+"Displaying Content from file sent by  :
"+split_data[0]+"\n");
                       while ((stream = read_buffer.readLine()) !=null)
                       {
                          text_screen.append(stream + "\n");//Displays the file
                       }
                    }
                 }
              }
           }
           catch(Exception ex)
           { }
        }
     }
  }

  //Function to filter text files
  class MyCustomFilter extends javax.swing.filechooser.FileFilter
  {
     @Override
     public boolean accept(File file)
     {
        // Allow only directories, or files with ".txt" extension
        return file.isDirectory() || file.getAbsolutePath().endsWith(".txt");
     }
     @Override
     public String getDescription()
     {
        // This description will be displayed in the dialog
        return "Text documents (*.txt)";
     }
  }

  @SuppressWarnings("unchecked")
  // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
  private void initComponents() {

     fileChooser = new javax.swing.JFileChooser();
     title = new javax.swing.JLabel();
     username_box = new javax.swing.JTextField();
     connect_button = new javax.swing.JButton();
     disconnect_button = new javax.swing.JButton();
     anonymous_login = new javax.swing.JButton();
     jScrollPane1 = new javax.swing.JScrollPane();
     text_screen = new javax.swing.JTextArea();
     enter_text = new javax.swing.JTextField();
     send_button = new javax.swing.JButton();
     username_add = new javax.swing.JLabel();
     jButton1 = new javax.swing.JButton();
     send_file = new javax.swing.JButton();
     jMenuBar1 = new javax.swing.JMenuBar();
     jMenu1 = new javax.swing.JMenu();
     OPEN = new javax.swing.JMenuItem();
     Exit = new javax.swing.JMenuItem();
```

```
fileChooser.setDialogTitle("This is my open dialog box");
fileChooser.setFileFilter(new MyCustomFilter());

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Chat - Client's frame");
setName("client"); // NOI18N
setResizable(false);

title.setText("CLIENT WINDOW");

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

connect_button.setText("CONNECT");
connect_button.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        connect_user(evt);
    }
});

disconnect_button.setText("DISCONNECT");
disconnect_button.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        disconnect_user(evt);
    }
});

anonymous_login.setText("ANONYMOUS LOGIN");
anonymous_login.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        anonymous_user(evt);
    }
});

text_screen.setBackground(new java.awt.Color(255, 204, 204));
text_screen.setColumns(20);
text_screen.setRows(5);
jScrollPane1.setViewportView(text_screen);

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

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

username_add.setText("USERNAME : ");

jButton1.setText("CLEAR SCREEN");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        clear_screen(evt);
    }
});

send_file.setText("SEND FILE");
send_file.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        send_fileActionPerformed(evt);
    }
});
```

```
        jMenu1.setText("SELECT TEXT FILE TO SEND");

        OPEN.setText("OPEN");
        OPEN.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                OPENActionPerformed(evt);
            }
        });
        jMenu1.add(OPEN);

        Exit.setText("EXIT");
        Exit.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ExitActionPerformed(evt);
            }
        });
        jMenu1.add(Exit);

        jMenuBar1.add(jMenu1);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(30, 30, 30)
                .addComponent(enter_text, javax.swing.GroupLayout.PREFERRED_SIZE, 450,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                    .addComponent(send_button, javax.swing.GroupLayout.DEFAULT_SIZE, 114, Short.MAX_VALUE)
                    .addComponent(jButton1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addGap(0, 0, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(username_add)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(username_box, javax.swing.GroupLayout.PREFERRED_SIZE, 163,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addComponent(anonymous_login))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(connect_button, javax.swing.GroupLayout.PREFERRED_SIZE, 99,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(disconnect_button, javax.swing.GroupLayout.PREFERRED_SIZE, 120,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(21, 21, 21))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(title, javax.swing.GroupLayout.PREFERRED_SIZE, 140,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(38, 38, 38)
                        .addComponent(send_file)
                        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 668,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
```

```
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(title, javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(send_file))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 413,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(disconnect_button)
                    .addComponent(connect_button, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(username_add, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(username_box, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addComponent(anonymous_login, javax.swing.GroupLayout.PREFERRED_SIZE, 28,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(15, 15, 15)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                .addComponent(enter_text, javax.swing.GroupLayout.PREFERRED_SIZE, 52,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(send_button)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jButton1)))
            .addContainerGap(13, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void connect_user(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_connect_user
        thread_listen();//Invokes thread
        if (Connection_status == false)
        {
            user_name = username_box.getText();
            username_box.setEditable(false);
            try
            {
                socket = new Socket(server_address, port_number);
                InputStreamReader streamreader = new InputStreamReader(socket.getInputStream());
                read_buffer = new BufferedReader(streamreader);
                write_buffer = new PrintWriter(socket.getOutputStream());
                write_buffer.println(user_name + ":has connected.:Connect");//Sends server about the new user
connection
                write_buffer.flush();
                Connection_status = true;
            }
            catch (Exception ex)
            {
                text_screen.append(format.format(new Date())+"Cannot Connect! Try Again. \n");
                username_box.setEditable(true);
            }
        }
        else if (Connection_status == true)
        {
            text_screen.append(format.format(new Date())+"You are already connected. \n");//Dsiplays if already
connected
        }
    }//GEN-LAST:event_connect_user

    private void disconnect_user(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_disconnect_user
        disconnect_send();
        Disconnected();
    }//GEN-LAST:event_disconnect_user
```

```
    private void anonymous_user(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_anonymous_user
        username_box.setText("");
        if (Connection_status == false)
        {
            String anon="anon"; //Gives a name for the anonymous user
            Random generator = new Random();
            int i = generator.nextInt(999) + 1;
            String is=String.valueOf(i);
            anon=anon.concat(is);
            user_name=anon;
            username_box.setText(anon);
            username_box.setEditable(false);
            try
            {
                socket = new Socket(server_address, port_number);
                InputStreamReader streamreader = new InputStreamReader(socket.getInputStream());
                read_buffer = new BufferedReader(streamreader);
                write_buffer = new PrintWriter(socket.getOutputStream());
                write_buffer.println(anon + ":has connected.:Connect"); // Sends the user name to server
                write_buffer.flush();
                Connection_status = true;
            }
            catch (Exception ex)
            {
                text_screen.append(format.format(new Date())+"Cannot Connect! Try Again. \n");
                username_box.setEditable(true);
            }
            thread_listen();//Invokes thread
        }
        else if (Connection_status == true)
            {
                text_screen.append(format.format(new Date())+"You are already connected. \n");//Displays if
connection is active
            }
    }//GEN-LAST:event_anonymous_user

    private void send_text(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_send_text
        String nothing = "";
        if ((enter_text.getText()).equals(nothing)) // Waits for user to enter data
        {
            enter_text.setText("");
            enter_text.requestFocus();
        }
        else
        {
            try
            {
              write_buffer.println(user_name + ":" + enter_text.getText() + ":" + "Chat");// Sends the chat text
entered by the user to server
              write_buffer.flush(); // flushes the buffer
            }
            catch (Exception ex)
            {
                text_screen.append(format.format(new Date())+"Message was not sent. \n");
            }
            enter_text.setText("");
            enter_text.requestFocus();
        }
        enter_text.setText("");
        enter_text.requestFocus();
    }//GEN-LAST:event_send_text

    private void text_username(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_text_username

    }//GEN-LAST:event_text_username

    private void clear_screen(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_clear_screen
        text_screen.setText("");
    }//GEN-LAST:event_clear_screen

    private void enter_chat(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_enter_chat
```

```
    }//GEN-LAST:event_enter_chat

    private void OPENActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_OPENActionPerformed
        int returnVal = fileChooser.showOpenDialog(this);
        try
        {
            if (returnVal == JFileChooser.APPROVE_OPTION) //Reads the file via file chooser
            {
                file = fileChooser.getSelectedFile();
                text_screen.read( new FileReader( file.getAbsolutePath() ), null );
            }
            else
            {
                System.out.println("File access cancelled by user.");
            }
        }
        catch (Exception ex)
        {}
        enter_text.setText("");
        enter_text.requestFocus();
    }//GEN-LAST:event_OPENActionPerformed

    private void ExitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_ExitActionPerformed
        System.exit(0);
    }//GEN-LAST:event_ExitActionPerformed

    private void send_fileActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_send_fileActionPerformed
        try
        {
            write_buffer = new PrintWriter(socket.getOutputStream(),true);
            write_buffer.println(user_name + ":is sending a file.:file"); //sends the information about file transfer
to server
            write_buffer.flush();
            FileInputStream fis = new FileInputStream(file); //Opens file stream
            byte[] buffer = new byte[4096];
            String fileCon = "";
            while (fis.read(buffer) > 0)  // Receives the file
            {
                fileCon+=new String(buffer);
            }
            write_buffer.println(fileCon);
            write_buffer.println("end"); // Determines the end of the file
            fis.close(); // Close file stream
            text_screen.append(format.format(new Date())+"Sent file \n"); // Displays fil has been sent
            text_screen.setCaretPosition(text_screen.getDocument().getLength());
        }
        catch (Exception ex)
        {}
    }//GEN-LAST:event_send_fileActionPerformed

    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable() //Invoke GUI
        {
            @Override
            public void run()
            {
                new Chat_client().setVisible(true);// Make Visible
            }
        }
        );
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JMenuItem Exit;
    private javax.swing.JMenuItem OPEN;
    private javax.swing.JButton anonymous_login;
    private javax.swing.JButton connect_button;
```

```
    private javax.swing.JButton disconnect_button;
    private javax.swing.JTextField enter_text;
    private javax.swing.JFileChooser fileChooser;
    private javax.swing.JButton jButton1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton send_button;
    private javax.swing.JButton send_file;
    private javax.swing.JTextArea text_screen;
    private javax.swing.JLabel title;
    private javax.swing.JLabel username_add;
    private javax.swing.JTextField username_box;
    // End of variables declaration//GEN-END:variables
}
```

Server

```
package chat_server;
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.SimpleDateFormat;

public class Chat_server extends javax.swing.JFrame
{   //Declarations
    ArrayList clientOutputStreams;
    ArrayList<String> users;
    public static Date date = new Date();
    public static SimpleDateFormat format = new SimpleDateFormat ("MM/dd/yy, h:mm a, ");
    public Socket socket1;
    String red;
    PrintWriter client_writer;

    //Run
    public class ServerStart implements Runnable
    {
        @Override
        public void run()
        {
            clientOutputStreams = new ArrayList();//Array list to store connected users
            users = new ArrayList();
            try
            {
                ServerSocket serverSock = new ServerSocket(4009);
                while (true)
                {
                    Socket clientSock = serverSock.accept();
                    client_writer = new PrintWriter(clientSock.getOutputStream());
                    clientOutputStreams.add(client_writer);
                    Thread listener = new Thread(new ClientHandler(clientSock, client_writer));
                    listener.start();//Start the thered
                    chat_print.append(format.format(new Date())+"Incoming Connection \n");
                }
            }
            catch (Exception ex)
            {
                chat_print.append(format.format(new Date())+"Error making a connection \n");
            }
        }
    }

   //Thread
   public class ClientHandler implements Runnable
   {
        BufferedReader client_buffer;
        PrintWriter client;
        public ClientHandler(Socket clientSocket, PrintWriter user)
        {
```

```
            client = user;
            try
            {
               socket1 = clientSocket;
               InputStreamReader client_reader = new InputStreamReader(socket1.getInputStream());
               client_buffer = new BufferedReader(client_reader);
            }
            catch (Exception ex)
            {
               chat_print.append(format.format(new Date())+"Unexpected error... \n");
            }

        }

    //Method to send about the disconnected user to other connected users.
    void remove_user (String name)
    {
       broadcast(name + ": is now offline :Chat");
    }

    //Method to broacast the news about usage of reduntant name
    public void reduntant_name()
    {
       red = "reduntant";
       broadcast(red +": :");
    }

        @Override
        public void run()
        {
            String message, connect = "Connect", disconnect = "Disconnect", chat = "Chat" ;
            String[] split_data;
            String file1= "file";
            try
            {
               while ((message = client_buffer.readLine()) != null)
               {
                  split_data = message.split(":");//Split the incoming data at occurances of ":"
                  if ((users.indexOf(split_data[0]) != -1)&&(split_data[2].equals(connect)))//Check is incomimg
uer name is already in use
                  {
                     reduntant_name();
                  }
                  else
                  {
                     chat_print.append(format.format(new Date())+"Received connection from : " + split_data[0]
+ "\n");
                     if (split_data[2].equals(connect)) //check in incomimg information is about a new user
connection
                     {
                        broadcast((split_data[0] + ":" + split_data[1] + " Available :" + chat));
                        Add_user(split_data[0]);
                     }
                     else if (split_data[2].equals(disconnect)) //check if incoming information is about a
discnnecting user.If yes, delete the user and brodcast the news
                     {
                        broadcast((split_data[0] + ": has disconnected. Unavailable " + ":" + chat));
                        remove_user(split_data[0]);
                        Delete_user(split_data[0]);

                     }
                     else if (split_data[2].equals(chat))//Check if a user wants to chat and send texts to chat room.
If yes broadcast the message
                     {
                        broadcast(message);
                     }
                     else if (split_data[2].equals(file1))// Check if incoming information is about a new file transfer.
If yes, receive the file and brodcaste it
                     {
                        chat_print.append(format.format(new Date())+"Receiving file from : " + split_data[0]);
                        receive_file(split_data[0],client_buffer);
```

```
                    chat_print.setCaretPosition(chat_print.getDocument().getLength());
                }
                else
                {
                    chat_print.append(format.format(new Date())+"Invalid User entry \n");
                }
            }
          }
       }
       catch (Exception ex)
       {
          chat_print.append(format.format(new Date())+"Lost a connection. \n");
          ex.printStackTrace();
          clientOutputStreams.remove(client);
       }
        }
 }

//Method to receive the incomig file.
public void receive_file(String user,BufferedReader client_buffer)
{
   try
   {
       FileOutputStream fos = new FileOutputStream("testfile.txt");
       byte[] buffer = new byte[4096];
       int filesize = 15123; // Send file size in separate msg
       int read = 0;
       int totalRead = 0;
       int remaining = filesize;
       String fileCon = null;
       broadcast(user+":Received file:File");//brodcaste the message that a file has been received
       while((fileCon=client_buffer.readLine())!=null)
       {
          if(fileCon.equals("end"))
             break;
          totalRead += read;
          remaining -= read;
          chat_print.append("\n"+ fileCon);
          fos.write(buffer, 0, read);
          brodcast_file(fileCon);//brodcaste the incomimg file.
       }
       client_writer.println("null");
       chat_print.setCaretPosition(chat_print.getDocument().getLength());
       fos.close();
        }
   catch (Exception ex)
   {}
}

//Method to brodcaste the incoming file.
public void brodcast_file(String fileCon)
{
    Iterator iterator1 = clientOutputStreams.iterator();//to broadcaste to all users
    try
    {
       while (iterator1.hasNext())
       {
       PrintWriter client_writer = (PrintWriter) iterator1.next();
       client_writer.println(fileCon);
       client_writer.flush();
       }
    }
    catch (Exception ex)
    {
       chat_print.append(format.format(new Date())+"Error in broadcasting. \n");
    }
}

//Method to broadcaste the message
 public void broadcast(String message)
 {
```

```
        Iterator iterator1 = clientOutputStreams.iterator();//to brodcaste to all users
      int flag =0;
      try
      {
         while (iterator1.hasNext())
         {
            flag++;
            PrintWriter client_writer = (PrintWriter) iterator1.next();
                  client_writer.println(message);//write message to the client
                  chat_print.append(format.format(new Date())+"Sending to user " +flag +" -> " + message
+"\n");
            client_writer.flush();
            chat_print.setCaretPosition(chat_print.getDocument().getLength());
         }
      }
      catch (Exception ex)
      {
         chat_print.append(format.format(new Date())+"Error in broadcasting. \n");
      }
   }

   //Method to delete the disconnected user
   public void Delete_user (String User_name)
   {
      String message, add = ": :Connect", connected_status = "Server: :Sending Connected Users";
      users.remove(User_name);
      String[] tempList = new String[(users.size())];
      users.toArray(tempList);
      for (String token:tempList)
      {
         message = (token + add);
         broadcast(message);
      }
      broadcast(connected_status);
   }

 //Method to add the new user to the user list and broadcaste the new connection
  public void Add_user (String User_name)
   {
      String message, add = ": :Connect", connected_status = "Server: :Sending Connected Users", name =
User_name;
      chat_print.append(format.format(new Date())+ " Adding User " + name + "\n");
      users.add(name);
      chat_print.append(format.format(new Date())+"Added user " + name + "\n");
      String[] tempList = new String[(users.size())];
      users.toArray(tempList);
      for (String token:tempList)
      {
         message = (token + add);
         broadcast(message);
      }
      broadcast(connected_status);
   }

   @SuppressWarnings("unchecked")
   // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
   private void initComponents() {

      jScrollPane1 = new javax.swing.JScrollPane();
      chat_print = new javax.swing.JTextArea();
      start_server_button = new javax.swing.JButton();
      stop_server_button = new javax.swing.JButton();
      connected_users_botton = new javax.swing.JButton();
      clear_button = new javax.swing.JButton();
      jLabel1 = new javax.swing.JLabel();

      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
      setTitle("Chat - Server's frame");
      setForeground(java.awt.Color.yellow);
      setName("server"); // NOI18N
      setResizable(false);
```

```
        chat_print.setBackground(new java.awt.Color(255, 204, 204));
        chat_print.setColumns(20);
        chat_print.setForeground(new java.awt.Color(51, 0, 102));
        chat_print.setLineWrap(true);
        chat_print.setRows(5);
        jScrollPane1.setViewportView(chat_print);

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

        stop_server_button.setText("STOP SERVER");
        stop_server_button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                stop_server_on_click(evt);
            }
        });

        connected_users_botton.setText("CONNECTED USERS");
        connected_users_botton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                display_users_on_click(evt);
            }
        });

        clear_button.setText("CLEAR SCREEN");
        clear_button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                clear_screen_on_click(evt);
            }
        });

        jLabel1.setText("SERVER WINDOW");

        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)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jScrollPane1)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(start_server_button, javax.swing.GroupLayout.PREFERRED_SIZE, 131,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(stop_server_button, javax.swing.GroupLayout.PREFERRED_SIZE, 137,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel1)
                        .addGap(129, 129, 129)
                        .addComponent(clear_button, javax.swing.GroupLayout.DEFAULT_SIZE, 108,
Short.MAX_VALUE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(connected_users_botton)
                        .addGap(20, 20, 20))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
```

```
                .addComponent(clear_button, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(connected_users_botton))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 477,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(start_server_button)
                .addComponent(stop_server_button))
            .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void stop_server_on_click(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_stop_server_on_click
        try
        {
            broadcast("Server:is stopping and all users will be disconnected :Chat"); // broadcaste the server is
stopping
            broadcast("Server:is stopping and all users will be disconnected :Stop");
            chat_print.append(format.format(new Date())+"Server stopping... \n");
            socket1.close();
            Thread.sleep(600);
            System.exit(0);//close server
        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
    }//GEN-LAST:event_stop_server_on_click

    private void start_server_on_click(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_start_server_on_click
        Thread starter = new Thread(new ServerStart());// start the server on click
        starter.start();
        chat_print.append(format.format(new Date())+"Server started. \n");
    }//GEN-LAST:event_start_server_on_click

    private void display_users_on_click(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_display_users_on_click
        chat_print.append(format.format(new Date())+"Online users : \n");
        for (String current_user : users)//display connected users
        {
            chat_print.append("                                                    "+current_user);
            chat_print.append("\n");
        }
    }//GEN-LAST:event_display_users_on_click

    private void clear_screen_on_click(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_clear_screen_on_click
        chat_print.setText("");//CLeat screen
    }//GEN-LAST:event_clear_screen_on_click

    //Initialise GUI
    public Chat_server()
    {
        initComponents();
    }

    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable() //Invoke GUI
        {
            @Override
            public void run()
            {
                new Chat_server().setVisible(true);//Set visible
```

```
        }
      }
      );
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JTextArea chat_print;
    private javax.swing.JButton clear_button;
    private javax.swing.JButton connected_users_botton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton start_server_button;
    private javax.swing.JButton stop_server_button;
    // End of variables declaration//GEN-END:variables
}
```

7. Reference

1. https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html
2. https://docs.oracle.com/javase/tutorial/networking/sockets/
3. https://netbeans.org/kb/docs/java/gui-functionality.html
4. https://blog.udemy.com/java-threads/
5. http://opensourceforu.com/2015/03/fundamental-of-java-multi-threading/
6. https://www.tutorialspoint.com/java/java_files_io.htm