

Nikit Gokhe

Class – Comp D1

Roll No. 224024

Gr. No. 21810522

## **Assignment 2**

**Aim-:** Implement chat server using socket programming

**Theory -:**

### **Java Socket Programming**

Java Socket programming is used for communication between the applications running on different JRE.

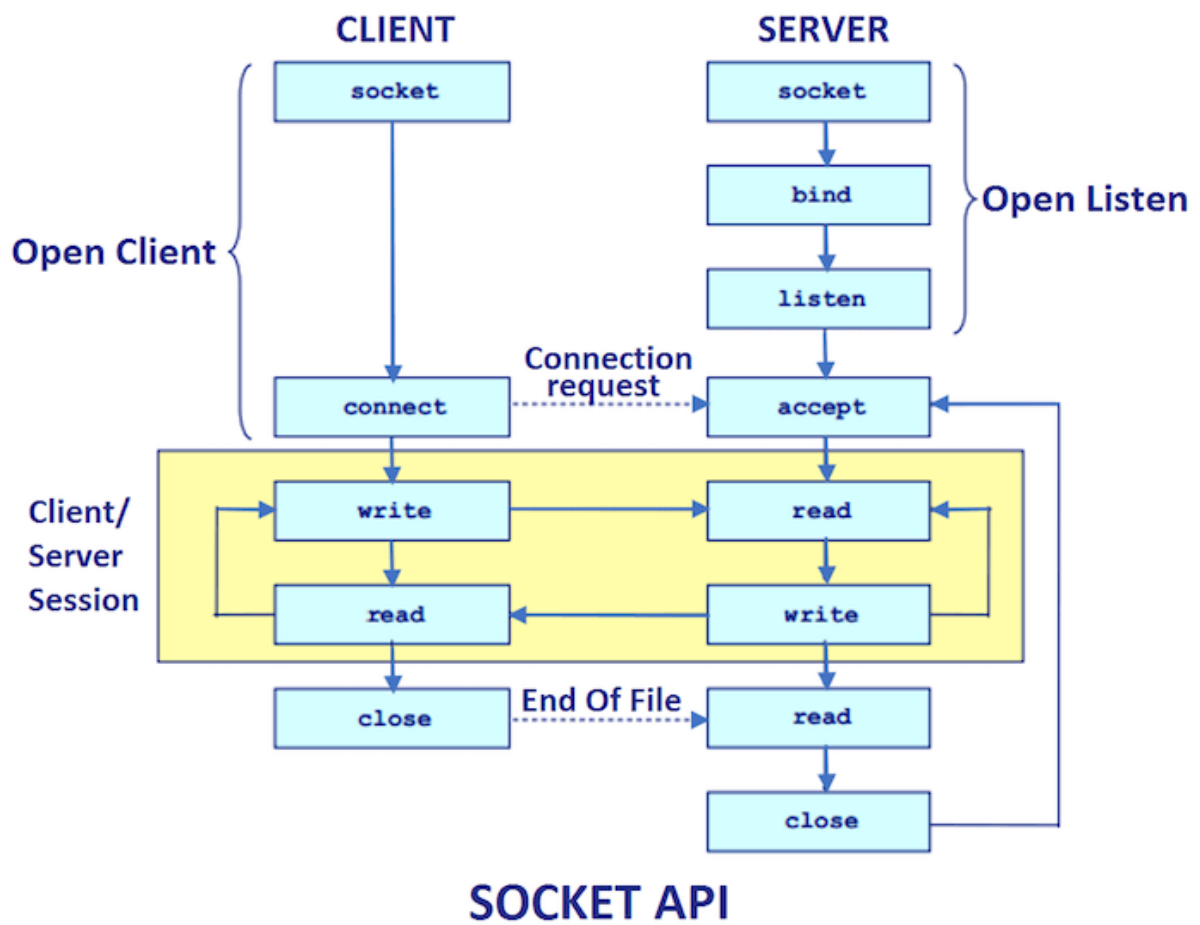
Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



## Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

### Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

### ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

### Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Code -:

### MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("Bye")){
str=din.readUTF();
System.out.println("Client: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

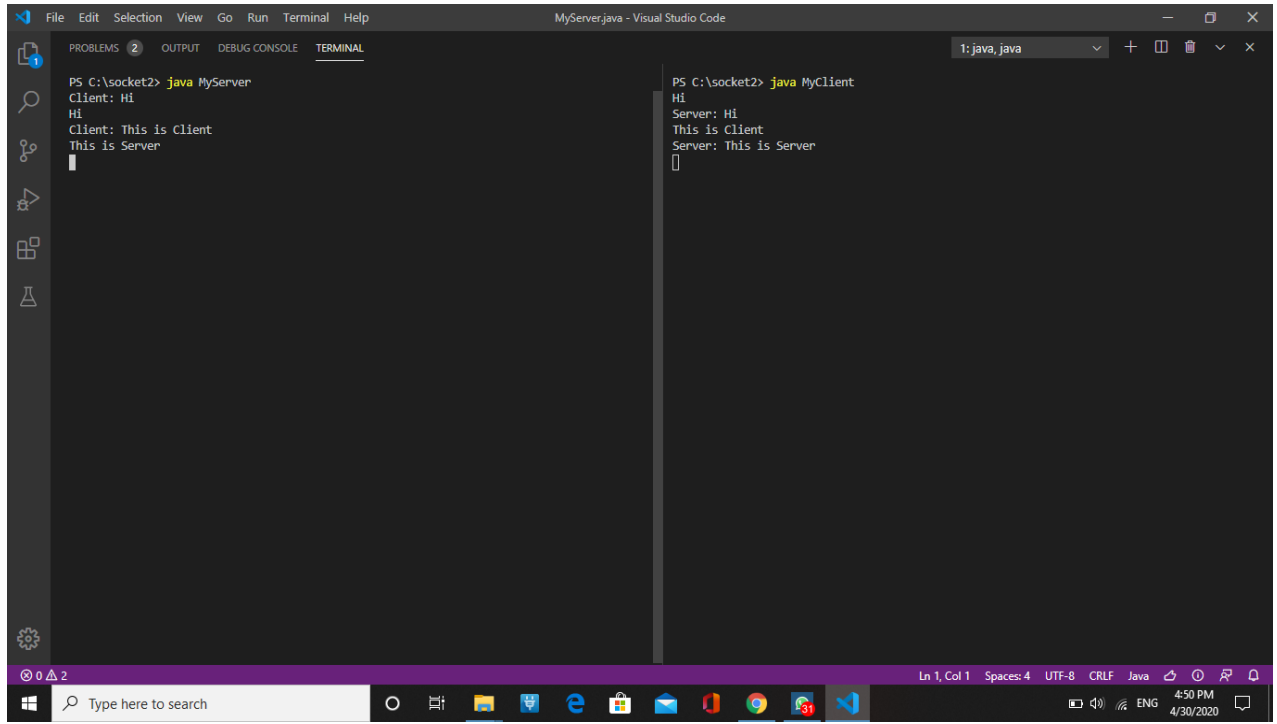
## MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server: "+str2);
}

dout.close();
s.close();
}}
```

## Output -:



The screenshot displays the Visual Studio Code interface with the 'TERMINAL' tab active. The terminal window shows the execution of two Java programs, MyServer and MyClient, which communicate over a socket. The output for MyServer shows it receiving 'Hi' and 'This is Client' from the client, and sending 'Hi' and 'This is Server' back. The output for MyClient shows it sending 'Hi' and 'This is Client' to the server, and receiving 'Hi' and 'This is Server' from the server.

```
PS C:\socket2> java MyServer
Client: Hi
Hi
Client: This is Client
This is Server

PS C:\socket2> java MyClient
Hi
Server: Hi
This is Client
Server: This is Server
```