UNIVERSITY OF ENGINEERING & MANAGEMENT
Good Education, Good Jobs

UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CSE)

SEM/YEAR: 6th Semester / 3rd Year

SECTION: CSE-3C

SUBJECT NAME: Computer Networks Lab

SUBJECT CODE: CS692

PROJECT: CHAT SYSTEM USING SOCKET PROGRAMMING

STUDENT DETAILS:

Name	Roll	Enrollment no.	Reg. no.
ARPAN DUTTA	21	12017009001313	304201700900161

CERTIFICATE

Certified that this project report **"CHAT SYSTEM USING SOCKET PROGRAMMING"** is the bonafide work of Arpan Dutta (Enrollment no. - 3312017009001313) of B.Tech, CSE, who carried out the project work under our supervision.

………………………………..

………………………………..

SIGNATURE

Examiner:

ACKNOWLEDGEMENT

The completion of this project could not have been accomplished without the support of our teachers and guide Prof. Sumit Anand, Prof. Nilanjan Byabarta & Prof. Arunabha Tarafdar. We are thankful to you for allowing us your time to research and write.

We are also very thankful to our respected teachers for their co-operation and suggestion regarding the project work.

Last but not the least we are very thankful to our HOD, Prof. Sukalyan Goswami for giving us an opportunity of doing such an interesting project work.

- **Arpan Dutta**

PROJECT REPORT

INTRODUCTION:

Online chat may refer to any kind of communication over the Internet that offers a real-time transmission of text messages from sender to receiver. Chat messages are generally short in order to enable other participants to respond quickly. Thereby, a feeling similar to a spoken conversation is created, which distinguishes chatting from other text-based online communication forms such as Internet forums and email. Online chat may address point-to-point communications as well as multicast communications from one sender to many receivers and voice and video chat, or may be a feature of a web conferencing service.

Types Of Communication:

Simplex Mode

In Simplex mode, the communication is unidirectional, as on a one-way street. Only one of the two devices on a link can transmit, the other can only receive. The simplex mode can use the entire capacity of the channel to send data in one direction.

Example: Keyboard and traditional monitors. The keyboard can only introduce input, the monitor can only give the output.

Half-Duplex Mode

In half-duplex mode, each station can both transmit and receive, but not at the same time. When one device is sending, the other can only receive, and vice versa. The half-duplex mode is used in cases where there is no need for communication in both direction at the same time. The entire capacity of the channel can be utilized for each direction.

Example: Walkie- talkie in which message is sent one at a time and messages are sent in both the directions.

Full-Duplex Mode

In full-duplex mode, both stations can transmit and receive simultaneously. In full_duplex mode, signals going in one direction share the capacity of the link with signals going in other direction, this sharing can occur in two ways:

Either the link must contain two physically separate transmission paths, one for sending and other for receiving.

Or the capacity is divided between signals travelling in both directions.

Full-duplex mode is used when communication in both direction is required all the time. The capacity of the channel, however must be divided between the two directions.

Example: Telephone Network in which there is communication between two persons by a telephone line, through which both can talk and listen at the same time.

We use a full duplex mode in our chat system model.

PROPOSED METHOD/SOLUTION APPROACH:

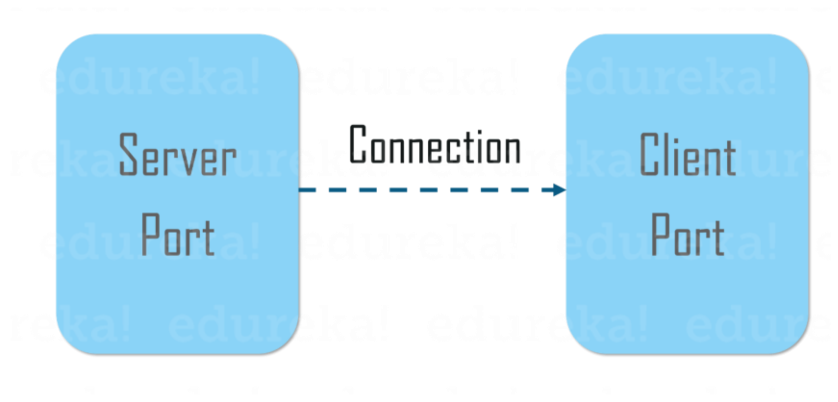
Socket Programming:

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other in order to form a connection.

The server forms the listener socket while the client reaches out to the server. Socket and Server Socket classes are used for connection-oriented socket programming.

Socket:

A socket in Java is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.



An endpoint is a combination of an IP address and a port number. The package in the Java platform provides a class, Socket that implements one side of a two-way connection between your Java program and another program on the network. The class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Now that you know, what is Socket in Java, let’s move further and understand how does client communicates with the server and how the server responds back.

Server Side Programming:

Basically, the server will instantiate its object and wait for the client request. Once the client sends the request, the server will communicate back with the response.

In order to code the server-side application, you need two sockets and they are as follows:

A ServerSocket which waits for the client requests (when a client makes a new Socket())

A plain old socket for communication with the client.

After this, you need to communicate with the client with the response.

Client Side Programming:

In the case of client-side programming, the client will first wait for the server to start. Once the server is up and running, it will send the requests to the server. After that, the client will wait for the response from the server. So, this is the whole logic of client and server communication. Now let’s understand the client side and server side programming in detail.

In order to initiate a clients request, you need to follow the below-mentioned steps:

1. Establish a Connection

The very first step is to establish a socket connection. A socket connection implies that the two machines have information about each other’s network location (IP Address) and TCP port.

You can create a Socket with the help of a below statement:

Socket socket = new Socket(“127.0.0.1”, 5000)

Here, the first argument represents the IP address of Server.

The second argument represents the TCP Port. (It is a number that represents which application should run on a server.)

2. Communication

In order to communicate over a socket connection, streams are used for both input and output the data. After establishing a connection and sending the requests, you need to close the connection.

3. Closing the connection

The socket connection is closed explicitly once the message to the server is sent.

Now, let’s implement server-side programming and then arrive at the output.

RESULTS & DISCUSSION:

After successful completion of this project we can conclude that we can use the concept of socket programming to create a full duplex chat system.

Here we used the Socket class in java to implement the socket programming and thus creating a chat system.

SOURCE CODE

Client.java

```
//CLIENT SIDE
import java.net.*;
import java.io.*;
import java.util.*;

public class Client{
    private Socket socket=null;
    private Scanner input=null;
    private PrintStream out=null;
    private ServerConnection sc=null;
    private String userName=null;
public Client(String address,int port,String userName){
    try{
        this.userName=userName;
        socket=new Socket(address,port);
        System.out.println("*** Connected ***");
        input=new Scanner(System.in);
        out=new PrintStream(socket.getOutputStream());
        out.println(userName);
        System.out.println("\nHello! Welcome to the chatroom.");
        System.out.println("Instructions:");
        System.out.println("1. Simply type the message to send broadcast t
o all active clients");
        System.out.println("2. Type '@username<space>yourmessage' without
quotes to send message to desired client");
        System.out.println("3. Type 'WHOISIN' without quotes to see list o
f active clients");
        System.out.println("4. Type 'OVER' without quotes to logoff from s
erver");
```

```
sc=new ServerConnection(this);
sc.start();
//sis=new ServerIsAlive(socket);
//sis.start();
String line;
while(true){
    System.out.print("> ");
    line=input.nextLine();

    if(line.equals("")){
        if(socket.isClosed()){
            break;
        }
        continue;
    }
    out.println(line);

    if(line.equalsIgnoreCase("Over") || socket.isClosed() ){
        break;
    }
}
try{
    input.close();
    //System.out.println("input closed");
    out.close();
    //System.out.println("out closed");
    socket.close();
    //System.out.println("socket closed");
    sc.getServerOutput().close();
    //System.out.println("serveroutput closed");
}
catch(IOException e){
```

```
            System.out.println(e);
        }
    }
    catch(UnknownHostException e){
        System.out.println(e);
    }
    catch(IOException e){
        System.out.println(e);
    }
}
public Socket getSocket(){
    return socket;
}
public Scanner getInput(){
    return input;
}
public PrintStream getOut(){
    return out;
}
public static int count=0;
synchronized public static void main(String args[]){
    Scanner scan=new Scanner(System.in);
    String s;
    int port=3000;
    String ip="127.0.0.1";

    try{
        if(args.length==0){
            System.out.println("enter a username: ");
            s= scan.nextLine();
            Client client=new Client(ip,port,s);
        }
        else if(args.length==1){
            Client client=new Client(ip,port,args[0]);


        }
        else if(args.length==2){
            Client client=new Client(ip,Integer.parseInt(args[1]),args[0])
;
        }
        else if(args.length==3){
            Client client=new Client(args[2],Integer.parseInt(args[1]),arg
s[0]);


        }
        else{
            System.out.println("client side initialization format: java cl
ient<space>username<space>port no.<space>ip");
            System.out.println("default client side initialization:  usern
ame= *given username*, port="+port+", ip="+ip);
        }
    }
    catch(Exception e){
        System.out.println("client side initialization format: java client
<space>username<space>port no.<space>ip");
        System.out.println("default client side initialization:  username=
 *given username*, port="+port+", ip="+ip);


    }
    scan.close();
}
}
```

ServerConnection.java

```
//CLIENT SIDE

import java.io.*;

import java.util.*;


public class ServerConnection extends Thread {

    private Client client;

    private Scanner serveroutput = null;

    private String line = null;

    public ServerConnection(Client client) {

        this.client=client;

        try {

            serveroutput = new
Scanner(client.getSocket().getInputStream());

        } catch (IOException e) {

            System.out.println(e);

        }

    }

    public Scanner getServerOutput(){

        return serveroutput;

    }

    public void run() {


        while (true) {

            try { line = serveroutput.nextLine();    } catch (Exception e)
{}

            if(line==null){

                break;

            }

            else if(line.equals("*** server closed ***") ||
line.equals("*** this username is already taken ***")){

                try {
                    System.out.println(line+"\npress 'enter key' to
exit");
                    client.getSocket().close();
                    client.getInput().close();
                    client.getOut().close();
                }
                catch (IOException e) {
                    System.out.println(e);
                }
                break;
            }
            else{
                System.out.println(line);
                System.out.print("> ");
            }
            line=null;
        }
            serveroutput.close();
    }
}
```

Server.java

```
//SERVER SIDE
import java.net.*;
//import java.io.*;
import java.util.*;
//import java.util.concurrent.*;
import java.text.SimpleDateFormat;


public class Server{
    private Socket socket=null;
    private ServerSocket server=null;
    private ArrayList<ClientHandler> clientList=new
ArrayList<ClientHandler>();
    private int id=0;
    //private ExecutorService pool = Executors.newFixedThreadPool(2);
    private ServerClose sc;
    private Date date;
    private SimpleDateFormat sdf;
    public Server(int port){
        try{
            sdf = new SimpleDateFormat("HH:mm:ss");
            server=new ServerSocket(port);
            date=new Date();
            System.out.println(sdf.format(date)+" Server started");
            sc=new ServerClose(this);
            sc.start();
            while(true){
                date=new Date();
                System.out.println(sdf.format(date)+" Server waiting for a
clients on port "+port);
                socket=server.accept();
                if(server.isClosed()){
                    break;
                }
                ClientHandler clientThread=new
ClientHandler(socket,id,clientList);
                clientList.add(clientThread);
                clientThread.start();
                //pool.execute(clientThread);
                id=id+1;
            }
        }
        catch(Exception e){}
    }
    public ArrayList<ClientHandler> getArrayList(){
        return clientList;
    }
    public static void main(String args[]){
        int port=3000;
        try{
            if(args.length==0){
                Server server=new Server(port);
            }
            else if(args.length==1){
                Server server=new Server(Integer.parseInt(args[0]));
            }
            else{
                System.out.println("server side initialization format:
java server<space>port no.");
                System.out.println("default server side initialization:
port="+port);
            }
        }
        catch(Exception e){
            System.out.println("server side initialization format: java
server<space>port no.");
            System.out.println("default server side initialization:
port="+port);
        }
    }
}
```

ClientHandler.java

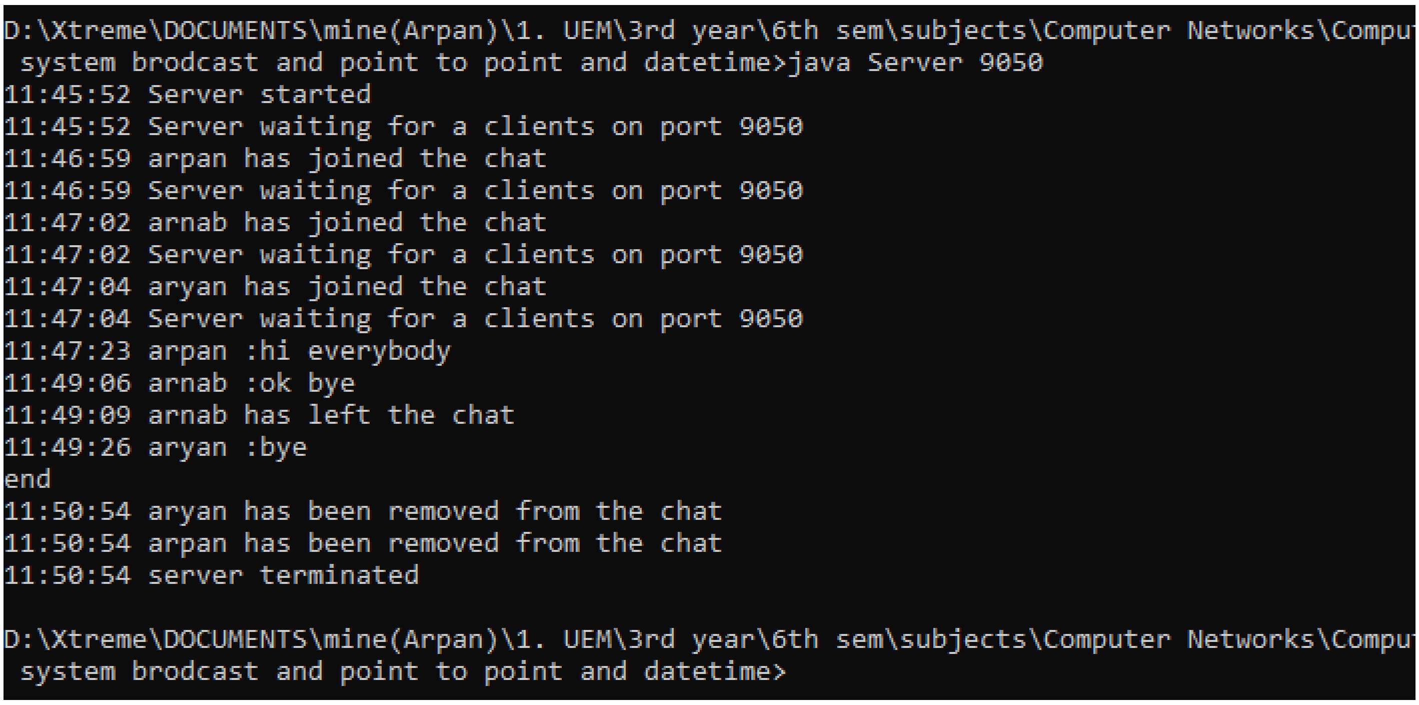
```
//SERVER SIDE
import java.io.IOException;
import java.io.PrintStream;
import java.net.*;
import java.util.ArrayList;
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.text.SimpleDateFormat;
import java.util.Date;


public class ClientHandler extends Thread {
    private Socket socket;
    private int id;
    private Scanner in = null;
    private PrintStream output = null;
    private ArrayList<ClientHandler> clientList;
    private String userName;
    private int flag=0;
    private Date date;
    private SimpleDateFormat sdf;


    public ClientHandler(Socket socket, int id,ArrayList<ClientHandler>
clientList) {
        sdf = new SimpleDateFormat("HH:mm:ss");
        this.socket = socket;
        this.id = id;
        this.clientList=clientList;
        try{
            in = new Scanner(socket.getInputStream());
            userName=in.nextLine();
            output = new PrintStream(socket.getOutputStream());

            int n=clientList.size();

            //System.out.println("n= "+n);

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

                //System.out.println("i= "+i);

                //System.out.println("this.userName= "+this.userName);


//System.out.println("clientList.get(i).userName="+clientList.get(i).userN
ame);

                if(this.userName.equals(clientList.get(i).userName)){

                    date=new Date();

                    output.println(" *** this username is already taken
***");

                    date=new Date();

                    System.out.println(sdf.format(date)+" user with same
username trying to enter");

                    flag=1;

                    break;

                }

            }

            if(flag==0){

                date=new Date();

                System.out.println(sdf.format(date)+" "+userName+" has
joined the chat");

            }

        }

        catch(IOException e){

            System.out.println(e);

        }

    }

    public String getUserName(){

        return userName;

    }

    public Scanner getIn() {
        return in;
    }
    public PrintStream getOutput() {
        return output;
    }
    public Socket getSocket(){
        return socket;
    }
    public void run() {

        if(flag==1){

            try {
                in.close();
                output.close();
                socket.close();
            }
            catch (IOException e){
                System.out.println(e);
            }
            int n=clientList.size();
            clientList.remove(n-1);
        }
        else{
            String line=null;

            while(true){
                try{
                line=in.nextLine();
                }
                catch(NoSuchElementException e){}
                //checks if its socket is closed by ServerClose thread
                //if yes, it breaks out of loop and thus terminates itself
                if(socket.isClosed()){
                    break;
                }
                else if(line.equalsIgnoreCase("Over")){
                    try {
                        socket.close();
                        in.close();
                        output.close();
                    }
                    catch (IOException e) {

                        e.printStackTrace();
                    }
                    for(int i=0;i<clientList.size();i++){
                        if(this.id==clientList.get(i).id ){
                            clientList.remove(i);
                            break;
                        }
                    }
                    date=new Date();
                    System.out.println(sdf.format(date)+" "+userName+" has
left the chat");
                    break;
                }
                else{
                    if(line.equalsIgnoreCase("whoisin")){
                        String list="CLIENT LIST:";
                        for(int i=0;i<clientList.size();i++){
                            list=list+"\n"+(i+1)+".
"+(clientList.get(i).getUserName());
                        }
                        date=new Date();
                        output.println(sdf.format(date)+" "+list);
                    }
                    //deal with private message
                    else if(line.charAt(0)=='@'){
                        String username=line.split(" ",2)[0];
                        String 
privateUserName=username.substring(1,username.length());
                        //check if msg is to self
                        if(this.userName.equals(privateUserName)){
                            date=new Date();
                            output.println(sdf.format(date)+" you can't 
send msg to yourself");
                        }
                        else{

                            //check if the requested user exist
                            int flag=0;

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

if(clientList.get(i).userName.equals(privateUserName)){
                                    flag=1;
                                    break;
                                }
                            }
                            //server msg if requested user doesn't exist
                            if(flag==0){
                                date=new Date();
                                output.println(sdf.format(date)+" the 
requested user isn't in the chat");
                            }
                            //server msg if requested user does exist
                            else{
                                String msg=line.split(" ",2)[1];

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

    if(privateUserName.equals(clientList.get(i).userName)){

                                        PrintStream output;
                                        try {
                                            output = new
    PrintStream(clientList.get(i).socket.getOutputStream());
                                            date=new Date();

    output.println(sdf.format(date)+" "+userName+" :"+msg);
                                            //output.close();
                                        } catch (IOException e) {

                                            e.printStackTrace();
                                        }
                                        break;

                                    }
                                }
                            }
                        }
                    }
                    else{
                        date=new Date();
                        System.out.println(sdf.format(date)+" "+userName+"
    :"+line);
                        date=new Date();
                        output.println(sdf.format(date)+" "+"server says:
    "+line);
                        for(int i=0;i<clientList.size();i++){
                            if(this.id!=clientList.get(i).id){


                                PrintStream output;

                                try {

                                    output = new
PrintStream(clientList.get(i).socket.getOutputStream());

                                    date=new Date();

                                    output.println(sdf.format(date)+"
"+userName+" :"+line);

                                } catch (IOException e) {

                                    e.printStackTrace();

                                }

                            }

                        }

                    }

                }


                line=null;

            }

        }

    }

}
```

ServerClose.java

```
//SERVER SIDE
import java.io.PrintStream;
import java.util.Scanner;
import java.util.Date;
import java.text.SimpleDateFormat;

public class ServerClose extends Thread {
    private Scanner in;
    private Server server;
    private PrintStream out;
    //private Date date;
    private SimpleDateFormat sdf;

    public ServerClose(Server server) {
        sdf = new SimpleDateFormat("HH:mm:ss");
        this.server = server;
        in = new Scanner(System.in);
    }

    public void run()  {

        try {
            while(true){
                String serverCommand=in.next();
                if (serverCommand.equalsIgnoreCase("end")) {
                    in.close();
                    int n=server.getArrayList().size();
                    for(int i=n-1;i>=0;i--){
                        out = new
PrintStream(server.getArrayList().get(i).getSocket().getOutputStream());
                        out.println("*** server closed ***");
                        out.close();
                        //date=new Date();
                        System.out.println(sdf.format(new Date())+"
"+server.getArrayList().get(i).getUserName()+" has been removed from the
chat");
                        server.getArrayList().get(i).getSocket().close();
                        server.getArrayList().get(i).getIn().close();
                        server.getArrayList().get(i).getOutput().close();
                        server.getArrayList().remove(i);
                    }
                    break;

                }
            }

        }
        catch(Exception e){
            //System.out.println(e);
        }
        finally{
            System.out.println(sdf.format(new Date())+" server
terminated");
            try {
                server.getServerSocket().close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

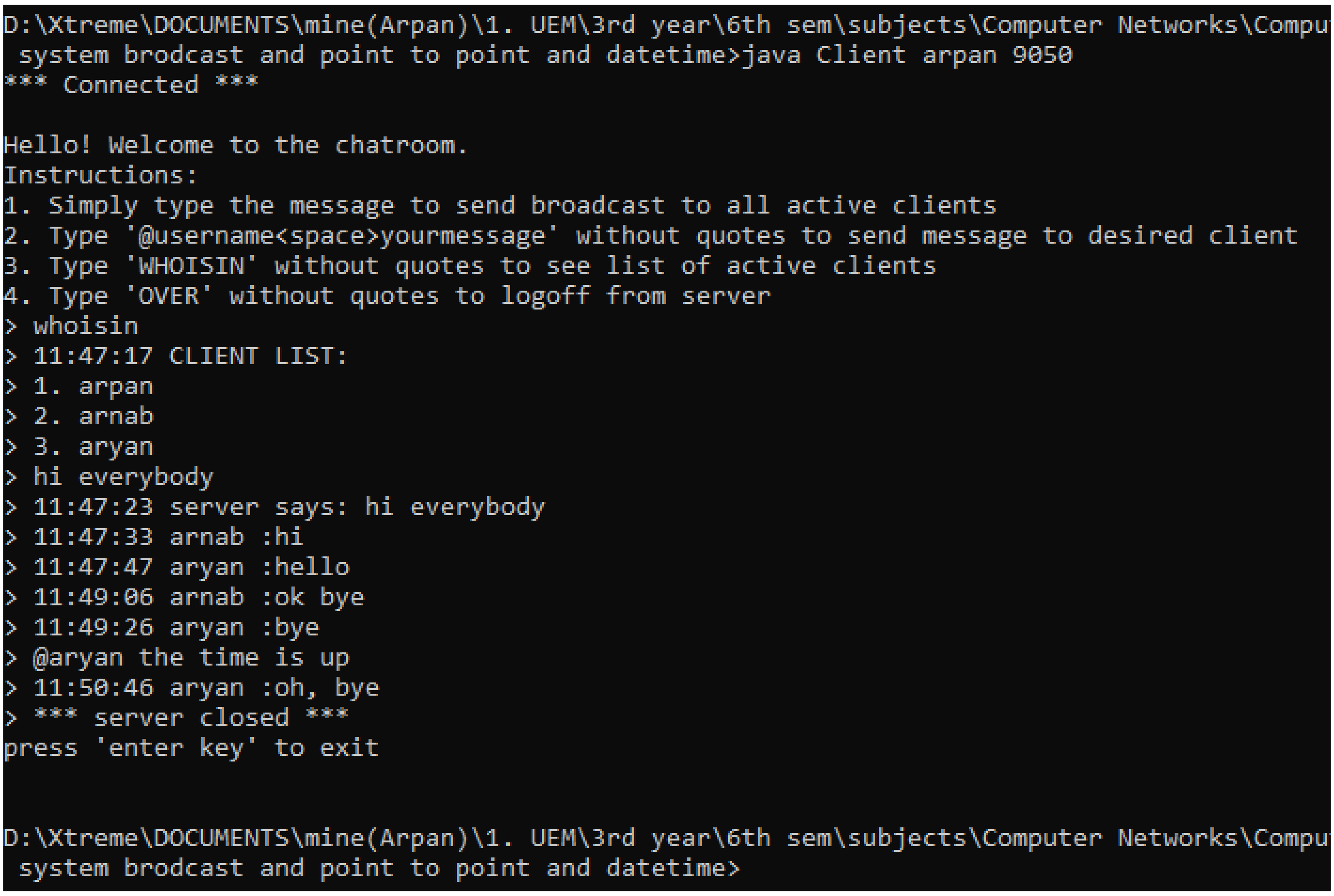
SCREENSHOTS OF OUTPUT

Server:



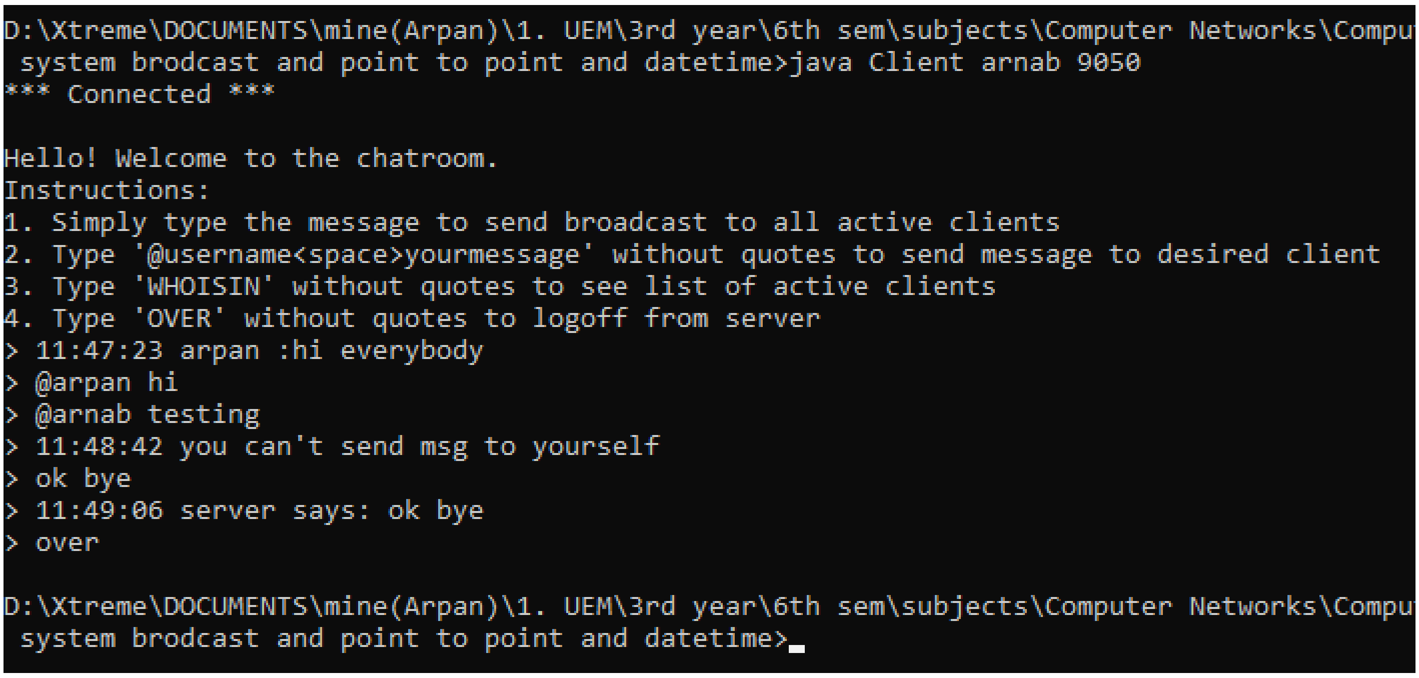
```
D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>java Server 9050
11:45:52 Server started
11:45:52 Server waiting for a clients on port 9050
11:46:59 arpan has joined the chat
11:46:59 Server waiting for a clients on port 9050
11:47:02 arnab has joined the chat
11:47:02 Server waiting for a clients on port 9050
11:47:04 aryan has joined the chat
11:47:04 Server waiting for a clients on port 9050
11:47:23 arpan :hi everybody
11:49:06 arnab :ok bye
11:49:09 arnab has left the chat
11:49:26 aryan :bye
end
11:50:54 aryan has been removed from the chat
11:50:54 arpan has been removed from the chat
11:50:54 server terminated

D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>
```

Client 1:

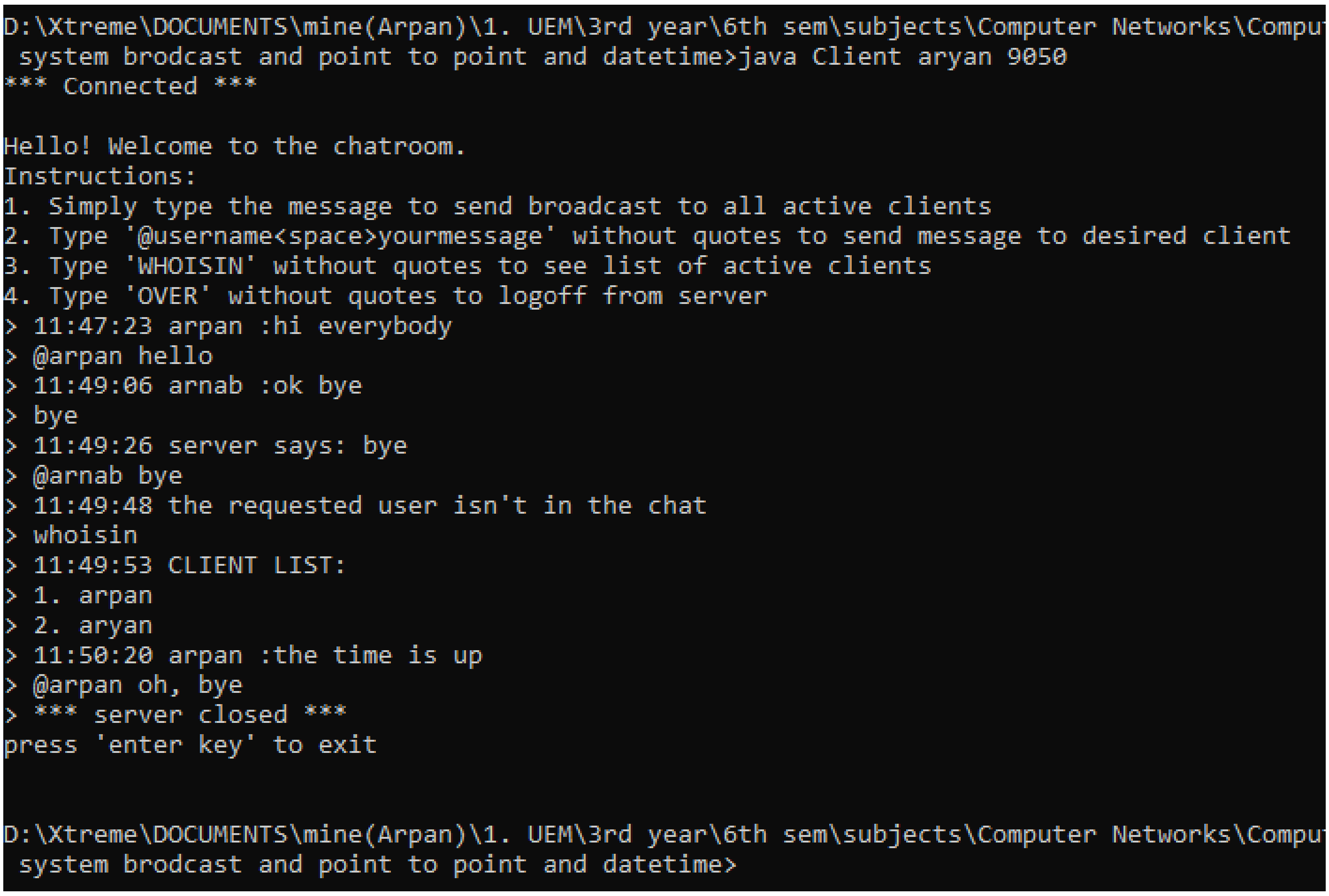


```
D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>java Client arpan 9050
*** Connected ***

Hello! Welcome to the chatroom.
Instructions:
1. Simply type the message to send broadcast to all active clients
2. Type '@username<space>yourmessage' without quotes to send message to desired client
3. Type 'WHOISIN' without quotes to see list of active clients
4. Type 'OVER' without quotes to logoff from server
> whoisin
> 11:47:17 CLIENT LIST:
> 1. arpan
> 2. arnab
> 3. aryan
> hi everybody
> 11:47:23 server says: hi everybody
> 11:47:33 arnab :hi
> 11:47:47 aryan :hello
> 11:49:06 arnab :ok bye
> 11:49:26 aryan :bye
> @aryan the time is up
> 11:50:46 aryan :oh, bye
> *** server closed ***
press 'enter key' to exit


D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>
```

Client 2:


```
D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>java Client arnab 9050
*** Connected ***

Hello! Welcome to the chatroom.
Instructions:
1. Simply type the message to send broadcast to all active clients
2. Type '@username<space>yourmessage' without quotes to send message to desired client
3. Type 'WHOISIN' without quotes to see list of active clients
4. Type 'OVER' without quotes to logoff from server
> 11:47:23 arpan :hi everybody
> @arpan hi
> @arnab testing
> 11:48:42 you can't send msg to yourself
> ok bye
> 11:49:06 server says: ok bye
> over

D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>_
```

Client 3:


```
D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>java Client aryan 9050
*** Connected ***

Hello! Welcome to the chatroom.
Instructions:
1. Simply type the message to send broadcast to all active clients
2. Type '@username<space>yourmessage' without quotes to send message to desired client
3. Type 'WHOISIN' without quotes to see list of active clients
4. Type 'OVER' without quotes to logoff from server
> 11:47:23 arpan :hi everybody
> @arpan hello
> 11:49:06 arnab :ok bye
> bye
> 11:49:26 server says: bye
> @arnab bye
> 11:49:48 the requested user isn't in the chat
> whoisin
> 11:49:53 CLIENT LIST:
> 1. arpan
> 2. aryan
> 11:50:20 arpan :the time is up
> @arpan oh, bye
> *** server closed ***
press 'enter key' to exit


D:\Xtreme\DOCUMENTS\mine(Arpan)\1. UEM\3rd year\6th sem\subjects\Computer Networks\Compu
 system brodcast and point to point and datetime>
```