B.M.S COLLEGE OF ENGINEERING

(Autonomous college under VTU)

Bull Temple Rd, Basavanagudi, Bengaluru, Karnataka 560019
2021
Department of Computer Applications

Report is submitted for the partial fulfillment of the subject

Java Programming

(20MCA2PCJP)

A report on

"Real-Time Based Chat Application"

By

Name of Student: Riti

USN:1BM20MC043

Under the Guidance

Dr. Ch Ram Mohan Reddy
(Associate Professor)

CONTENTS

ABSTRACT

The project is about a Chat Application based on Java, which uses TCP socket communication which is another module of a remote procedure call. We have a server as well as a client and both can run on the same machine or different machines. If both are running in the same machine, the address to be given at the client side is the local host address.
If both are running in different machines, then on the client side we need to specify the ip address of the machine in which the server application is running. We will be running a Chat Application on sockets and its parameters, to work out with our requirement.
The Java Chat application you are going to build is a console application that is launched from the command line. The server and clients can run on different computers in the same network, e.g. Local Area Network (LAN), or on the same system as Server and Client.
There can be multiple clients connected to a server and they can chat to each other, just like in a chat room where everyone can see other users' messages. Since we are using multi-threading, the same concepts can be used with very slight modification to extend the above idea and create a chatting application similar to facebook messenger, whatsapp etc.

Requirements

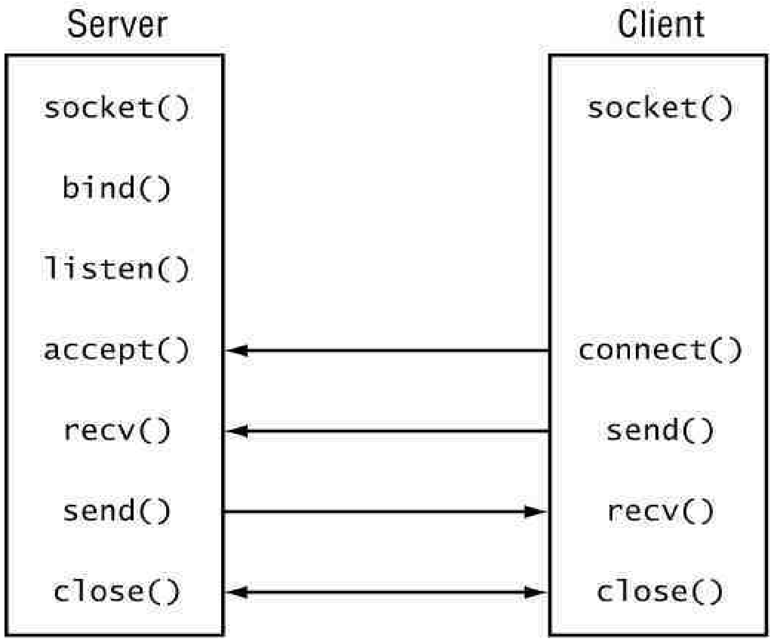
HARDWARE

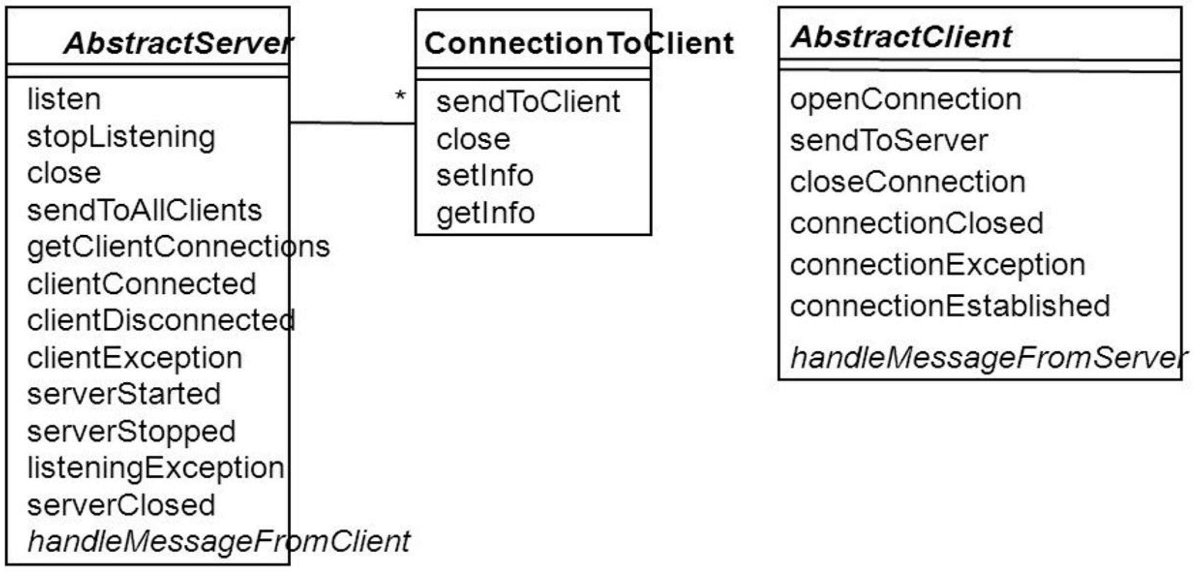
RAM : 8 GB Hard Disk : 1 Terabytes Processor : Intel core i5

SOFTWARE

TextPad8, Jdk8, Swing, Command Terminal, Socket, Multithreading, Operating System.

UML Diagram


Server
Client
socket()
bind()
listen()
accept()
recv()
send()
close()
socket()
connect()
send()
recv()
close()


AbstractServer
listen
stopListening
close
sendToAllClients
getClientConnections
clientConnected
clientDisconnected
clientException
serverStarted
serverStopped
listeningException
serverClosed
handleMessageFromClient
*
ConnectionToClient
sendToClient
close
setInfo
getInfo
AbstractClient
openConnection
sendToServer
closeConnection
connectionClosed
connectionException
connectionEstablished
handleMessageFromServer

Modules

A socket identifies an endpoint in a network.

Socket communication takes place via a protocol.

Internet protocol (IP) is a low level routing protocol that breaks data into small packets and sends them to an address across a network.

TCP (transmission controls protocol) is a higher-level protocol that manages to robustly string together these packets sorting and retransmitting them as necessary to reliably transmit data.

TCP/IP are used to implement reliable, bidirectional, persistent, point to point stream based connections between hosts on the internet.

A socket can be used to connect java's I/O system to other programs that may reside either on local machine or on any other machine on the internet .

There are two kinds of TCP sockets as Java. One is servers and another is clients.

ServerSocket is designed as to be a listener which waits for clients to connect before doing anything.

This is a typical flow of events for a connection-oriented socket:

1. The socket() API creates an endpoint for communications and returns a socket descriptor that represents the endpoint.
2. When an application has a socket descriptor, it can bind a unique name to the socket. Servers must bind a name to be accessible from the network.
3. The listen() API indicates a willingness to accept client connection requests. When a listen() API is issued for a socket, that socket cannot actively initiate connection requests. The listen() API is issued after a socket is allocated with a socket() API and the bind() API binds a name to the socket. A listen() API must be issued before an accept() API is issued.
4. The client application uses a connect() API on a stream socket to establish a connection to the server.
5. The server application uses the accept() API to accept a client connection request. The server must issue the bind() and listen() APIs successfully before it can issue an accept() API.
6. When a connection is established between stream sockets (between client and server), you can use any of the socket API data transfer APIs. Clients and servers have many data transfer APIs from which to choose, such as send(), recv(), read(), write(), and others.
7. When a server or client wants to stop operations, it issues a close() API to release any system resources acquired by the socket.

Code

Client

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

public class Client extends Frame implements ActionListener, Runnable {

    Socket socket;
    Panel topControlsP, bottomControlsP;
    List chatsL;
    TextField ipTF, portTF, messageTF;
    Button connectB, sendB, exitB;
    BufferedReader bufferedReader;
    BufferedWriter bufferedWriter;
    Thread th;

    public Client(String str) {
        super(str);
        //creating controls objects
        topControlsP = new Panel();
        Label ln = new Label("Name");
        TextField ipTF1 = new TextField(10);
        Label lp = new Label("IP Address ");
        ipTF = new TextField(15);
        Label lpo = new Label("Port");
        portTF = new TextField(5);
        connectB = new Button("Connect");
        chatsL = new List();
        bottomControlsP = new Panel();
        messageTF = new TextField(20);
        sendB = new Button("Send");
        exitB = new Button("Exit");
        //adding controls to Panel
        topControlsP.add(ln);
        topControlsP.add(ipTF1);
        topControlsP.add(lp);
        topControlsP.add(ipTF);
        topControlsP.add(lpo);
        topControlsP.add(portTF);
        topControlsP.add(connectB);
        //adding controls to Panel
        bottomControlsP.add(messageTF);
        bottomControlsP.add(sendB);
        bottomControlsP.add(exitB);
        //adding to controls and panel to Frame
        add(topControlsP, BorderLayout.NORTH);
        add(chatsL, BorderLayout.CENTER);
        add(bottomControlsP, BorderLayout.SOUTH);

        //adding listeners on Connect, Send and Exit Button
        connectB.addActionListener(this);
        sendB.addActionListener(this);
        exitB.addActionListener(this);

        setSize(800, 600);//setting size of frame/window
        setLocation(300, 0);//setting location of frame/window on the screen
        setBackground(Color.RED);//setting background for frame/window
        setVisible(true);//setting it to visible state
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getSource().equals(exitB)) {
            System.exit(0);
        } else if (event.getSource().equals(connectB)) {
            try {
                socket = new Socket(ipTF.getText(), Integer.parseInt(portTF.getText()));
                bufferedWriter = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
                bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                ipTF.setText("Connected");
                th = new Thread(this);
                th.start();
            } catch (IOException ioe) {
                ipTF.setText(ioe.getMessage());
            }
        } else {
            try {
                if (bufferedWriter != null) {
                    bufferedWriter.write(messageTF.getText());
                    bufferedWriter.newLine();
                    bufferedWriter.flush();
                    chatsL.add("You: " + messageTF.getText());
                    messageTF.setText("");
                }
            } catch (IOException ioe) {
                ipTF.setText(ioe.getMessage());
            }
        }
    }

    public void run() {
        try {
            socket.setSoTimeout(1);
        } catch (Exception e) {
            ipTF.setText(e.getMessage());
        }
        while (true) {
            try {
                String message = bufferedReader.readLine();
                if(message == null) {
                    break;
                }
                chatsL.add("Server: " + message);
            } catch (Exception e) {

            }
        }
    }

    public static void main(String[] ar) {
        new Client("Client Program");
    }
}
```

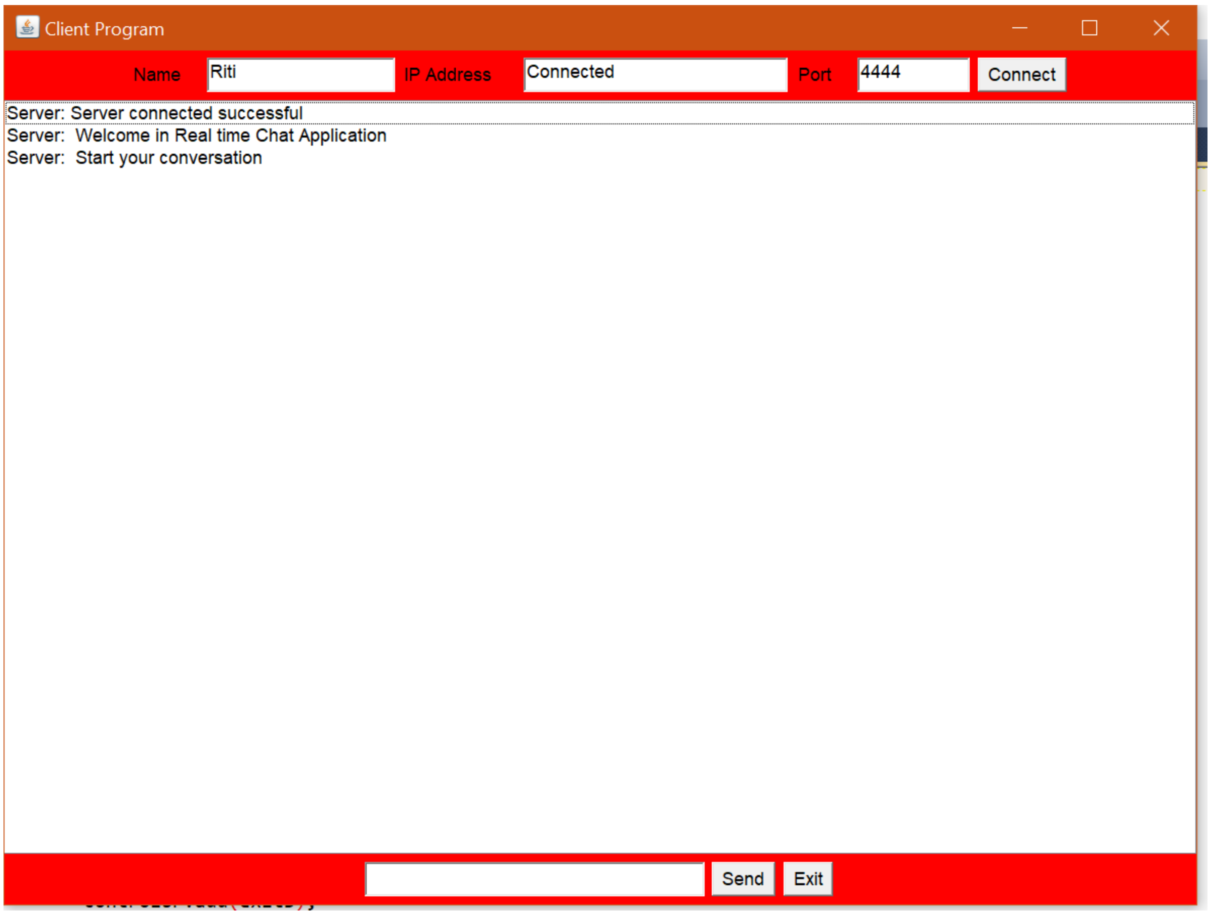
Server

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

public class Server extends Frame implements ActionListener, Runnable {
```

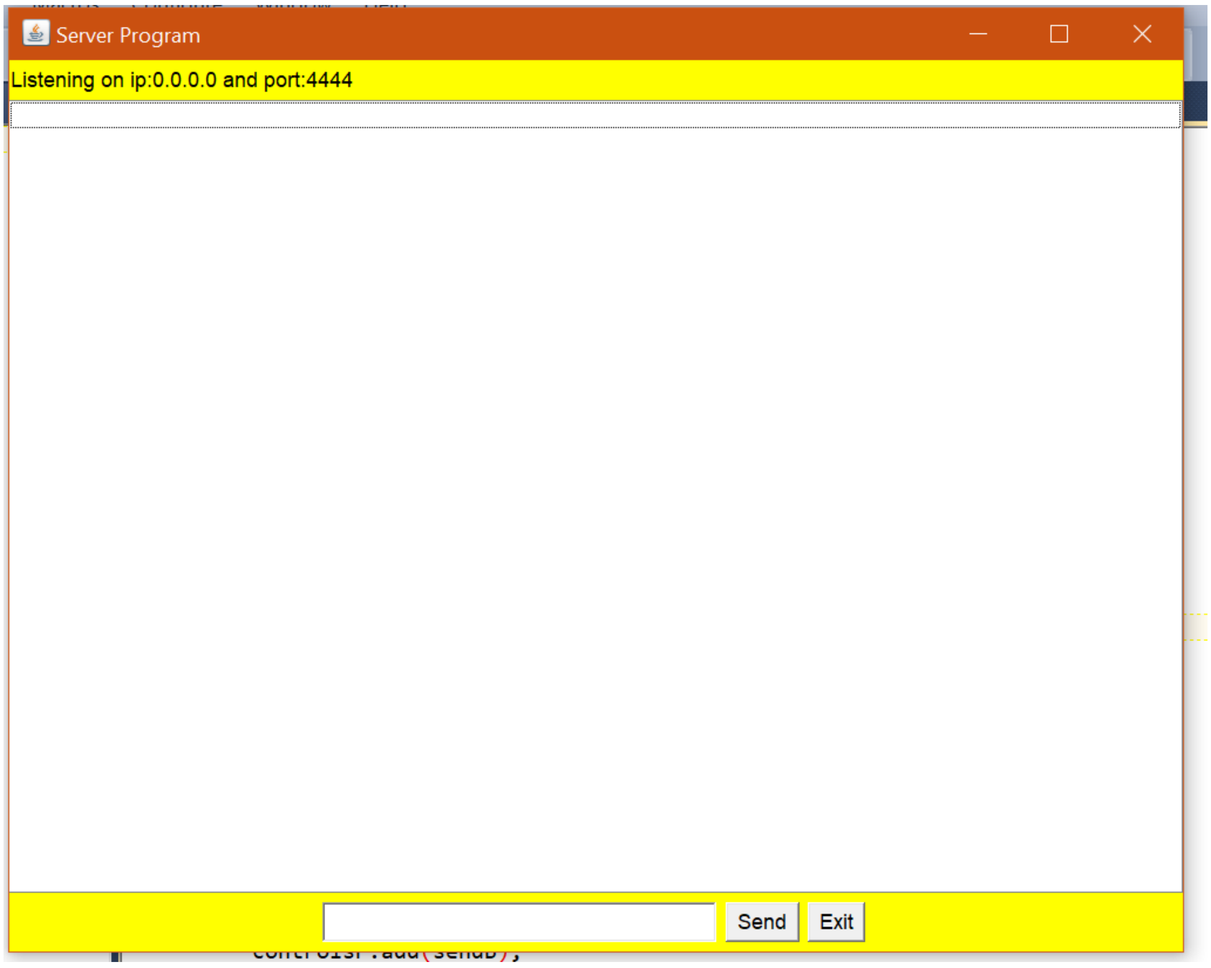
```
int port = 4444;
ServerSocket serverSocket;
Socket socket;
Label statusL;
List chatsL;
Panel controlsP;
TextField messageTF;
Button sendB, exitB;
BufferedReader bufferedReader;
BufferedWriter bufferedWriter;

public Server(String m) {
    super(m);
    //creating controls objects
    statusL = new Label("Status");
    chatsL = new List();
    messageTF = new TextField(20);
    controlsP = new Panel();
    sendB = new Button("Send");
    exitB = new Button("Exit");
    //adding controls to Panel
    controlsP.setLayout(new FlowLayout());
    controlsP.add(messageTF);
    controlsP.add(sendB);
    controlsP.add(exitB);
    //adding to controls and panel to Frame
    add(statusL, BorderLayout.NORTH);
    add(chatsL, BorderLayout.CENTER);
    add(controlsP, BorderLayout.SOUTH);
    //adding listeners on Send Button and Exit Button
    sendB.addActionListener(this);
```

```
        exitB.addActionListener(this);


        setSize(500, 500);//setting size of frame/window
        setLocation(0, 0);//setting location of frame/window on the screen
        setBackground(Color.YELLOW);//setting background for frame/window
        setVisible(true);//setting it to visible state


        listen();
    }


    public void listen() {
        try {
            serverSocket = new ServerSocket(port);
            statusL.setText("Listening on ip:" + serverSocket.getInetAddress().getHostAddress() + " and
port:" + port);
            socket = serverSocket.accept();
            bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            bufferedWriter.write("Server connected successful");
            bufferedWriter.write("\n Welcome in Real time Chat Application");
            bufferedWriter.write("\n Start your conversation");
            bufferedWriter.newLine();
            bufferedWriter.flush();
            Thread th;
            th = new Thread(this);
            th.start();
        } catch (Exception e) {
            statusL.setText(e.getMessage());
        }
    }


    public void actionPerformed(ActionEvent e) {
        if (e.getSource().equals(exitB)) {
            System.exit(0);
        } else {
            try {
                bufferedWriter.write(messageTF.getText());
                bufferedWriter.newLine();
                bufferedWriter.flush();
                chatsL.add("You: " + messageTF.getText());
                messageTF.setText("");
            } catch (IOException ioe) {
                statusL.setText(ioe.getMessage());
            }
        }
    }

    public void run() {
        try {
            socket.setSoTimeout(1000);
        } catch (Exception e) {
        }
        statusL.setText("Client Connected");
        while (true) {
            try {
                String message = bufferedReader.readLine();
                if(message == null) {
                    serverSocket.close();
                    break;
                }
                chatsL.add("Client: " + message);
            } catch (IOException ioe) {


            }
        }
        listen();
    }

    public static void main(String[] ar) {
        new Server("Server Program");
    }
}
```

Output



```
Server: Server connected successful
Server:  Welcome in Real time Chat Application
Server:  Start your conversation
```



```
Listening on ip:0.0.0.0 and port:4444
```

CONCLUSION

Network programming **makes use of socket for interprocess communication between hosts** where sockets act as the endpoint of the interprocess communication. Here sockets can also be termed as network socket or Internet socket since communication between computers is based on Internet protocol.

socket programming over TCP . Network programming makes use of socket for interprocess communication between hosts where sockets act as the endpoint of the interprocess communication. Here sockets can also be termed as network socket or Internet socket since communication between computers is based on Internet protocol. So Network programming is also Socket Programming. The paper also describes about socket programming in java over TCP. Because java has been preferred more than any other language for establishing connections between clients and servers using sockets. Socket programming in java is easy.