**Chapter 1**

**INTRODUCTION**

* 1. **About the project**

Chat server is one of the most widely used mode of communication among the younger generation of the society. On inclusion of certain required functions and codes, Chat server can be used even for the transmission of images.

**1.2 Objective of project**

The objective of the project is to provide an environment to the students that let them chat on an autonomous network systems consisting of Routers, hosts, wireless or LAN network system. The project is a collection of two distinct but related modules named:

* SERVER
* CLIENT

**1.2.1 SERVER**

A server computer is a computer dedicated to running a server application. A server application is a computer program that accepts computer network connections in order to service requests by sending back responses. Examples of server applications include Mail transfer agent, Fileserver, and Proxy server.Server is also a designation for computer models intended for use in running server applications, often under heavy workloads, unattended, for an extended period of time. While any workstation computer can run server operating systems and server applications, a server computer usually has special features intended to make it more suitable. These features can include a faster central processing unit, faster and more plentiful RAM, and larger hard disk drive, but these traits are shared with high-end desktop computer

**1.2.2 CLIENT**

A client is an Application software or system that accesses a remote service on another Computer system, known as a Server computing, by way of a Network. The term was firstapplied to Peripheral device that were not capable of running their own stand-alone Computer program, but could interact with remote computers via a network. These Dumb terminals were clients of the Time sharing Mainframe computer.

**1.3 CHARACTERISTICS OF A SERVER**

* Never initiates requests or activities
* Listens to network and responds only to requests from connected, authorized clients.
* Waits for and replies to requests from connected clients
* A server can remotely install/uninstall applications and transfer data to clients

**1.4FUTURE SCOPE**

In future, the project can be enhanced by making it more secure by providing password protection. Presently the project runs on the local network i.e. LAN or local wireless network but later we can make this run on internet by providing public IP address and making it run global. We can handle user data separately proving them password access. We can also improvise it for   private conversation with a user selected user. We can also add various emotions and smileys. Most important addition about which we our thinking is telecommunication through chatting software as presently available in g-talk and other chatting software.

* 1. **Need for the project**

Developing this application can help students to understand basics about socket programming in java programming in java programming language and how communication is established between end to end through TCP IP and other protocols.

**1.6 Advantage of java**

Java program can reach out & touch a program on another machine.

All low level networking details are taken care of by classes in the java.net package.

Big benefit of java is sending & receiving data over a network is just I/O.

**Chapter 2**

**SYSTEM ANALYSIS**

**2.1 PROPOSED SYSTEM**

The project is to develop an application which will help in communication between multiple users in real time. The system will allow users to chat with other online users in common room & privately. You are free to chat privately with one or more people at the same time, exchange text messages simultaneously in the same “session” over the network .User can broadcast the same message to many users at the same time. To provide text based communication.

To allow numerous people to exchange text messages simultaneously in the same “session” over the network in real time.Ability to broad cast the same message to many users at the same time.

**2.2 FEASIBLITY STUDY**

Feasibility is the determination of whether or not a project is worth doing. The process followed in making this determination is called the FEASIBILITY STUDY.Feasibility study is the test of system, according to its workability, impact on the organization, ability tomeet the user needs, and effective use of resources. A feasibility study is conducted to select the best system that meets performance and requirements. Thus this study tells whether the resources which are used would be able to fulfill the requirements of the project, whether the approach used to make the system is suitable enough, whether the prescribed budget is sufficient to meet the needs and whether the system would be able to meet the user’s requirements.

**To perform feasibility study basically following question should arises:**

* Is there any new and better way to do the job that will benefit the user?
* What are the cost and savings of the alternative?
* Is the prescribed method is flexible or not?

There are various considerations, which must be taken in account while performing feasibility study. These are

**2.2.1 Technical**

This is concerned with specifying equipment and soft wares that will successfully satisfy the user’s requirement. The technical need of the system may include

* The facility to produce output in a given time.
* Response time under certain conditions.
* Ability to process a certain volume of transaction at a particular speed.

In this feasibility configuration of the system is given minor importance. This configuration should give the complete picture system requirement.

**2.2.2 Behavioral**

It determines how much effort and time will go into educating, selling and training the user staff on a candidate system.

**2.2.3 Economical**

A system developed technically and that will be used if installed must be still profitable for theorganization .Financial benefits must equal or exceed the cost. In this feasibility following things are calculated.

* The cost of hardware and software.
* The cost of checking costly errors.

**2.3 Feasibility Study for the Project**

The Project “CHAT SYSTEM”satisfies all the feasibility criteria. The software used are open source software easily available in market. The hardware required to develop this system is a desktop computer with descent computing power.In case of financial feasibility, it is quite economical, only hardware circuit requires spending small money.In case of administrative feasibility no special training is required for the users as the project is very user-friendly, only working knowledge of computer is required.

**2.4 The client-server architecture**

The clientserver architecture modeldistinguishes Client (computing) systems fromServer (computing) systems, which communicate over a Computer network. A client-server application is a Distributed system comprised of both client and server software. A client software process may initiate a communication session, while the server waits for requests from any client. Client/server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfills the request. Although programs within a single computer can use the client/server idea, it is a more important idea in a network.

 Computer transactions using the client/server model are very common. Most Internet applications, such as email, web access and database access, are based on the client/server model. For example, a Web browser is a client program at the user computer that may access information at any web server in the world. To check your bank account from your computer, a web browser client program in your computer forwards your request to a web server program at the bank. That program may in turn forward the request to its own database client program that sends a request to a database server at another bank computer to retrieve your account balance. The balance is returned back to the bank database client, which in turn serves it back to the web browser client in your personal computer, which displays the information.

**2.6 Advantages**

In most cases, client-server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers that are known to each other only through a network. This creates an additional advantage to this architecture: greater ease of maintenance. For example, it is possible to replace, repair, upgrade, or even relocate a server while its clients remain both unaware and unaffected by that change. This independence from change is also referred to as Information hiding.

All the data is stored on the servers, which generally have far greater security controls than most clients. Servers can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data.

Since data storage is centralized, updates to that data are far easier to administer than what would be possible under a P2P paradigm. Under a P2P architecture, data updates may need to be distributed and applied to each "peer" in the network, which is both time-consuming and error-prone, as there can be thousands or even millions of peers.

Many mature client-server technologies are already available which were designed to ensure security, 'friendliness' of the user interface, and ease of use.

It functions with multiple different clients of different capabilities.

**2.7 Disadvantages**

Traffic congestion on the network has been an issue since the inception of the client-server paradigm. As the number of simultaneous client requests to a given server increases, the server can become severely overloaded. Contrast that to a P2P network, where its bandwidth actually increases as more nodes are added, since the P2P networks over all bandwidth can be roughly computed as the sum of the bandwidths of every node in that network.

The client-server paradigm lacks the robustness of a good P2P network. Under client-server, should a critical server fail, client’s requests cannot be fulfilled. In P2P networks, resources are usually distributed among many nodes. Even if one or more nodes depart and abandon a downloading file, for example, the remaining nodes should still have the data needed to complete the download. Specific types of clients include Web browser, Email client, and Online chat clients .Specific types of servers include Web server, FTP server, Application Server, Database Server, Mail Server, File Server, Print Server and Terminal Server. Most web services are also types of servers.

**2.8REQUIREMENTS SPECIFICATION**

**2.8.1 Software Required**

The project is implemented in Core Java and JSP (Java Server Pages) as it provides theimplementation of Socket and Server Socket classes that are used to connect distinct applications, hence the software’s required in the creation and execution of the project are j2sdk1.6 or JAVA DEVELOPMENT KIT 6 .As we know JAVA is a platform independent language so this software runs with JRE environment on any desired platform i.e. windows 9x,XP, or 2000 operating system and Apache Tomcat 7.0 as a web server.

**2.8.2 Hardware Required**

As the project does not involve any database, its hardware requirements are minimal. Any System with Pentium P2 or above processor, 32MB RAM, 1GB Hard Disk, a LAN Card, and a CDROM is sufficient. Its network based software so computers connected with any kind of mode (wireless, LAN connected etc) will suit its requirements. It can also be run on a single machine for its demo use. Best suited in laboratory where we can run its server on any machine and many clients can use it simultaneously.

**Chapter 3**

**SYSTEM DESIGN**

The proposed chat application is implemented with three modules namely Client, Server and Database.

**3.1 Data Flow diagram**

The Figure 3.1 shows Data Flow Diagram of our proposed system. It represents that all the three modules are connected to same database through our project.

And only admin has complete access on database server application he can accept new clients or even reject.

**Database**

**Users**

****

**Figure 3.2**  *Data Flow Diagram*

**3.2 Flow chart**

The Flowchart diagram represents the flow of control between different modules.

Start

Initiate Server

**No**

**Yes**

LoginClients

**No**

**Yes**

Start Chatting

****

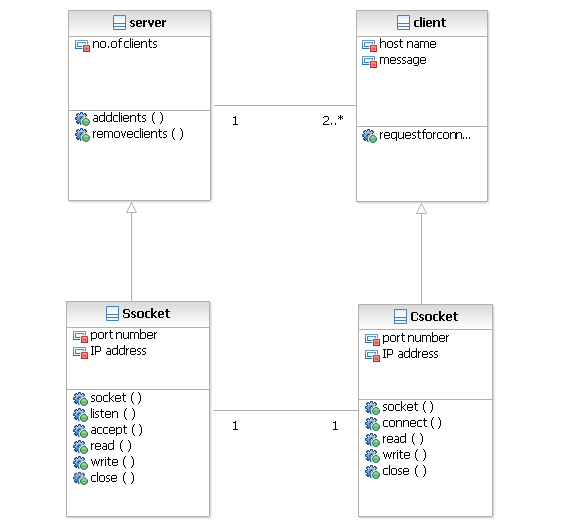
****

Stop

Logout

**3.3 Class diagram**

This diagram represents the different classes needed and how they are interconnected for the implementation of chat application.



**Figure 3.3** *ClassDiagram*

**Chapter 4**

**IMPLEMENTATION**

To implement the chat application we need client socket, server socket to make them communicate with each.

**4.1 SOCKET**

Sockets is a method for communication between a client program and a server program in a network. A socket is defined as "the endpoint in a connection." Sockets are created and used with a set of programming requests or "function calls" sometimes called the sockets application programming interface (API). Sockets can also be used for communication between processes within the same computer.

**4.1.1 Creating server socket**

Server application makes a Server Socket , on a specific port.

* Server Socket s=new Server Socket(5000)

**4.1.2 Creating client socket**

2 things are must to know about the server

* What is its IP address
* What port it’s running on

Client makes socket in following way

* Socket s1=new Socket(“ipaddress”,port)

**4.1.3 Server and client server**

Server creates a new socket for client

* Socket s2=serversocket.accept()

**4.2 Communication between server and client**

Assuming that server module creates a User Object which will have details of the client socket number. So whenever the chat server runs it opens the socket at a pocket and starts looking for a client to connect, and if the client connects it starts a thread process.

In the Client side module, after it gets connected to the chat server it will be connected to the server through the socket and communicates with it.

We use streams once socket connection is made from client to server

* Data is read into socket by making an InputStreamReader chained to the socket’s low level input stream
* DataInputStreamReader=new DataInputStream(sock.getInputStream())

Data is written into sockets by making an output stream reader chained to sockets low level output stream

* DataOutputStream=new DataOutputStream(sock.getOutputStream())

**Chapter 5**

**SOURCE CODE**

**CLIENT CODE:**

packageclientchat;

importcom.sun.corba.se.spi.activation.RepositoryPackage.ServerDef;

importjava.awt.Container;

importjava.awt.FlowLayout;

importjava.awt.event.ActionEvent;

importjava.awt.event.ActionListener;

importjava.io.BufferedReader;

importjava.io.IOException;

importjava.io.InputStreamReader;

importjava.net.ConnectException;

importjava.net.InetAddress;

importjava.net.InetSocketAddress;

importjava.net.SocketAddress;

importjava.net.UnknownHostException;

importjava.nio.ByteBuffer;

importjava.nio.channels.SelectionKey;

importjava.nio.channels.Selector;

importjava.nio.channels.ServerSocketChannel;

importjava.nio.channels.SocketChannel;

importjava.nio.charset.Charset;

importjava.nio.charset.CharsetDecoder;

importjava.util.Iterator;

importjava.util.LinkedList;

importjava.util.Set;

importjavax.swing.JButton;

importjavax.swing.JFrame;

importjavax.swing.JLabel;

importjavax.swing.JScrollPane;

importjavax.swing.JTextArea;

importjavax.swing.JTextField;

public class myFrame extends JFrame{

privateJTextAreaChatBox=new JTextArea(10,45);

privateJScrollPanemyChatHistory=new JScrollPane(ChatBox,JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,

JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

privateJTextAreaUserText = new JTextArea(5,40);

privateJScrollPanemyUserHistory=new JScrollPane(UserText,JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED,

JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

privateJButton Send = new JButton("Send");

privateJButton Start = new JButton("Connect");

private Client ChatClient;

privateReadThreadmyRead=new ReadThread();

privateJTextField Server=new JTextField(20);

privateJLabelmyLabel=new JLabel("Server Name :");

privateJTextField User=new JTextField(20);

private String ServerName;

private String UserName;

publicmyFrame() {

setResizable(false);

setTitle("Client");

setSize(560,400);

Container cp=getContentPane();

cp.setLayout(new FlowLayout());

cp.add(new JLabel("Chat History"));

cp.add(myChatHistory);

cp.add(new JLabel("Chat Box : "));

cp.add(myUserHistory);

cp.add(Send);

cp.add(Start);

cp.add(myLabel);

cp.add(Server);

cp.add(User);

Send.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(ChatClient!=null) {

System.out.println(UserText.getText());

ChatClient.SendMassage(UserText.getText());

}

}

});

Start.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

ChatClient=new Client();

ChatClient.start();

}

});

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) {

newmyFrame();

}

public class Client extends Thread {

private static final int PORT=9999;

privateLinkedList Clients;

privateByteBufferReadBuffer;

privateByteBufferwriteBuffer;

privateSocketChannelSChan;

private Selector ReadSelector;

privateCharsetDecoderasciiDecoder;

public Client() {

Clients=new LinkedList();

ReadBuffer=ByteBuffer.allocateDirect(300);

writeBuffer=ByteBuffer.allocateDirect(300);

asciiDecoder = Charset.forName( "US-ASCII").newDecoder();

}

public void run() {

ServerName=Server.getText();

System.out.println(ServerName);

UserName=User.getText();

Connect(ServerName);

myRead.start();

while (true) {

ReadMassage();

try {

Thread.sleep(30);

} catch (InterruptedExceptionie){

}

}

}

public void Connect(String hostname) {

try {

ReadSelector = Selector.open();

InetAddressaddr = InetAddress.getByName(hostname);

SChan = SocketChannel.open(new InetSocketAddress(addr, PORT));

SChan.configureBlocking(false);

SChan.register(ReadSelector, SelectionKey.OP\_READ, new StringBuffer());

}

catch (Exception e) {

}

}

public void SendMassage(String messg) {

prepareBuffer(UserName+" says: "+messg);

channelWrite(SChan);

}

public void prepareBuffer(String massg) {

writeBuffer.clear();

writeBuffer.put(massg.getBytes());

writeBuffer.putChar('\n');

writeBuffer.flip();

}

public void channelWrite(SocketChannel client) {

longnum=0;

longlen=writeBuffer.remaining();

while(num!=len) {

try {

num+=SChan.write(writeBuffer);

Thread.sleep(5);

} catch (IOException ex) {

ex.printStackTrace();

} catch(InterruptedException ex) {

}

}

writeBuffer.rewind();

}

public void ReadMassage() {

try {

ReadSelector.selectNow();

Set readyKeys = ReadSelector.selectedKeys();

Iterator i = readyKeys.iterator();

while (i.hasNext()) {

SelectionKey key = (SelectionKey) i.next();

i.remove();

SocketChannel channel = (SocketChannel) key.channel();

ReadBuffer.clear();

longnbytes = channel.read(ReadBuffer);

if (nbytes == -1) {

ChatBox.append("You logged out !\n");

channel.close();

} else {

StringBuffersb = (StringBuffer)key.attachment();

ReadBuffer.flip( );

String str = asciiDecoder.decode(ReadBuffer).toString( );

sb.append(str );

ReadBuffer.clear( );

String line = sb.toString();

if ((line.indexOf("\n") != -1) || (line.indexOf("\r") != -1)) {

line = line.trim();

ChatBox.append("> "+ line);

ChatBox.append(""+'\n');

sb.delete(0,sb.length());

}

}

}

} catch (IOExceptionioe) {

} catch (Exception e) {

}

}

}

classReadThread extends Thread {

public void run() {

ChatClient.ReadMassage();

}}}

**SERVER CODE:**

packageserverchat;

importjava.awt.Container;

importjava.awt.FlowLayout;

importjava.awt.event.ActionEvent;

importjava.awt.event.ActionListener;

importjava.io.IOException;

importjava.net.InetAddress;

importjava.net.InetSocketAddress;

importjava.net.ServerSocket;

importjava.net.SocketAddress;

importjava.nio.ByteBuffer;

importjava.nio.channels.SelectionKey;

importjava.nio.channels.Selector;

importjava.nio.channels.ServerSocketChannel;

importjava.nio.channels.SocketChannel;

importjava.nio.charset.Charset;

importjava.nio.charset.CharsetDecoder;

importjava.nio.charset.CoderResult;

importjava.util.Iterator;

importjava.util.LinkedList;

importjava.util.Set;

importjavax.swing.JButton;

importjavax.swing.JFrame;

importjavax.swing.JLabel;

importjavax.swing.JScrollPane;

importjavax.swing.JTextArea;

public class myFrame extends JFrame{

privateJTextAreaChatBox=new JTextArea(10,45);

privateJScrollPanemyChatHistory=new JScrollPane(ChatBox,JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,

JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

privateJTextAreaUserText = new JTextArea(5,40);

privateJScrollPanemyUserHistory=new JScrollPane(UserText,JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED,

JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

privateJButton Send = new JButton("Send");

privateJButton Start = new JButton("Start Server!");

private server ChatServer;

privateInetAddressServerAddress ;

publicmyFrame() {

setTitle("Server");

setSize(560,400);

Container cp=getContentPane();

cp.setLayout(new FlowLayout());

cp.add(new JLabel("Server History"));

cp.add(myChatHistory);

cp.add(new JLabel("Chat Box : "));

cp.add(myUserHistory);

cp.add(Send);

cp.add(Start);

Start.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

ChatServer=new server();

ChatServer.start();

}

});

Send.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

ChatServer.SendMassage(ServerAddress.getHostName()+" < Server > "+UserText.getText());

}

});

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) {

newmyFrame();

}

public class server extends Thread {

private static final int PORT=9999;

privateLinkedList Clients;

privateByteBufferReadBuffer;

privateByteBufferWriteBuffer;

public ServerSocketChannelSSChan;

private Selector ReaderSelector;

privateCharsetDecoderasciiDecoder;

public server() {

Clients=new LinkedList();

ReadBuffer=ByteBuffer.allocateDirect(300);

WriteBuffer=ByteBuffer.allocateDirect(300);

asciiDecoder = Charset.forName( "US-ASCII").newDecoder();

}

public void InitServer() {

try {

SSChan=ServerSocketChannel.open();

SSChan.configureBlocking(false);

ServerAddress=InetAddress.getLocalHost();

System.out.println(ServerAddress.toString());

SSChan.socket().bind(new InetSocketAddress(ServerAddress,PORT));

ReaderSelector=Selector.open();

ChatBox.setText(ServerAddress.getHostName()+"<Server> Started. \n");

} catch (IOException ex) {

ex.printStackTrace();

}

}

public void run() {

InitServer();

while(true) {

acceptNewConnection();

ReadMassage();

try {

Thread.sleep(100);

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

}

public void acceptNewConnection() {

SocketChannelnewClient;

try {

while ((newClient = SSChan.accept()) != null) {

ChatServer.addClient(newClient);

sendBroadcastMessage(newClient,"Login from: " +newClient.socket().getInetAddress());

SendMassage(newClient,ServerAddress.getHostName()+"<server> welcome you !\n Note :To exit" +

" from server write 'quit' .\n");

}

} catch (IOException e) {

e.printStackTrace();

}

}

public void addClient(SocketChannelnewClient) {

Clients.add(newClient);

try {

newClient.configureBlocking(false);

newClient.register(ReaderSelector,SelectionKey.OP\_READ,newStringBuffer());

} catch (IOException ex) {

ex.printStackTrace();

}

}

public void SendMassage(SocketChannel client ,String messg) {

prepareBuffer(messg);

channelWrite(client);

}

public void SendMassage(String massg) {

if(Clients.size()>0) {

for(inti=0;i<Clients.size();i++) {

SocketChannel client=(SocketChannel)Clients.get(i);

SendMassage(client,massg);

}

}

}

public void prepareBuffer(String massg) {

WriteBuffer.clear();

WriteBuffer.put(massg.getBytes());

WriteBuffer.putChar('\n');

WriteBuffer.flip();

}

public void channelWrite(SocketChannel client) {

longnum=0;

longlen=WriteBuffer.remaining();

while(num!=len) {

try {

num+=client.write(WriteBuffer);

Thread.sleep(5);

} catch (IOException ex) {

ex.printStackTrace();

} catch(InterruptedException ex) {

}

}

WriteBuffer.rewind();

}

public void sendBroadcastMessage(SocketChannelclient,Stringmesg) {

prepareBuffer(mesg);

Iterator i = Clients.iterator();

while (i.hasNext()) {

SocketChannel channel = (SocketChannel)i.next();

if (channel != client) {

channelWrite(channel);

}

}

}

public void ReadMassage() {

try {

ReaderSelector.selectNow();

Set readkeys=ReaderSelector.selectedKeys();

Iterator iter=readkeys.iterator();

while(iter.hasNext()) {

SelectionKey key=(SelectionKey) iter.next();

iter.remove();

SocketChannel client=(SocketChannel)key.channel();

ReadBuffer.clear();

longnum=client.read(ReadBuffer);

if(num==-1) {

client.close();

Clients.remove(client);

sendBroadcastMessage(client,"logout: " +

client.socket().getInetAddress());

} else {

StringBufferstr=(StringBuffer)key.attachment();

ReadBuffer.flip();

String data= asciiDecoder.decode(ReadBuffer).toString();

ReadBuffer.clear();

str.append(data);

String line = str.toString();

if ((line.indexOf("\n") != -1) || (line.indexOf("\r") != -1)) {

line = line.trim();

System.out.println(line);

if (line.endsWith("quit")) {

client.close();

Clients.remove(client);

ChatBox.append("Logout: " + client.socket().getInetAddress());

sendBroadcastMessage(client,"Logout: "

+ client.socket().getInetAddress());

ChatBox.append(""+'\n');

} else {

ChatBox.append(client.socket().getInetAddress() + ": " + line);

sendBroadcastMessage(client,client.socket().getInetAddress()

+ ": " + line);

ChatBox.append(""+'\n');

str.delete(0,str.length());

}

}

}

}

} catch (IOException ex) {

ex.printStackTrace();

}

}

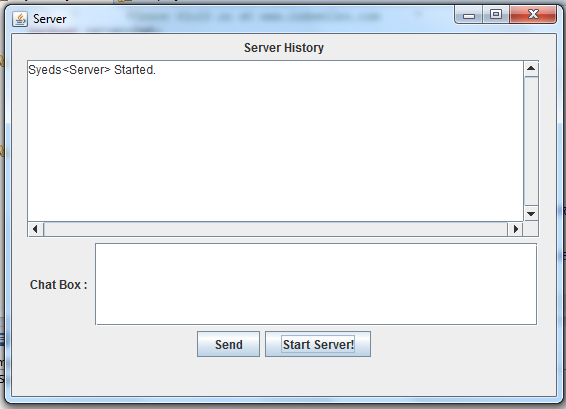
}

}

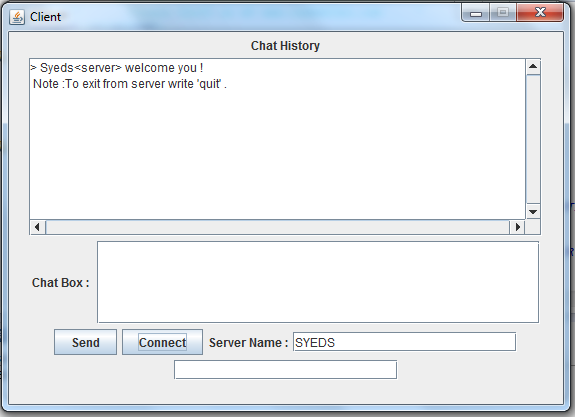
**Chapter 6**

**SCREEN SHOTS**

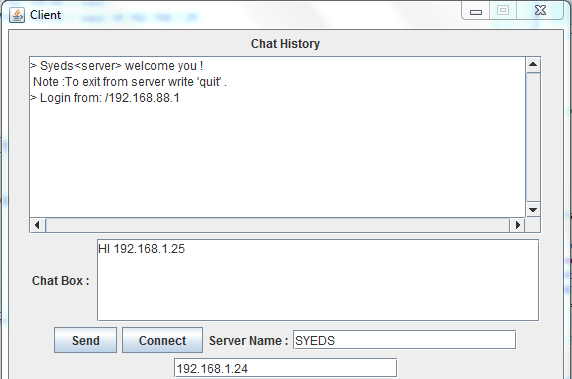
**1.SERVER :**

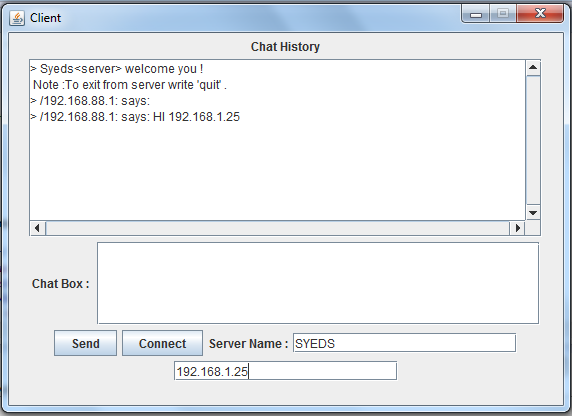
****

**2.CLIENT :**

****

**OUTPUT :**

****

****

**Chapter 7**

**CONCLUSIONS**

**7.1 Conclusions**

Chat server embodies the essence of normal Internet relay chat into customization. This mini project has the option of chatting even without creation of account. This reduces the effort of the user by reducing the overhead of creation of account. In a network two users only by the use of Ip address and Port number we can start a relay chat.

With the help of this mini project I could get a deep and conceptual understanding of Socket programming.

**REFERENCES**

The Java Complete Reference - Herbert Schildt

www.javazoom.in

www.roseindia.net

Java network programming -O'Reilly