DESCRIPTION

In this project, we will learn to create a Client-Server console application in Java. The project will help us to understand the low-level network communication, and also a low-level understanding of how popular chat applications like messenger and WhatsApp are built.

**Sockets** Programming helps us to communicate with the various computers running on a network. In Java, Socket programming can be either connection-oriented or connectionless. We will design the connection-oriented application that uses the Client-Server model.

In the **Client-Server model**, the Server has a unique IP Address and port number. The client tries to make a connection with the server using this port number and IP address. The server listens and accepts the connection. Once the connection is built, the Server can receive a message from the client as well as respond back a message to the client.

Since the project involves multiple clients that can send messages to each other, we will use threads. The thread ensures that every client gets their own server socket.

For the Server application, we use the ServerSocket Class that binds to a specific port number. We then create a server socket and listen for a connection with the Client and accept it. The thread ServerThread is instantiated and started. All the threads are added to an ArrayList so that multiple clients can send messages to each other.

In ServerThread.java, we received sockets and a list of active threads from the Main.java using constructor. When we start the thread from main, the run method is called. The BuffereReader helps us to receive information from the client. Any information that Server wants to send is sent using PrintWriter. The method printToALLClients() sends the output to each client in the thread.

For the Client, we use the Socket class and initiate the connection to a server bypassing the IP address and port number. We use the Scanner to get the input from the user and send the data to the server using the PrintWriter object.

We have used ClientThread class to listen to the response from the server, without getting blocked while reading from a Scanner. We have used input BufferReader to get information from the client.

CODE

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

public class Server {

private final ServerSocket serverSocket;

public Server(ServerSocket serverSocket) {

this.serverSocket = serverSocket;

}

public void startServer() {

try {

while (!serverSocket.isClosed()) {

Socket socket = serverSocket.accept();

System.out.println("A new client has connected!");

ClientHandler clientHandler = new ClientHandler(socket);

Thread thread = new Thread(clientHandler);

thread.start();

}

} catch (IOException e) {

closeServerSocket();

}

}

public void closeServerSocket() {

try {

if (serverSocket != null) {

serverSocket.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(1234);

Server server = new Server(serverSocket);

server.startServer();

}

}

import java.io.\*;

import java.net.Socket;

import java.util.ArrayList;

public class ClientHandler implements Runnable {

public static ArrayList<ClientHandler> clientHandlers = new ArrayList<>();

private Socket socket;

private BufferedReader bufferedReader;

private BufferedWriter bufferedWriter;

private String clientUsername;

public ClientHandler(Socket socket) {

try {

this.socket = socket;

this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

this.bufferedWriter= new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

this.clientUsername = bufferedReader.readLine();

clientHandlers.add(this);

broadcastMessage("SERVER: " + clientUsername + " has entered the chat!");

} catch (IOException e) {

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

@Override

public void run() {

String messageFromClient;

while (socket.isConnected()) {

try {

messageFromClient = bufferedReader.readLine();

broadcastMessage(messageFromClient);

} catch (IOException e) {

closeEverything(socket, bufferedReader, bufferedWriter);

break;

}

}

}

public void broadcastMessage(String messageToSend) {

for (ClientHandler clientHandler : clientHandlers) {

try {

if (!clientHandler.clientUsername.equals(clientUsername)) {

clientHandler.bufferedWriter.write(messageToSend);

clientHandler.bufferedWriter.newLine();

clientHandler.bufferedWriter.flush();

}

} catch (IOException e) {

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

}

public void removeClientHandler() {

clientHandlers.remove(this);

broadcastMessage("SERVER: " + clientUsername + " has left the chat!");

}

public void closeEverything(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter) {

removeClientHandler();

try {

if (bufferedReader != null) {

bufferedReader.close();

}

if (bufferedWriter != null) {

bufferedWriter.close();

}

if (socket != null) {

socket.close();

}

} catch (IOException e) {

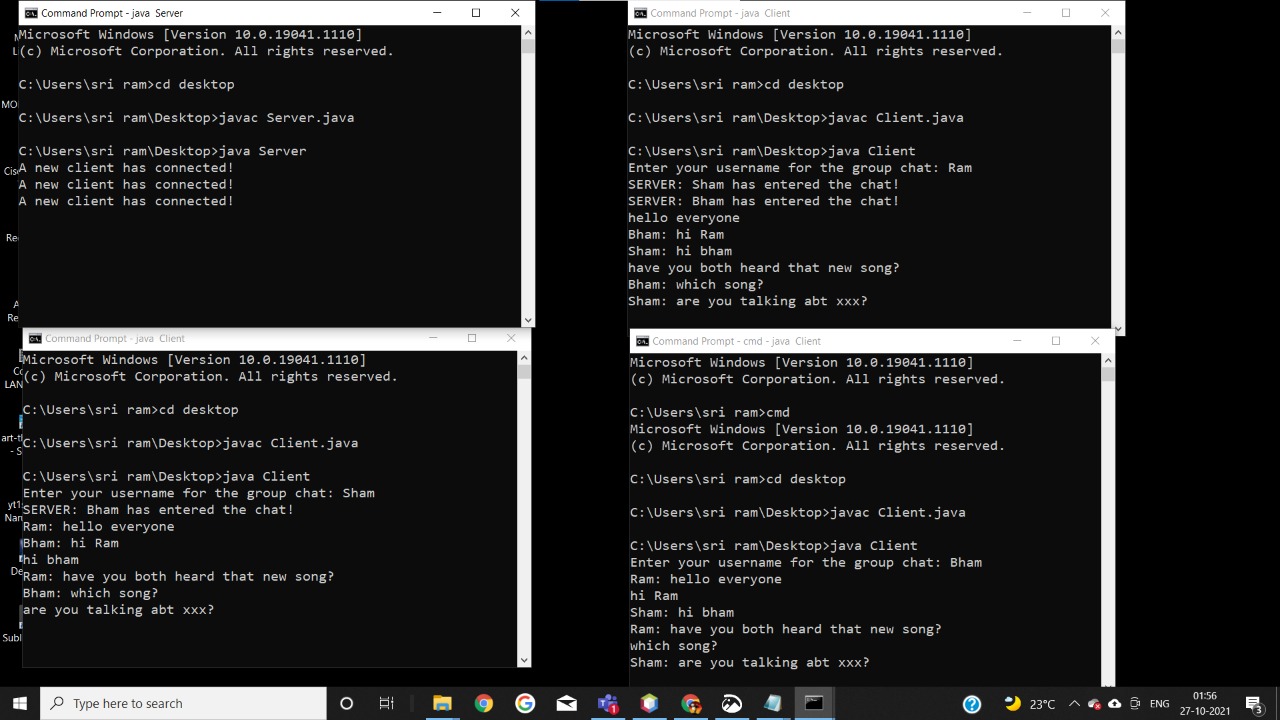
e.printStackTrace();

}

}

}

OUTPUT OF THIS CODE



CONCLUSION

At last we would like to conclude that this project is about MULTI CHATTING APPLICATION USING JAVA SOCKET. Hope this project was helpful to understand the concept of networking and the client server model