

Report

On

ChatApp

(A Chat Server Application)

Greater Noida College of Technology

(Dr. APJ Abdul Kalam Technical University, Lucknow)

Department of

Computer Science & Engineering

(NCS-753)

Submitted To :

Ms. Anjali Mittal

Submitted By :

Raja Kumar

TABLE OF CONTENTS

Page

DECLARATION

ii

CERTIFICATE	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	v
INTRODUCTION, BACKGROUND OF THE PROBLEM	7
PROJECT DEVELOPMENT LIFE CYCLE	11
1. REQUIREMENT ANALYSIS.....	11
2. HIGH –LEVEL DESIGN.....	11
3. LOW-LEVEL DESIGN.....	14
a. LOGIN INTERFACE.....	
i. INTRODUCTION.....	15
ii. INTERACTION WITH CHAT-SERVER.....	16
iii. CODE FOR LOGIN INTERFACE.....	14
b. REGISTRATION INTERFACE.....	
i. INTRODUCTION.....	17
ii. INTERACTION WITH CHAT-SERVER.....	18
iii. CODE FOR LOGIN INTERFACE.....	19
c. CHAT INTERFACE.....	
i. INTRODUCTION.....	26
ii. INTERACTION WITH CHAT-SERVER.....	27
iii. CODE FOR LOGIN INTERFACE.....	27
d. CHAT SERVER.....	
i. INTRODUCTION.....	33
ii. INTERACTION WITH CLIENT INTERFACE	33
iii. CODE FOR LOGIN INTERFACE.....	34
4. CONSTRUCTION.....	38
5. TESTING.....	38
6. ACCEPTANCE.....	38
APPENDIX	
i. REFERENCES	

CERTIFICATE

This is to certify that Project Report entitled “*ChatApp*” which is submitted by ***Raja Kumar*** in partial fulfillment of the requirement for the award of degree B. Tech. in Department of ***Computer Science & Engineering***. of Dr. APJ Abdul Kalam Technical University, Lucknow, is a record of the candidate own work carried out by him under my/our supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

HOD, CSE

Mr. Nawneet Pandey

PROJECT GUIDE

Mr. Rahul Chauhan

DECLARATION

We hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature

Name : Raja Kumar

Roll No. : 1581010034

Date : 03-12-18

ACKNOWLEDGEMENT

*It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Final Year. We owe special debt of gratitude to **Mr. Nawneet Pandey**, Department of Computer Science & Engineering, **Greater Noida College of Technology, Greater Noida** for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavors have seen light of the day.*

We also take the opportunity to acknowledge the contribution of Mr. Rahul Chauhan, from Incapp , Greater Noida for his full support and assistance during the development of the project.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Date : 03-12-18

Raja Kumar

ABSTRACT

This project is a report in partial fulfillment of the requirements for the award of a degree of Bachelor of Technology in Computer Science. The purpose of this project was to develop an window based chat application (ChatApp) to be used in any website and can be used on java platforms as well as java supported platform. ChatApp is important because it allows for real-time online conversation. The system is meant to enable the general users (Students, Lecturer, Employees & staff members) to interact online with the College or any organizational staffs without meeting physically . The system is necessary for registering and updating the user database.

INTRODUCTION

A chat application is utility software that enables users to communicate over networks. A chat application can be efficiently used as a medium for various forums. The functionalities provided by chat applications include displaying messages in a chat room, displaying a list of online users, and enabling users to send personal messages to other users. In addition, chat applications enable you to communicate with chat friends. We have built a chat application by using Java concepts. The use of Java, which is a platform-independent language, will enable you to run your chat application on any platform. The concepts that we have used in chat application are:

- Java Swing API
- Networking
- Socket programming
- I/O concepts
- Event handling
- Error and exception handling

Java Swing API classes will be used to create graphical interfaces of the application. A chat application is a network-based application. I'll use networking and socket programming to establish connections between users over a network. In addition, the I/O classes of Java will be used extensively in this application. These classes can be used for sending messages over a network to the input and output streams established using socket programming. Various events, errors, and exceptions will be handled effectively in this application.

First Thing First

Sockets

Our communications between client and server will pass through a Java object called a Socket. Sockets are not at all Java-specific; the term is taken directly from the terminology of general IP (Internet Protocol) network programming. In Java programs, a Socket object is simply a wrapper around the low-level sockets that Internet programmers have been using for years. And the abstraction used by the Java language is very clean, so socket programming in the Java language is much more pleasant than it is, for example, in C.

The most important thing to know about a Socket object is that it contains (among other things) two Streams. One is for reading data coming in, and the other is for writing data out. That is to say, a Socket has an `InputStream` and an `OutputStream`. (If these Stream classes are unfamiliar to you, then suffice it to say that they are objects used for reading

and writing data, often as a stream of bytes. If you don't know about them yet, you really should. See the *java.io* package for more information.)

What does the server do?

Before we describe the Listener class, we'll describe the server. Doing so has a certain chronological elegance, because in our running system, the server will have to start before any of the clients can connect to it.

Our server will be a stand-alone program -- a single Java process running on its own machine. It won't require any support software other than a Java virtual machine. And it won't require a Web server or application server, although a Web server or application server will likely be used to serve the client applet to the client.

More advanced server systems often embed the server code within a larger framework. This framework might be used to supply features such as load-balancing, special libraries for handling large numbers of clients, process migration, and database services. However, our example is going to stand all by itself. It will take care of all networking responsibilities on its own. As we'll see, this isn't very hard.

Listening on a port

The first thing we have to do is to get ready to receive incoming connections. To do this, we must listen on a port.

A port can be thought of as an address within a single computer. Remember that often a single machine might serve as a Web server, a chat server, an FTP server, and several other kinds of servers at the same time. Because of this, a connection to a server needs to specify not only the address of the machine itself, but also the particular service within the machine. This internal address is a port and is represented by a single integer between 1 and 65535.

Many standard services have a dedicated port number. For example, telnet uses port 23, FTP uses ports 20 and 21, and Web servers, by default, use port 80. Since our chat system is not famous (yet), we're going to have to use one of the port numbers allocated for general use.

We'll use port 5000. This means that our server is going to *listen* for connections on port 5000. Our clients, when connecting to our server machine, will specify that they want to connect to port 5000 on our server machine. This way, our clients and our server will be able to talk.

Why use multithreading?

A detailed discussion of threads is beyond the scope of this tutorial. There are a few reasons why you'd want to use threads in your program, but there is one reason most pertinent to the construction of a chat server: input/output.

Your chat server is communicating (in a sense) with the users at the client. Users are usually much slower than servers, which means that your server code is going to spend a lot of time simply waiting for users to say things. And you never know who is going to say something first. If you have a single thread, and it's waiting for user #0 to say something, then it's not going to know that users #1 through #10 are talking like crazy.

For this reason, we're going to create a thread for *each* user connected to the system. The advantage of multithreading is that when one thread is listening for a slow user to say something, it essentially goes to sleep until something comes in from that user. In the meantime, another thread can be receiving data from another user. In effect, multithreading allows us to respond as quickly as we can to each user.

All you really need to know about multithreading for this tutorial is that we're going to create a new thread for each connection. This thread, more or less by itself, is going to take care of that connection until it is severed.

The communications protocol

Now that we've gotten to the point where we're going to be talking to the client, we should talk a little bit about our communication protocol.

Every client/server system has a communications protocol, which is nothing more than the format you use to send the data back and forth. The protocol can be so simple it hardly deserves the title of protocol, or it can be a sophisticated standard that has been ratified by consortia all over the world. Either way, it's a protocol.

We're going to create our own protocol, because in the Java language it's very easy to do, and because there's little for us to gain from using an existing standard. Our protocol will be very simple.

The Java language has a pair of extremely useful classes called `DataInputStream` and `DataOutputStream`. These classes allow you to read and write low-level data objects (like integers and strings) to a stream, without having to consider the format in which they are written. Because these classes use the same format, and because this format doesn't change, you can be sure that an integer written to a `DataOutputStream` will be properly read from the `DataInputStream` at the other end.

So our protocol will be this:

- When a user types something into their chat window, their message will be sent as a string through a `DataOutputStream`.
- When the server receives a message, through a `DataInputStream`, it will send this same message to all users, again as a string through a `DataOutputStream`.
- The users will use a `DataInputStream` to receive the message. And that's it!

Removing dead connections

It is important to inform the main Server object each time a connection has closed, so that the server can remove the connection from any internal lists, and also so that the server doesn't waste time sending messages to clients that are no longer connected.

The importance of cleaning up

This step of removing dead connections is given its own place as one of our seven elements of server construction because it's so crucial, and because it's often forgotten by people writing their first server. If programmers forget this step, they find that their server works fine for a while. Then it begins to get slower and slower, and starts using more and more memory. When they look at the output of the server process, they see exceptions whizzing by at an alarming rate.

What's happened is that the server has kept around all the connections that have died, is trying to send messages to them, and is finding that it can't.

Let's say your server is receiving connections, on average, about once every five seconds.

And let's further assume that the average user stays on about fifteen minutes.

This means you will have, on average, about 180 users on at any given moment. As time passes, many users will connect and disconnect, keeping the average around 180.

After 24 hours, 17,280 users will have connected to your server. At the end of this first day, you'll still have about 180 live connections, and you'll have about 17,100 dead connections. If you haven't taken care to remove these dead connections, then you can be sure that your server is spending most of its time trying, and failing, to write to these connections. (In fact, it might be spending most of its time spewing Exceptions to System.out!) For this reason, it's *crucial* that you clean up after these dead connections.

The Project Life Cycle

The development life cycle of a project usually involves three stages:

- Project initiation
- Project execution
- Project completion
-

In the project initiation stage, a team prepares the project plan and finalizes the outcome of each stage. In this stage, the team also prepares a comprehensive list of tasks involved in this stage, and the project manager assigns responsibilities to the team members, depending on their skills. In the project execution stage, the team develops the product.

In our case, we developed the online chat utility. This stage consists of the following phases :

- Requirements analysis
- High-level design
- Low-level design
- Construction
- Testing
- Acceptance

Requirements Analysis Phase

During the requirements analysis, we will analyze the requirements to be fulfilled by the chat utility and identified the probable approach for meeting these requirements. To identify the requirements for youChat, We studied the existing chat utilities at various Web sites and conducted extensive interviews with chat application users.

List of the requirements for the chat utility :

The application should:

- Enable a first-time user to register by filling in some personal details
- Enable a registered user to log on after his/her login details are validated
- Allow an online user to view a list of other online users
- Allow users to chat in the common chat room

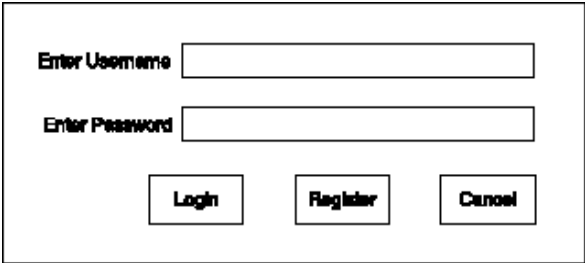
The High-Level Design Phase

In this stage, the team decides how the system should function. The formats for data input and output are finalized in this stage. The functional specifications documentation of the system is presented in a language that can be understood by all. The finished project design is, however, executed only on the project manager's approval.

The Login Interface

The login interface is the first screen that a user will view. It will accept the username and the password for the chat utility. The user will have the option to cancel the login attempt. First-time users can select the option to register with YouChat.

A sketch of the login interface is depicted in Figure 1 :

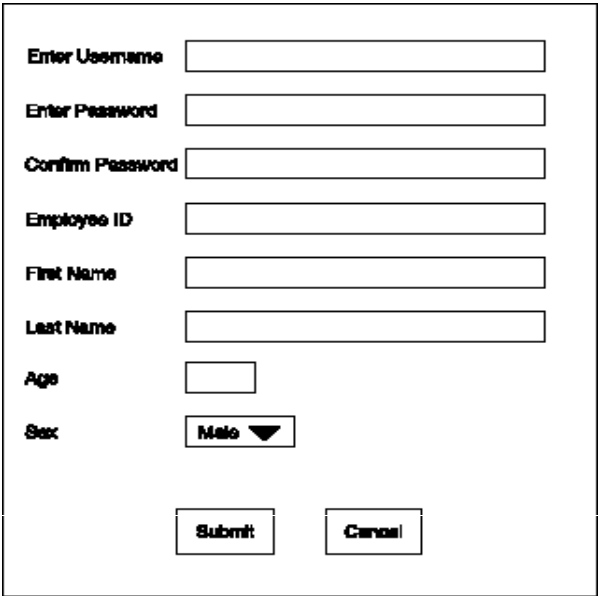


A sketch of a login interface. It features two text input fields: "Enter Username" and "Enter Password". Below these fields are three buttons: "Login", "Register", and "Cancel".

Figure 1 : Sketch of the login interface

The Registration Interface

The registration interface enables users to register with YouChat. The interface will enable a user to specify a username and a password. In addition, other personal information about the user is accepted. The registration interface designed by us after much discussion and consultation is shown in Figure2 :



A sketch of a registration interface. It features several text input fields: "Enter Username", "Enter Password", "Confirm Password", "Employee ID", "First Name", and "Last Name". There is also a small input field for "Age" and a dropdown menu for "Sex" with "Male" selected. At the bottom are two buttons: "Submit" and "Cancel".

Figure 2 : Sketch of the registration interface

The Chat Room Interface

After a registered user is authenticated, he/she will enter the chat room. In the chat room interface, a user can view the messages sent by other online users. In addition, the names of the online users will be visible.

The chat room, as designed by us, is illustrated in Figure 3 :

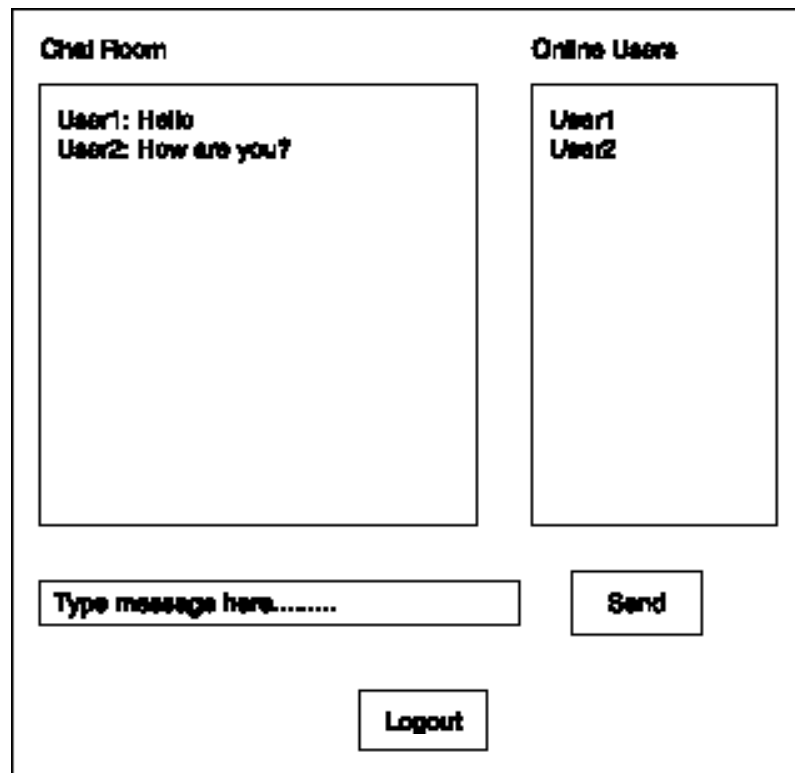


Figure 3 : Sketch of the chat room interface

The Low-Level Design Phase

In this phase, a detailed design of the software modules, based on the high-level design, is produced. In addition, the team also lays down specifications for the various software modules of an application. We decided on a number of Java classes, names of these classes, and other such details for the project. The functionality of the YouChat application and the interaction between the chatserver and various interfaces in the application are illustrated in Figure :

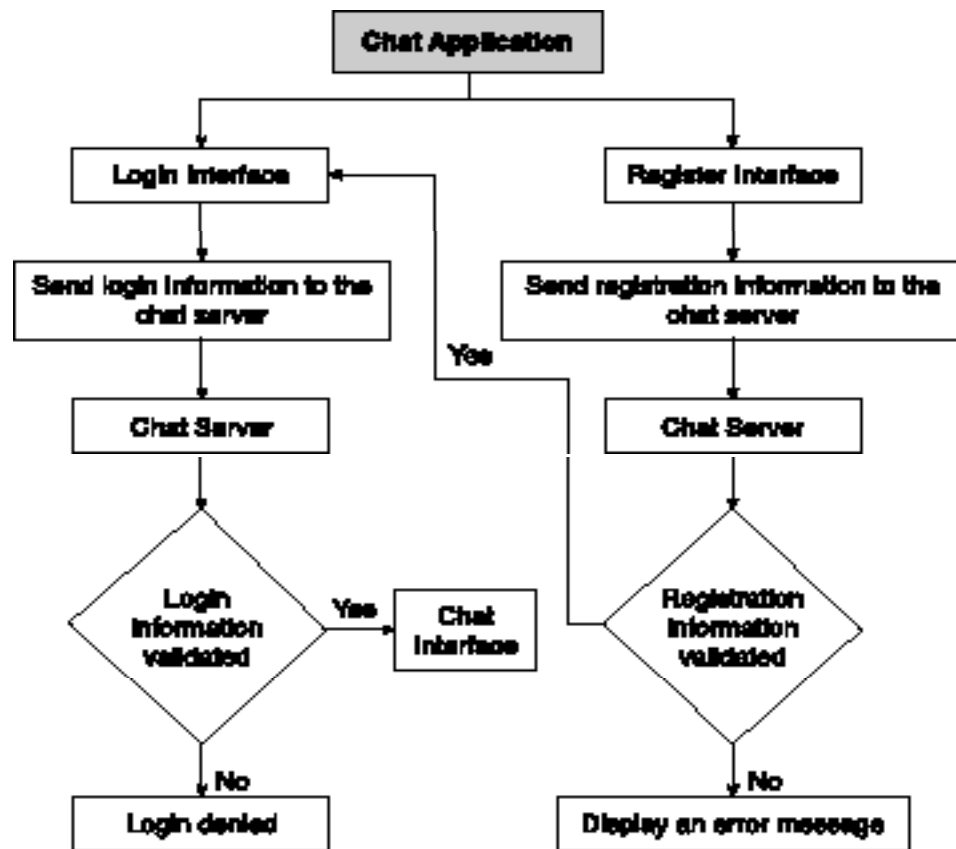


Figure 4 : Interaction between login interface and chat server

Designing the Login Interface

As per the high-level design, the YouChat application allow only registered users to log in and chat. The login information generally includes usernames and passwords. Some applications may require users to provide other information such as their name and date of birth. We has decided to accept only the username and the password from a user.

Since the login interface is a network client application, the user's credentials are validated by the server application of YouChat. The username and the password of the user are sent to the server, which validates the information. Upon successful validation, the user is directed to the chat room.

For unregistered users, the option to register with the YouChat application needs to be provided. In addition, a user can cancel the login attempt. Therefore, the options for registration and cancellation are provided in the login interface.

These functionalities of the login screen identified during the design phase are listed as follows:

- When the user clicks the Login button, the login information is passed to the server and the information is verified at the server end.
- The login screen has an option for enabling first-time users to register with the YouChat application. A user can access the registration screen by clicking the Register button on the login screen.
- In case a user decides to quit the login attempt, the user can do so by clicking the Cancel button on the login screen.

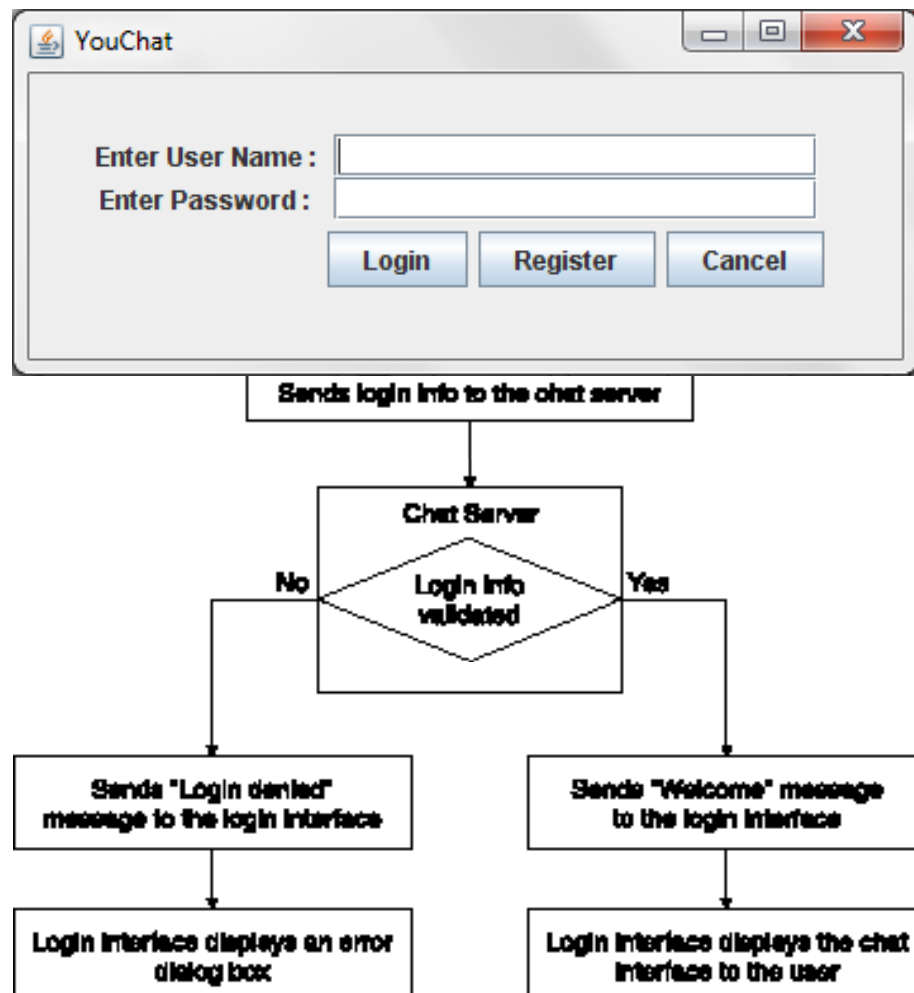


Figure 5 : Interaction between login interface and Chat Server

CODE FOR LOGIN INTERFACE :

Class Name : Login.java

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

class Login extends JFrame implements ActionListener
{
    JLabel lb1,lb2;
    JTextField tf;
    JPasswordField pf;
    JButton b1,b2,b3;
    String uname;
    char[] upass;
    String spass;
    Socket toServer;
    ObjectInputStream streamFromServer;
    PrintStream streamToServer;
    JFrame f;
```



```

public static void main(String g[])
{
    new Login();
}

public Login()
{
    JPanel p1=new JPanel(new GridBagLayout());
    GridBagConstraints gbc=new GridBagConstraints();

    gbc.gridx=0;
    gbc.gridy=0;
    lb1=new JLabel("Enter User Name :");
    p1.add(lb1,gbc);

    gbc.gridx=1;
    gbc.gridy=0;
    tf=new JTextField(20);
    p1.add(tf,gbc);

    gbc.gridx=0;
    gbc.gridy=1;
    lb2=new JLabel("Enter Password :");
    p1.add(lb2,gbc);

    gbc.gridx=1;
    gbc.gridy=1;
    pf=new JPasswordField(20);
    p1.add(pf,gbc);

    JPanel p2=new JPanel();

    b1=new JButton("Login");
    p2.add(b1);
    b1.addActionListener(this);

    b2=new JButton("Register");
    p2.add(b2);
    b2.addActionListener(this);

    b3=new JButton("Cancel");
    p2.add(b3);
    b3.addActionListener(this);

    gbc.gridx=1;
    gbc.gridy=3;

```

```

p1.add(p2,gbc);

f=new JFrame("YouChat");
f.add(p1);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setSize(420,170);
f.setVisible(true);
}                                     // end of constructor

public void actionPerformed(ActionEvent ae)
{
    JButton b=(JButton)ae.getSource();
    if(b.equals(b3)) {
        f.dispose();
    }
    else if(b.equals(b2))
    {
        new Register();
        f.dispose();
    }
    else {
        try{
            toServer =new Socket("Localhost",7777);           // creater client socket
            streamFromServer=new ObjectInputStream(toServer.getInputStream());
            streamToServer = new PrintStream(toServer.getOutputStream());
            streamToServer.println("LoginInfo");               //send message to
server for login

            uname=tf.getText();
            upass=pf.getPassword();
            spass=new String(upass);
            streamToServer.println(uname + ":" + spass); //send user name & password to the
server

            String          frmServer=(String)streamFromServer.readObject();
            if(frmServer.equals("Welcome"))
            {
                new ClientInt(uname);                           //start the client chat
screen
                f.dispose();
            }
            else {
                showerrordlg();
            }
        } //end of try

        catch(Exception e)    {

```

```

                System.out.println("Exception Occured : "+ e);
            }
        }
    } //end of actionPerformed

    void showerrordlg()
    {
        JOptionPane.showMessageDialog(this,"Invalid Password or User
        name","Message",JOptionPane.ERROR_MESSAGE);
    }

} //end of class

```

Designing the Registration Interface

As per the design, the YouChat application should enable new users to register with the application before logging in. The new users will have the freedom to select any username or password of their choice provided the combination of the username and the password does not match exactly with that of an existing user.

A separate screen needs to be created, which will accept the user information for registration. The registration screen will be accessible from the login screen. The login screen will have a Register button that will invoke the registration screen. The interface of the registration screen is displayed in following snapshot :



The screenshot shows a web browser window with the title "New User : Registration". Inside the window, there is a form titled "Registration Info" in a bold red font. The form contains the following fields and controls:

- Enter Username**: A text input field.
- Enter Password**: A text input field.
- Confirm Password**: A text input field.
- Student ID**: A text input field.
- First Name**: A text input field.
- Last Name**: A text input field.
- Age**: A text input field.
- Sex**: A dropdown menu with "Male" selected.
- Submit**: A button.
- Cancel**: A button.

Snapshot 2 : Registration Inteface

The registration screen will accept the following information from the user:

- Username
- Password
- Employee ID
- First name
- Last name
- Age
- Sex

The registration process needs to perform some validations on the user inputs before accepting a request for registration. These validations are:

- The Username, Password, Confirm Password, Employee ID, First Name, and Last Name fields should not be empty.
- The specified age should be between 17 and 60.

- The entries in the Password and the Confirm Password text boxes should match exactly.

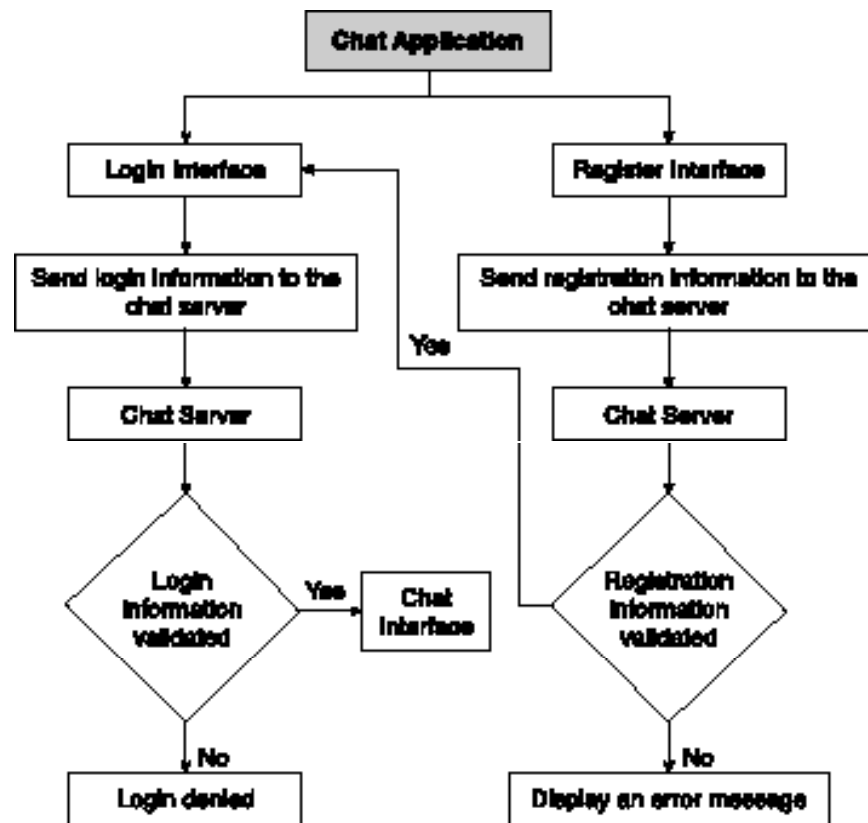


Figure 6 : Interaction between Registration interface and Chat Server

CODE FOR REGISTRATION INTERFACE :

Class Name : Register.java

```

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

import java.net.*;

public class Register extends JFrame implements ActionListener
{
    JLabel lheading;

```

```
JLabel luname;  
JLabel lpass;  
JLabel lcfpass ;  
JLabel lfname ;  
JLabel llname ;  
JLabel lage ;  
JLabel lstudentid;  
JLabel lsex;
```

```
JComboBox lstSex;  
JTextField txuname;  
JPasswordField txupass;  
JPasswordField txcfpass;  
JTextField txfname;  
JTextField txlname;  
JTextField txage;  
JTextField txstudentid;  
Font f;  
Color r;  
JButton bsubmit;  
JButton bcancel;
```

```
String uname;  
char[] upass;  
char[] cfpass;  
String fname;  
String lname;  
String age;  
String studentid;
```

```
Socket toServer;  
ObjectInputStream streamFromServer;  
PrintStream streamToServer;  
JFrame frame;
```

```
public static void main(String g[])  
{  
    new Register();  
}
```

```
public Register()  
{  
    JPanel panel = new JPanel();  
    panel.setLayout(new GridBagLayout());  
    GridBagConstraints gbCons = new GridBagConstraints();
```

```
    gbCons.gridx = 0;
```

```

gbCons.gridy = 0;
lheading = new JLabel("Registration Info");
Font f = new Font("Calibri" , Font.BOLD , 24);
lheading . setFont(f) ;
Color c = new Color(0,150,0);
lheading.setForeground(new Color(135,25,38));
//lheading.setVerticalAlignment(SwingConstants.TOP);
gbCons.anchor = GridBagConstraints.EAST;
panel.add(lheading, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 1;
luname= new JLabel("Enter Username");
gbCons.anchor = GridBagConstraints.WEST;
panel.add(luname, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 1;
txuname = new JTextField(15);
panel.add(txuname, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 2;
lpass = new JLabel("Enter Password");
panel.add(lpass, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 2;
txupass = new JPasswordField(15);
panel.add(txupass, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 3;
lcfpass = new JLabel("Confirm Password");
panel.add(lcfpass, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 3;
txcfpass = new JPasswordField(15);
panel.add(txcfpass, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 4;
lstudentid = new JLabel("Student ID");
panel.add(lstudentid, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 4;
txstudentid = new JTextField(15);
panel.add(txstudentid, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 5;
lfname = new JLabel("First Name");
panel.add(lfname, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 5;
txfname = new JTextField(15);
panel.add(txfname, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 6;
llname = new JLabel("Last Name");
panel.add(llname, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 6;
txlname = new JTextField(15);
panel.add(txlname, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 7;
lage = new JLabel("Age");
panel.add(lage, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 7;
txage = new JTextField(3);
panel.add(txage, gbCons);

```

```

gbCons.gridx = 0;
gbCons.gridy = 8;
lsex = new JLabel("Sex");
panel.add(lsex, gbCons);
gbCons.gridx = 1;
gbCons.gridy = 8;
String[] sex = {"Male", "Female"};
JComboBox listsex = new JComboBox(sex);
listsex.setSelectedIndex(0);
panel.add(listsex, gbCons);

```

```

JPanel btnPanel = new JPanel();
bsubmit = new JButton("Submit");
btnPanel.add(bsubmit);
bsubmit.addActionListener(this);
bcancel = new JButton("Cancel");
btnPanel.add(bcancel);
bcancel.addActionListener(this);

```

```

gbCons.gridx = 0;

```



```

gbCons.gridy = 9;
gbCons.anchor = GridBagConstraints.EAST;
panel.add(btnPanel, gbCons);

frame=new JFrame("New User : Registration");
frame.add(panel);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(450,350);
frame.setVisible(true);
} //end of constructor

public void actionPerformed(ActionEvent ae)
{
    JButton button = (JButton)ae.getSource();
    if(button.equals(bcancel))
    {
        frame.dispose();
    }
    else {
        // coding for submit button
        int ver =verify();
        if(ver==1){
            try
            {
                toServer = new Socket("Localhost",7777);
                streamFromServer = new ObjectInputStream(toServer.getInputStream());
                streamToServer = new PrintStream(toServer.getOutputStream());
                streamToServer.println("RegisterInfo");

                uname = txuname.getText();
                upass = txupass.getPassword();
                String pwd = new String(upass);
                streamToServer.println(uname + ":" + pwd);
                String frmServer =(String)streamFromServer.readObject(); //read
the response from the server
                if(frmServer.equals("Registered"))
                {
                    frame.dispose();
                    showUserRegistered();
                    new Login();
                }
                else if(frmServer.equals("User Exists")){
                    showUserExists();
                }
            }
        } //end of try
        catch(Exception e) {
            System.out.println(e);
        }
    }
}

```

```

        } //end of if
    } //end of else (click on submit button logic)
} //end of actionPerformed

int verify()
{
    int ctr = 0 ;
    int trimage = 0 ;
    try
    {
        uname=txuname.getText();
        upass=txupass.getPassword();
        cfpass=txcfpass.getPassword();
        fname=txfname.getText();
        lname=txlname.getText();
        age=txage.getText();
        studentid=txstudentid.getText();
        String strupass = new String(upass);
        String strcfpass = new String(cfpass);
        try{
            trimage =(int)Integer.parseInt(age.trim());
        }
        catch(Exception e){
            showErrorDlgAge();
        }
        if((uname.length()>0)&&(strupass.length()>0)&&(strcfpass.length()>0)           &&
        (fname.length()>0)&&(lname.length()>0)&&(trimage>16)
        &&(trimage<60)&&(studentid.length()>0) &&(strupass.equals(strcfpass)) )
        {
            ctr=1;
            return ctr;
        }
        else {
            showErrorDlg();
        }
    } // end of try
    catch(Exception e)
    {
        System.out.println("Exception thrown : "+ e);
    }
    return ctr;
} //end of verify

void showErrorDlg()
{
    JOptionPane.showMessageDialog(this,"                Incorrect                Entry                !",
    "Message",JOptionPane.ERROR_MESSAGE);

```

```

}

void showErrorAge()
{
JOptionPane.showMessageDialog(this,"Age incorrect!",
"Message",JOptionPane.ERROR_MESSAGE);
}

void showUserExists()
{
JOptionPane.showMessageDialog(this,"User Already exists!",
"Message",JOptionPane.ERROR_MESSAGE);
}

void showUserRegistered()
{
JOptionPane.showMessageDialog(this,"New User Registered!",
"Message",JOptionPane.INFORMATION_MESSAGE);
}

} //end of class

```

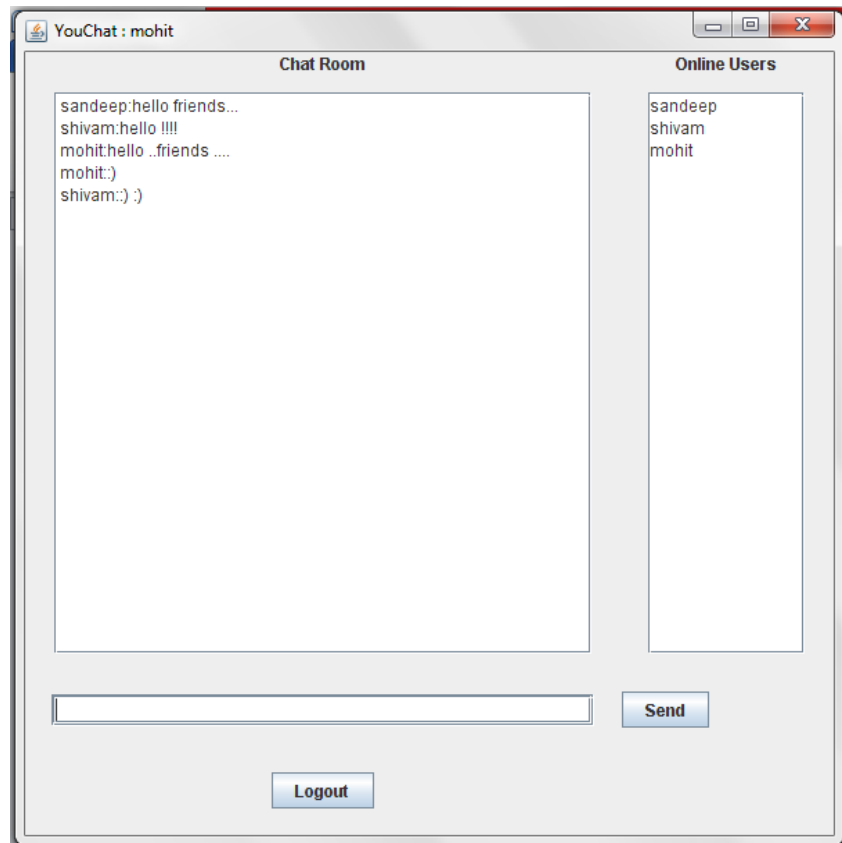
Designing the Chat Interface

After a successful login to the chat application, the user can view the chat screen. Here, the user can chat with other online users. The upper part of the chat screen will show the messages sent by the online users and also a list of online users. A text box will be provided in which the user can type a message and send it by clicking the Send button. The screen also have a Logout button that will close the chat screen. Therefore, the chat screen have the following components:

- A messages box
- A users list box
- A send message text box
- A Send button
- A Logout button

As per the design, the chat screen will perform the following functions:

- It will show a list of online users.
- It will show the messages sent by other users in a common chat room.
- When the user logs out, by clicking the Logout button or by closing the chat screen, that user will no longer be visible on the chat screen of other users.
- The chat room and online users box will be updated automatically.



Snapshot 3 : Chat Interface

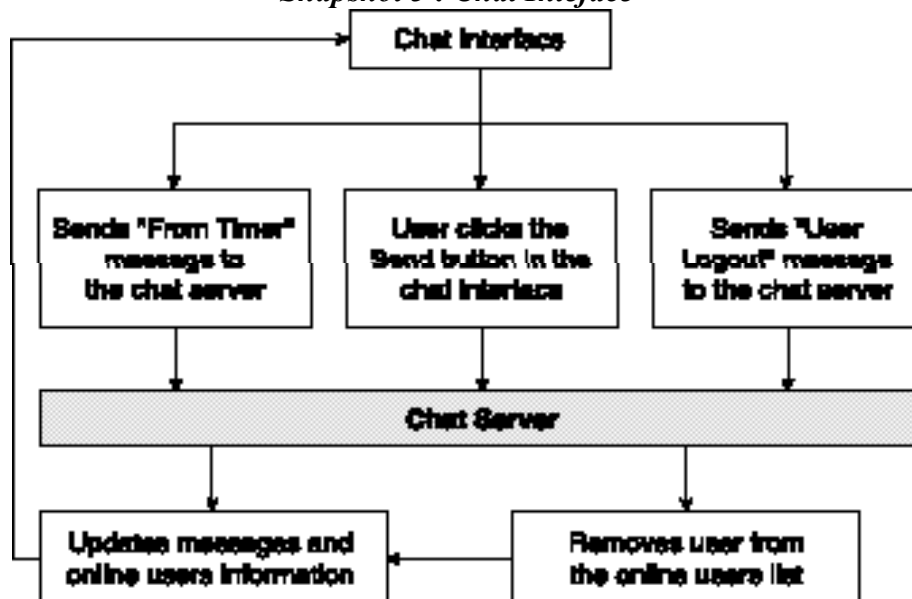


Figure 7 : Interaction between chat interface and chat server

CODE FOR CLIENT CHAT INTERFACE :

Class Name : ClientInt.java

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

public class ClientInt extends JFrame implements ActionListener
{
    Timer t = new Timer(5000,new TimerAction());

    class TimerAction implements ActionListener
    {
        Socket toServer;
        ObjectInputStream streamFromServer;
        PrintStream streamToServer;
        public void actionPerformed(ActionEvent e2)
        {
            try
            {
                toServer = new Socket("Localhost",7777) ;
                streamFromServer = new ObjectInputStream(toServer.getInputStream());
                streamToServer = new PrintStream(toServer.getOutputStream());
                message=txtMsg.getText();
                streamToServer.println("From Timer");           //send a message to
the server
                Vector vector =(Vector)streamFromServer.readObject();
                //receive vectors from the server
                Vector vector1 =(Vector)streamFromServer.readObject();

                txtListUsers.setText("");           //show the online
users
                for(int j=1;j<vector1.capacity();j++)
```

```

        {
            txtListUsers.append((String)vector1.elementAt(j));
            txtListUsers.append("\n" );
        }

        int i=messageCount ;
        for(;i<vector.capacity();i++)
        {
            txtMessages.append((String)vector.elementAt(i));
            txtMessages.append("\n");
        }
        messageCount=i;
    }
    catch(Exception e)
    {
        System.out.println("Exception Occured "+e);
    }
} // end of action performed
} //end of TimerListener class

```

```

JFrame frame;
JTextArea txtMessages;
JTextArea txtListUsers;
JTextField txtMsg;
JButton msgSendBtn;
JButton userLoginBtn;
JButton userRegisterBtn;
JButton userLogoutBtn;
JLabel lcRoom;
JLabel luList;
JScrollPane jspSendMsgPane;
JScrollPane jspTxtMsgPane;
JScrollPane jspUserListPane;
JTextField textWriteMsg;
String message;
int nsend;

int messageCount = 0 ;
String name;
PrintStream streamToServer;
ObjectInputStream streamFromServer;
Socket toServer;
String usr_name;
public String remUser;

```

```

public ClientInt(String nm)
{

```

```

remUser=nm;
usr_name=nm;
JPanel panel = new JPanel();
panel.setLayout(new GridBagLayout());
GridBagConstraints gbCons = new GridBagConstraints();
gbCons.gridx=0;
gbCons.gridy=0;
lcRoom = new JLabel("Chat Room", SwingConstants.LEFT);
panel.add(lcRoom, gbCons);

gbCons.gridx=1;
gbCons.gridy=0;
luList=new JLabel("Online Users", SwingConstants.LEFT);
panel.add(luList, gbCons);

gbCons.gridx=0;
gbCons.gridy=1;
gbCons.gridwidth=1;
gbCons.gridheight=1;
gbCons.weightx=1.0;
gbCons.weighty=1.0;
txtMessages=new JTextArea(25,35);
txtMessages.setEditable(false);
jspTxtMsgPane=new
JScrollPane(txtMessages,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORI
ZONTAL_SCROLLBAR_AS_NEEDED ) ;
panel.add(jspTxtMsgPane, gbCons);

gbCons.gridx=1;
gbCons.gridy=1;
gbCons.gridwidth=1;
gbCons.gridheight=1;
gbCons.weightx=1.0;
gbCons.weighty=1.0;
txtListUsers=new JTextArea(25,10);
txtListUsers.setEditable(false);
jspUserListPane=new
JScrollPane(txtListUsers,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZ
ONTAL_SCROLLBAR_AS_NEEDED ) ;
panel.add(jspUserListPane, gbCons);

gbCons.gridx=0;
gbCons.gridy=2;
gbCons.gridwidth=1;
gbCons.gridheight=1;
gbCons.weightx=1.0;
gbCons.weighty=1.0;

```

```

txtMsg=new JTextField(35);
jspSendMsgPane=new
JScrollPane(txtMsg,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTAL
AL_SCROLLBAR_AS_NEEDED ) ;
panel.add(jspSendMsgPane, gbCons);

gbCons.gridx=1;
gbCons.gridy=2;
gbCons.gridwidth=1;
gbCons.gridheight=1;
gbCons.weightx=1.0;
gbCons.weighty=1.0;
gbCons.anchor=GridBagConstraints.WEST;
msgSendBtn=new JButton("Send");
panel.add(msgSendBtn, gbCons);
msgSendBtn.addActionListener(this);

JPanel btnPanel = new JPanel();
userLogoutBtn = new JButton("Logout");
userLogoutBtn.addActionListener(this);

this.addWindowListener(                                     //add a listener to the window
    new WindowAdapter()
    {
        public void windowClosing(WindowEvent e1)
        {
            try

            {
                Socket toServer;
                ObjectInputStream streamFromServer;
                PrintStream streamToServer;
                toServer = new Socket("localhost",7777 ) ;
                streamToServer = new
PrintStream(toServer.getOutputStream());
                streamToServer.println("User Logout");
                streamToServer.println(remUser);
            }//end of try
            catch(Exception e2)
            {
                System.out.println("Exception Occured : "+e2);
            }
        }
    }
);

```



```

btnPanel.add(userLogoutBtn);
gbCons.gridx = 0;
gbCons.gridy = 3;
gbCons.gridwidth = 1;
gbCons.gridheight = 1;
gbCons.weightx = 1.0;
gbCons.weighty = 1.0;
gbCons.anchor = GridBagConstraints.EAST;
gbCons.fill = GridBagConstraints.HORIZONTAL;
panel.add(btnPanel, gbCons);

frame=new JFrame("YouChat : "+usr_name);
frame.add(panel);
frame.setSize(600,600);
frame.setVisible(true);
t.start();
} // end of constructor

public static void main(String g[])
    {
        String nm=new String();
        ClientInt Cl=new ClientInt(nm);
    }

public void actionPerformed(ActionEvent ae)
{
    JButton button = (JButton)ae.getSource();
    if(button.equals(userLogoutBtn)) //if logout button clicked
    {
        try
        {
            toServer = new Socket("Localhost",7777 ) ;
            streamToServer = new PrintStream(toServer.getOutputStream());
            streamToServer.println("User Logout"); //send a msg to server for
logging out
            streamToServer.println(remUser);
        }
        catch(Exception e)
        {
            System.out.println("Exception Occured: " + e ) ;
        }
        frame.dispose();
    }

    else //else if Send button is clicked
    {
        //int num1=0, num2=0 , res=0 ;

```

```

String name = "" ;
try
{
    toServer = new Socket("localhost",7777 ) ;
    streamFromServer = new ObjectInputStream(toServer.getInputStream());
    streamToServer = new PrintStream(toServer.getOutputStream());
    message = txtMsg.getText();
    String msg = message;
    streamToServer.println(usr_name+":"+msg);           //send the username
and msg typed to the server
    txtMsg.setText("");

    Vector vector = (Vector)streamFromServer.readObject();           //read the reply
from the server
    int i = messageCount;
    for(;i<vector.capacity();i++)
    {
        txtMessages.append((String)vector.elementAt(i));           //display the
messages
        txtMessages.append("\n");
    }
    messageCount=i ;
} //end of try
catch(Exception e)
{
    System.out.println("ExceptionOccurred: "+e);
}
} //end of else
} //end of actionPerformed()
} //end of class

```

The Chat Server

The server of the YouChat application will run from the command prompt and it will not be a GUI. Now, take a look at the mechanism to create the server. The Java components to be used in the server are:

- Network programming concepts
- Thread handling
- Exception handling

The following are the required functions of the chat server. It should:

- Maintain a list of online users

- Update the messages on each client chat screen
- Validate login attempts
- Register new users

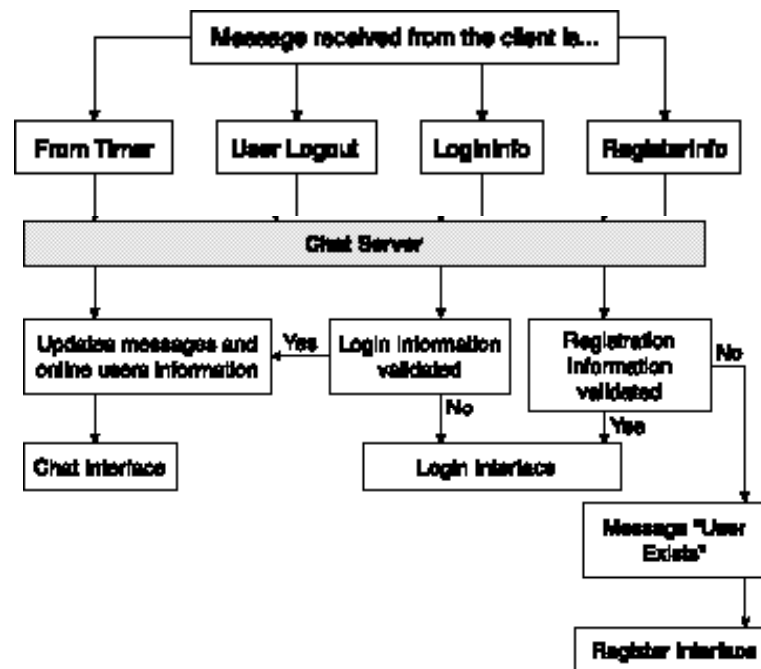


Figure 8 : Interaction between chat server and interfaces of the application
CODE FOR CHAT SERVER :

Class Name : AppServer.java

```

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

public class AppServer implements Runnable
{
    ServerSocket server;
    Socket fromClient;
    Thread serverThread;

    public static void main(String args[])
    {
        new AppServer();
    }
  
```

```

public AppServer()
{
    System.out.println("YouChat server started.....");
    try
    {
        server = new ServerSocket(7777);
        serverThread = new Thread(this);
        serverThread.start();
    }
    catch(Exception e)
    {
        System.out.println("Sorry ! Cannot start the thread: "+ e);
    }
} //end of AppServer

public void run()
{
    try {
        while(true){
            fromClient = server.accept();           //Listening to the clients request
            System.out.println(fromClient);
            Connect con = new Connect(fromClient);    //Creating the connect object(is class
me jo code hoga wo clientInt interface ko maintain karega)

        }
    }
    catch(Exception e){
        System.out.println("Sorry ! Cannot listen to the client"+ e);
    }
} //end of run
} //end of AppServer

```

Class Name : Connect.java

```

import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.String.*;
class Connect
{
    ObjectOutputStream streamToClient;

```

```
BufferedReader streamFromClient;
int ctr = 0;
```

```
static Vector vector;
static Vector vctrList;
String message = " ";
static String str = new String("UsrList");
static
{
vector = new Vector(1,1);
vctrList = new Vector(1,1);
vctrList.addElement((String)str);
}
```

```
public Connect(Socket inFromClient)
{
String msg = "" ;           //Retrieving the clients stream
String mseg = "" ;
try{
    streamFromClient      =      new      BufferedReader(new
InputStreamReader(inFromClient.getInputStream()));
    streamToClient = new ObjectOutputStream(inFromClient.getOutputStream());
    msg = streamFromClient.readLine();
    if((msg.equals("From Timer")))
    {
        streamToClient.writeObject(vector);
        streamToClient.writeObject(vctrList);
    }
    else if(msg.equals("LoginInfo"))
    {
        msg=streamFromClient.readLine();
        System.out.println(msg);
        int ver = verify(msg);
        if(ver==1)
        {
            String colon = new String(":");
            int index =((String)msg).lastIndexOf(colon);
            String userName =(String)msg.substring(0,index);
            if(!(vctrList.indexOf((String)userName)>0))
            {
                streamToClient.writeObject("Welcome");
                vctrList.addElement((String)userName);
            }
        }
    }
    else {
        streamToClient.writeObject("Login denied");
    }
}
```

```

        }
    }
    else if(msg.equals("RegisterInfo"))
    {
        msg = streamFromClient.readLine();
        System.out.println(msg);
        int ret = checkFile(msg);
        if (ret==0 )
            streamToClient.writeObject("User Exists");
        if(ret==1)
        {
            FileOutputStream out = new
FileOutputStream("UserPass.txt",true);
            PrintStream p = new PrintStream(out);
            p.println(msg);
            p.close();
            streamToClient.writeObject("Registered");
        }
    }

    else if(msg.equals("User Logout"))
    {
        String remUser = streamFromClient.readLine();
        boolean b = vctrList.removeElement((String)remUser);
    }
    else
    {
        message=message+msg;
        vector.addElement((String)message);
        streamToClient.writeObject(vector);
    }
} // end of try

catch(Exception e)    {System.out.println("Exception Occured : "+e);
    }

finally
{
    try{
        inFromClient.close();
    }
    catch(IOException e)
    {
        System.out.print("Exception Occurred: "+e);
    }
}

} //end of constructor

int verify(String mesg)

```

```

{
try
    {
        RandomAccessFile RAS = new RandomAccessFile("UserPass.txt", "r");
        String str = "";
        while((RAS.getFilePointer()!=(RAS.length()))
        {
            str=RAS.readLine();
            if(str.equals(mesg))
            {
                ctr=1;
                break;
            }
        }
        RAS.close();
    }
catch(Exception e)    {
        System.out.print("Exception Occurred: "+e);
    }
return ctr;
} //end of verify()

```

```

int checkFile(String mesg)
{
    int chk = 1 ;
    try
    {
        RandomAccessFile RS = new RandomAccessFile("UserPass.txt", "r");
        int i = 0 ;
        String str = "";
        String colon = new String(":");
        int index = ((String)mesg).lastIndexOf(colon);
        //System.out.println(index);
        String userName = (String)mesg.substring(0,index);

        while((RS.getFilePointer()!=(int)(RS.length()))
        {
            str = RS.readLine();
            int index1 = ((String)str).lastIndexOf(colon);
            //System.out.println(index1);
            String usrName = (String)str.substring(0,index1);
            //System.out.println(usrName);
            if(usrName.equals(userName))
            {
                chk = 0;
                break;
            }
        }
    }
}

```

```

        }//end of while
        RS.close() ;

    }//end of try
    catch(Exception e)
    {
        System.out.print("Exception Occurred : "+e);
    }
    return chk;
} //end of chkFile

}

```

The Testing Phase

Software modules are tested for their functionality as per the requirements identified during the requirements analysis phase . To test the functionality of youChat ,a Quality Assurance (QA) team will form . The requirements identified during the requirements analysis phase were submitted to the QA team. The QA team teste youChat for these requirements .

Testing Methodology

- Unit Testing
- Integration Testing

Unit Testing:

Unit testing is a procedure used to validate that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

Ideally, each test case is independent from the others; mock or fake objects as well as test harnesses can be used to assist testing a module in isolation. Unit testing is typically

done by software developers to ensure that the code they have written meets software requirements and behaves as the developer intended.

Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.

Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

Integration Testing

It is sometimes called I&T i.e. Integration and testing, it is the phase of software testing in which individual software modules are combined and tested as a group. It follows unit testing and precedes system testing.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test that all components within assemblages interact correctly

The Acceptance Phase

In this phase, based on the pre-defined acceptance criteria, the marketing team conducts acceptance testing for the client projects. Acceptance was obtained from the Quality Assurance team. After the project will develop and the people will start using youchat, constant support will be provided by youTeam in terms of installation and debugging errors, if any.

SUMMARY

What we've learned

This project has gone into great detail about the ideas used in the construction of a multithreaded, connection-oriented server in the Java language. You've learned how to deal with multiple clients at once, and you've come to know the seven basic elements that make up such a server.

REFERENCES

These books provide valuable information on Java programming:

- *"Java Network Programming, 2nd Edition"* by Elliotte Rusty Harold
- *"Java Network Programming, 2nd Edition"* by Merlin Hughes, Michael Shoffner, and
Derek Hamner (different book with the same title)
- *Java the Complete reference*
- *Java : Head first java*
- *Javadoc(Documentation provided along with jdk)*

- *O'reilly java Swing*

Other tutorials related to this topic are:

- Introduction to socket Programming
- Java AWT, Swing
- Java input/output
- Mutithreading
- Networking Concepts