

Experiment No: 03

Aim: Write a program to demonstrate Client - Server Chat application

Theory:

Group Communication

Group communication is a paradigm for multi-party communication that is based on the notion of groups as a main abstraction. A group is a set of parties that, presumably, want to exchange information in a reliable, consistent manner. For example:

- The participants of a message-based conferencing tool may constitute a group. Ideally, in order to have meaningful communication, each participant wants to receive all communicated messages from each other participant. Moreover, if one message is a response to another, the original message should be delivered before the response
- The set of replicas of a fault-tolerant database server may constitute a group. Consider update messages to the server. Since the contents of the database depend on the history of all update messages received, all updates must be delivered to all replicas. Furthermore, all updates must be delivered in the same order. Otherwise, inconsistencies may arise.

Group Communication Primitives

Group communication is implemented using middleware that provides two sets of primitives to the application:

- Multicast primitive (e.g., post): This primitive allows a sender to post a message to the entire group.
- Membership primitives (e.g., join, leave, query_membership): These primitives allow a process to join or leave a particular group, as well as to query the group for the list of all current participants.

Group Communication Semantics

Different approaches to group communication present different delivery semantics concerning the post primitive. Several alternatives are possible:

- Best effort delivery: No guarantees are given on message delivery. No guarantees are given on the order of messages delivered to the destination. Obviously, this is not a useful abstraction.
- Reliable messages delivery: The communication subsystem ensures the following three properties:
If a correct process broadcasts message *m*, then all correct processes eventually deliver *m*.
If a correct process delivers message *m*, then all correct processes eventually deliver *m*. (Note that the difference of this property from the previous one is that this property should hold even if the sender was not a correct process. This is important to ensure

consistency even when the sender crashes in the middle of a broadcast after some correct receivers have already delivered the message).

Every process delivers m at most once, and only if it was previously broadcast.

- **FIFO message delivery:** It's a reliable message delivery that also ensures that messages from the same source are delivered in the order they were sent. (If a process broadcasts message m before broadcasting message m' then no correct process delivers m' unless it has previously delivered m). FIFO delivery does not guarantee that messages from different sources will be delivered in the same order. For example, if process A responds to a message from process B, these messages can be delivered in different orders to processes C and D who are listening to the conversation. This is because the messages do not originate from the same sender. TCP implements FIFO delivery for the special case of a group of two participants. (Note that: a. Every byte gets delivered. b. Bytes from the same sender are delivered in order.)
- **Causal message delivery:** It generalizes FIFO message delivery to a scenario where messages are delivered in the order of potential causality. In other words, if message m' could have been a response to m , all correct processes must deliver m before m' . Causal message delivery delivers messages in the order they were sent according to Lamport's happened before relation. Lamport proposed that in the absence of global time in a distributed system event A happens before event B is if:
Event A precedes event B on the same processor
 - Event A is that of sending message m , and event B is that of receiving the same message m
 - Event A is shown to occur before B by transitivity (e.g., A happens before C which happens before B)

Totally ordered message delivery: All messages are delivered in the same order to all destinations

The steps for creating a simple server program are:

1. Open the Server Socket: `ServerSocket server = new ServerSocket(PORT);`
2. Wait for the Client Request: `Socket client = server.accept();`
3. Create I/O streams for communicating to the client
`DataInputStream is = new DataInputStream(client.getInputStream());`
`DataOutputStream os = new DataOutputStream(client.getOutputStream());`
4. Perform communication with client Receive from client:
`String line = is.readLine();`
Send to client: `os.writeBytes("Hello\n");`
5. Close socket: `client.close();`

The steps for creating a simple client program are:

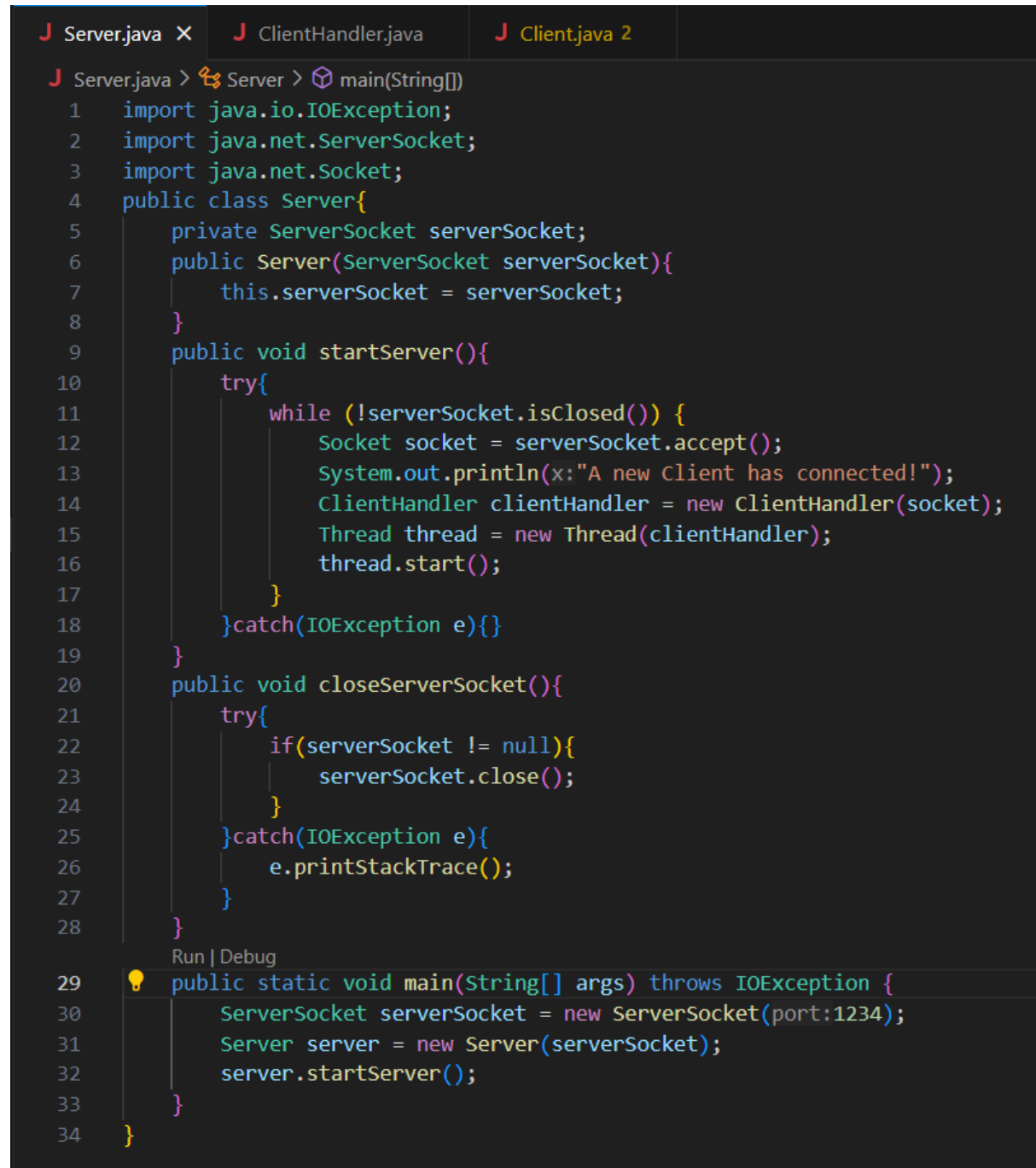
- 1 Create a Socket Object: `Socket client = new Socket(server, port_id);`
- 2 Create I/O streams for communicating with the server.
`is = new DataInputStream(client.getInputStream());`
`os = new DataOutputStream(client.getOutputStream());`
- 3 Perform I/O or communication with the server: Receive data from the server:
`String line = is.readLine();`

- Send data to the server:
os.writeBytes("Hello\n");
- 4 Close the socket when done: client.close();

Implementation:

Code

Server.java



```
J Server.java X J ClientHandler.java J Client.java 2
J Server.java > Server > main(String[])
1 import java.io.IOException;
2 import java.net.ServerSocket;
3 import java.net.Socket;
4 public class Server{
5     private ServerSocket serverSocket;
6     public Server(ServerSocket serverSocket){
7         this.serverSocket = serverSocket;
8     }
9     public void startServer(){
10        try{
11            while (!serverSocket.isClosed()) {
12                Socket socket = serverSocket.accept();
13                System.out.println(x:"A new Client has connected!");
14                ClientHandler clientHandler = new ClientHandler(socket);
15                Thread thread = new Thread(clientHandler);
16                thread.start();
17            }
18        }catch(IOException e){}
19    }
20    public void closeServerSocket(){
21        try{
22            if(serverSocket != null){
23                serverSocket.close();
24            }
25        }catch(IOException e){
26            e.printStackTrace();
27        }
28    }
29    Run | Debug
30    public static void main(String[] args) throws IOException {
31        ServerSocket serverSocket = new ServerSocket(port:1234);
32        Server server = new Server(serverSocket);
33        server.startServer();
34    }
```

ClientHandler.java

```

1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.IOException;
4  import java.io.InputStreamReader;
5  import java.io.OutputStreamWriter;
6  import java.net.Socket;
7  import java.util.ArrayList;
8  public class ClientHandler implements Runnable{
9      public static ArrayList<ClientHandler> clientHandlers = new ArrayList<>();
10     private Socket socket;
11     private BufferedReader bufferedReader;
12     private BufferedWriter bufferedWriter;
13     private String clientUserName;
14     public ClientHandler(Socket socket){
15         try{
16             this.socket = socket;
17             this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
18             this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
19             this.clientUserName = bufferedReader.readLine();
20             clientHandlers.add(this);
21             broadcastMessage("SERVER"+ clientUserName + " has entered the chat!");
22         }catch(IOException e){
23             closeEverything(socket, bufferedReader, bufferedWriter);
24         }
25     }
26     @Override
27     public void run() {
28         String messageFromClient;
29         while(socket.isConnected()){
30             try{
31                 messageFromClient = bufferedReader.readLine();
32                 broadcastMessage(messageFromClient);
33             }catch(IOException e){
34                 closeEverything(socket, bufferedReader, bufferedWriter);
35                 break;
36             }
37         }
38     }
39     public void broadcastMessage(String messageToSend){
40         for(ClientHandler clientHandler : clientHandlers){
41             try{
42                 if(!clientHandler.clientUserName.equals(clientUserName)){
43                     clientHandler.bufferedWriter.write(messageToSend);
44                     clientHandler.bufferedWriter.newLine();
45                     clientHandler.bufferedWriter.flush();
46                 }
47             }catch(IOException e){
48                 closeEverything(socket, bufferedReader, bufferedWriter);
49             }
50         }
51     }
52     public void removeClientHandler(){
53         clientHandlers.remove(this);
54         broadcastMessage("SERVER"+ clientUserName+ " has left the chat!");
55     }
56     public void closeEverything(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter){
57         removeClientHandler();
58         try{
59             if(bufferedReader != null){
60                 bufferedReader.close();
61             }
62             if(bufferedWriter != null){
63                 bufferedWriter.close();
64             }
65             if(socket != null){
66                 socket.close();
67             }
68         }catch(IOException e){
69             e.printStackTrace();
70         }
71     }
72 }

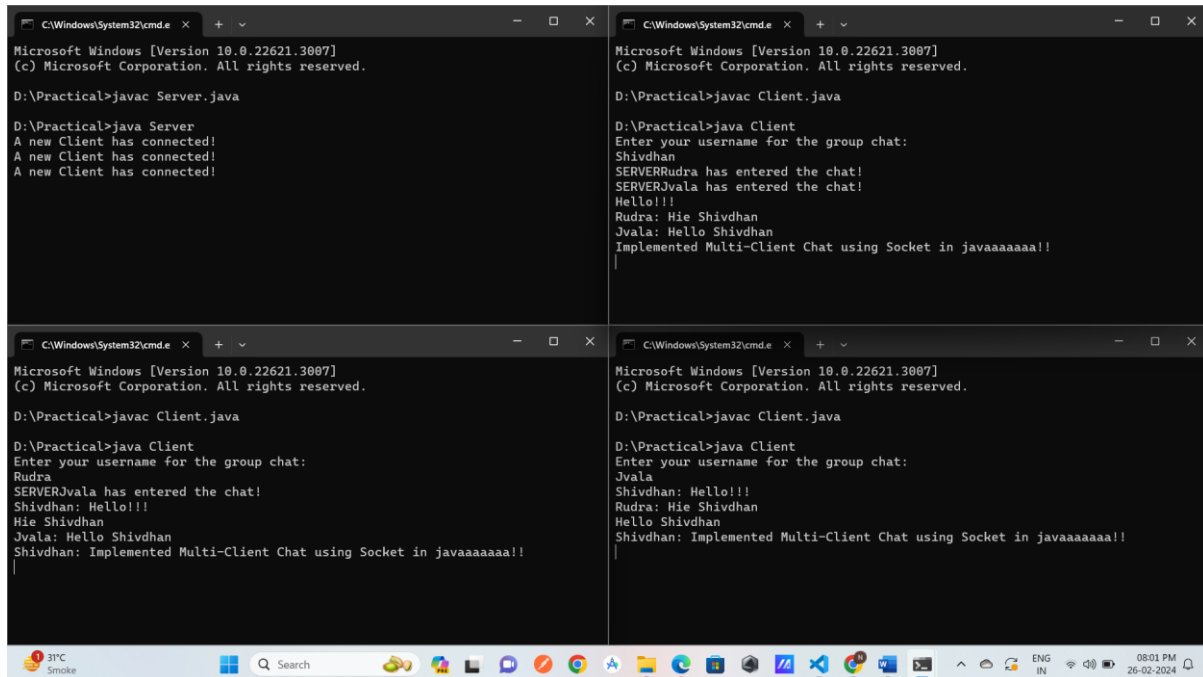
```

Client.java

```
1 import java.io.BufferedReader;
2 import java.io.BufferedWriter;
3 import java.io.IOException;
4 import java.io.InputStreamReader;
5 import java.io.OutputStreamWriter;
6 import java.net.Socket;
7 import java.util.Scanner;
8
9 public class Client {
10     private Socket socket;
11     private BufferedReader bufferedReader;
12     private BufferedWriter bufferedWriter;
13     private String username;
14
15     public Client(Socket socket, String username){
16         try{
17             this.socket = socket;
18             this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
19             this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
20             this.username = username;
21         }catch(IOException e){
22             closeEverything(socket, bufferedReader, bufferedWriter);
23         }
24     }
25
26     public void sendMessage(){
27         try{
28             bufferedWriter.write(username);
29             bufferedWriter.newLine();
30             bufferedWriter.flush();
31
32             Scanner scanner = new Scanner(System.in);
33             while (socket.isConnected()) {
34                 String messageToSend = scanner.nextLine();
35                 bufferedWriter.write(username + ": " + messageToSend);
36                 bufferedWriter.newLine();
37                 bufferedWriter.flush();
38             }
39         }catch(IOException e){
40             closeEverything(socket, bufferedReader, bufferedWriter);
41         }
42     }
43
44     public void listenForMessage(){
45         new Thread(new Runnable() {
46             public void run(){
47                 String messageFromGroupChat;
48
49                 while (socket.isConnected()) {
50                     try{
51                         messageFromGroupChat = bufferedReader.readLine();
52                         System.out.println(messageFromGroupChat);
53                     }catch(IOException e){
54                         closeEverything(socket, bufferedReader, bufferedWriter);
55                     }
56                 }
57             }
58         }).start();
59     }
60
61     public void closeEverything(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter){
62         try{
63             if(bufferedReader != null){
64                 bufferedReader.close();
65             }
66             if(bufferedWriter != null){
67                 bufferedWriter.close();
68             }
69             if(socket != null){
70                 socket.close();
71             }
72         }catch(IOException e){
73             e.printStackTrace();
74         }
75     }
76
77     public static void main(String[] args) throws IOException {
78         Scanner scanner = new Scanner(System.in);
79         System.out.println("Enter your username for the group chat: ");
80         String userName = scanner.nextLine();
81         Socket socket = new Socket("localhost", 1234);
82         Client client = new Client(socket, userName);
83         client.listenForMessage();
84         client.sendMessage();
85     }
86 }
87
```

Output:

Clients



```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

D:\Practical>javac Server.java

D:\Practical>java Server
A new Client has connected!
A new Client has connected!
A new Client has connected!

C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

D:\Practical>javac Client.java

D:\Practical>java Client
Enter your username for the group chat:
Shivdhan
SERVERRudra has entered the chat!
SERVERJvala has entered the chat!
Hello!!!
Rudra: Hie Shivdhan
Jvala: Hello Shivdhan
Implemented Multi-Client Chat using Socket in javaaaaaaa!!

C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

D:\Practical>javac Client.java

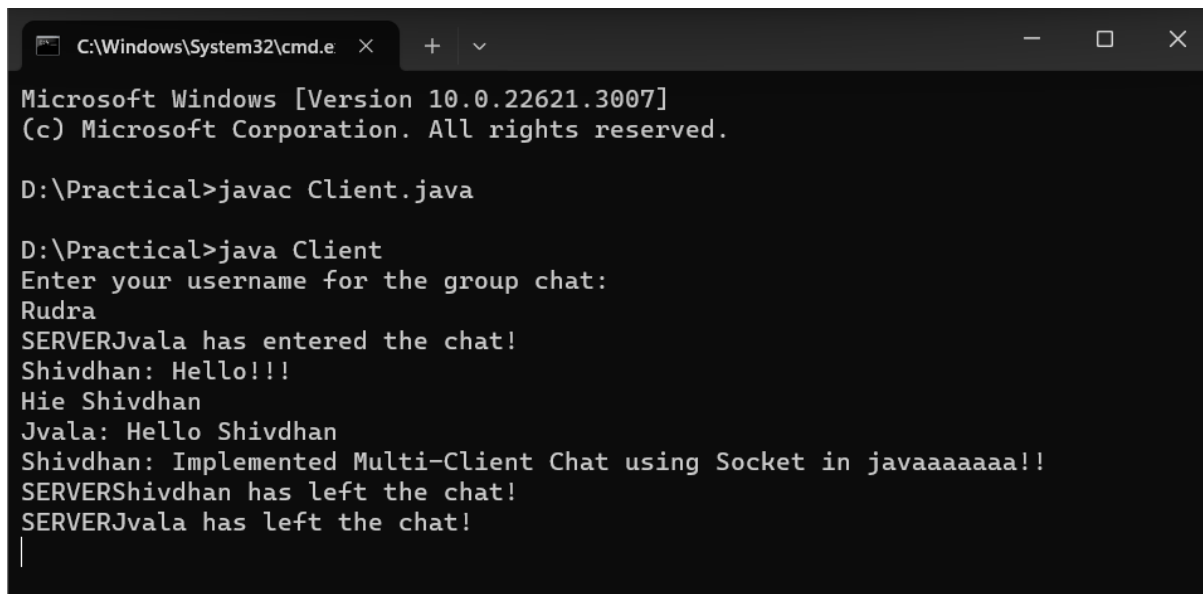
D:\Practical>java Client
Enter your username for the group chat:
Rudra
SERVERJvala has entered the chat!
Shivdhan: Hello!!!
Hie Shivdhan
Jvala: Hello Shivdhan
Shivdhan: Implemented Multi-Client Chat using Socket in javaaaaaaa!!

C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

D:\Practical>javac Client.java

D:\Practical>java Client
Enter your username for the group chat:
Jvala
Shivdhan: Hello!!!
Rudra: Hie Shivdhan
Hello Shivdhan
Shivdhan: Implemented Multi-Client Chat using Socket in javaaaaaaa!!
```

Server



```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

D:\Practical>javac Client.java

D:\Practical>java Client
Enter your username for the group chat:
Rudra
SERVERJvala has entered the chat!
Shivdhan: Hello!!!
Hie Shivdhan
Jvala: Hello Shivdhan
Shivdhan: Implemented Multi-Client Chat using Socket in javaaaaaaa!!
SERVERShivdhan has left the chat!
SERVERJvala has left the chat!
```

Conclusion:

Thus, we have successfully completed chat application to demonstrate group communication. This application consists of a chat server and a chat client. The server accepts connections from the clients and delivers all the messages from each client to other client.