

Name of Student : Bhavesh Vsant Laku		
Roll Number : 28		LAB Assignment Number: 1
Title of LAB Assignment: To develop a multi-client chat server application where multiple clients chat with each other concurrently, where the messages sent by different clients are first communicated to the server and then the server, on behalf of the source client, communicates the messages to the appropriate destination client		
DOP : 05-08-2024		DOS : 12-08-2024
CO Mapped: CO1	PO Mapped: PO3, PO5, PO7, PO12, PSO1, PSO2	Signature:

Practical No. 1

Aim:

To develop a multi-client chat server application where multiple clients can chat with each other concurrently. The server receives messages from clients and forwards them to the intended destination clients.

Description:

A multi-client chat server application allows multiple clients to connect to a central server and communicate with each other in real-time. Each client sends messages to the server, which then forwards these messages to the appropriate destination clients. This setup enables concurrent communication, allowing multiple users to chat simultaneously.

Concepts:

1. **Remote Procedure Call (RPC):**
 - RPC allows a program to execute code on a remote server as if it were a local procedure call. While the chat application does not explicitly use RPC, it operates similarly: client requests (messages) are handled by the server and then forwarded to other clients. The server acts as a remote entity processing and distributing messages.
2. **Socket Programming:**
 - A socket is an endpoint for communication between two machines. Socket programming involves creating sockets on both the server and client sides to establish a network connection and facilitate communication.
3. **Types of Sockets:**
 - **TCP (Transmission Control Protocol) Sockets:** Reliable, connection-oriented communication (used in our chat server).
 - **UDP (User Datagram Protocol) Sockets:** Connectionless communication, typically faster but less reliable.
4. The chat server and clients use TCP sockets to establish reliable connections. The server listens for incoming connections, and clients connect to the server using sockets to send and receive messages.
5. **ServerSocket and Socket Classes:**
 - **ServerSocket Class:** Provides mechanisms to listen for incoming client connections on a specific port and establishes a connection when a client attempts to connect.
 - **Socket Class:** Represents the client-side endpoint of the connection. It is used by the client to connect to the server and by the server to communicate with each connected client.
 - **ServerSocket:** Used by the server to listen for and accept client connections.
 - **Socket:** Used by clients to connect to the server and by the server to communicate with clients.
6. **Concurrency:**

- The ability to execute multiple tasks simultaneously. In a multi-client chat server, concurrency is crucial for handling multiple clients connecting and sending messages simultaneously. The server uses separate threads to handle each client connection concurrently, ensuring that one client's actions do not block or delay others.

How It Works:

1. Server Initialization:

- The server starts by creating a **ServerSocket** on a specified port and listens for incoming client connections.
- When a client attempts to connect, the server accepts the connection and creates a new **Socket** object for communication.
- The server spawns a new thread (using the **ClientHandler** class) for each connected client, allowing concurrent handling.

2. Client Connection:

- Each client creates a **Socket** object and connects to the server using the server's IP address and port number.
- Upon successful connection, the client sends its name to the server and enters the chat room.

3. Message Communication:

- Clients send messages to the server, which broadcasts the messages to all other connected clients.
- The server appends the sender's name to each message to identify the sender.
- The **ClientHandler** class on the server reads messages from clients and forwards them to other clients.

4. Disconnection:

- When a client disconnects (e.g., closes the application), the server removes the client from the list of active clients and broadcasts a message to notify other clients.

Advantages of This Architecture:

1. **Scalability:** The server can handle multiple clients simultaneously, making it suitable for large chat rooms.
2. **Centralized Control:** The server acts as a central point for message distribution, simplifying communication management.
3. **Real-Time Communication:** Using threads allows real-time communication between clients, minimizing delay.

Challenges:

1. **Concurrency Management:** Properly managing threads and ensuring thread safety is crucial to prevent race conditions or deadlocks.
2. **Error Handling:** Robust error handling is necessary to manage unexpected client disconnections or network failures.

3. **Security:** Ensuring secure communication between clients and the server (e.g., using encryption) is important for protecting user data.

Steps to Run the Chat Application:

1. **Create a New Java Project:**
 - Open Eclipse.
 - Go to **File > New > Java Project**.
 - Name the project (e.g., **ChatApplication**) and click **Finish**.
2. **Create the Server Class:**
 - Right-click on the **src** folder in your project.
 - Select **New > Class**.
 - Name the class **Server** and check the box for **public static void main(String[] args)**.
 - Click **Finish**.
 - Copy and paste the **Server.java** code into the **Server** class.
3. **Create the Client Class:**
 - Right-click on the **src** folder in your project.
 - Select **New > Class**.
 - Name the class **Client** and check the box for **public static void main(String[] args)**.
 - Click **Finish**.
 - Copy and paste the **Client.java** code into the **Client** class.
4. **Running the Server and Client:**
 - **Running the Server:**
 - Open the **Server.java** file.
 - Right-click on the file and select **Run As > Java Application**.
 - The server will start running and will wait for client connections.
 - **Running the Client:**
 - To run the client, open a new Eclipse terminal.
 - Option 1: Open another instance of Eclipse.
 - Option 2: Use the same Eclipse instance and run the client in a new console.
 - Open the **Client.java** file in the new instance.
 - Right-click on the file and select **Run As > Java Application**.
 - The client will start, and you'll be prompted to enter your name to join the chat room.
5. **Running Multiple Clients:**
 - Repeat the steps above to open additional instances of the client.
 - Each client instance will connect to the server, allowing you to simulate multiple users chatting simultaneously.

Code:

ChatServer.java:

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ChatServer {
    private static Set<PrintWriter> clientWriters = new HashSet<>();
    public static void main(String[] args) {
        System.out.println("Chat server started...");
        int port = 12345;
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            while (true) {
                new ClientHandler(serverSocket.accept()).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static class ClientHandler extends Thread {
        private Socket socket;
        private PrintWriter out;
        private BufferedReader in;
        public ClientHandler(Socket socket) {
            this.socket = socket;
        }
        public void run() {
            try {
                in = new BufferedReader(new
```

```
InputStreamReader(socket.getInputStream()));

out = new PrintWriter(socket.getOutputStream(), true);
synchronized (clientWriters) {
    clientWriters.add(out);

}

String message;
while ((message = in.readLine()) != null) {
    System.out.println("Received: " + message);
    synchronized (clientWriters) {
        for (PrintWriter writer : clientWriters) {
            writer.println(message);
        }
    }
}
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

synchronized (clientWriters) {
    clientWriters.remove(out);
}

}

}
```

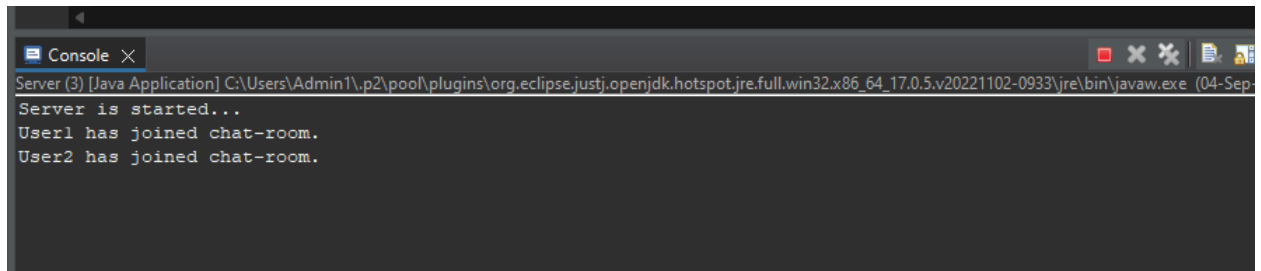
```
}  
}
```

ChatClient.java:

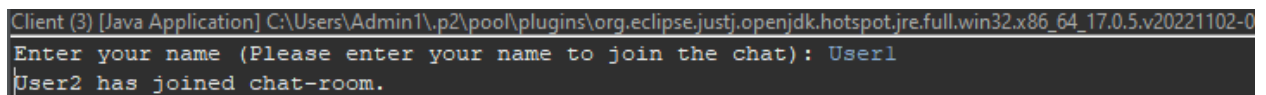
```
import java.io.*;  
import java.net.*;  
import java.util.Scanner;  
public class ChatClient {  
    private Socket socket;  
    private PrintWriter out;  
    private BufferedReader in;  
    public ChatClient(String serverAddress, int serverPort) {  
        try {  
            socket = new Socket(serverAddress, serverPort);  
            out = new PrintWriter(socket.getOutputStream(), true);  
            in = new BufferedReader(new  
                InputStreamReader(socket.getInputStream()));  
  
            new Thread(new IncomingMessageHandler()).start();  
            Scanner scanner = new Scanner(System.in);  
            String message;  
            while (scanner.hasNextLine()) {  
                message = scanner.nextLine();  
                out.println(message);  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
    } finally {  
        try {  
            socket.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
private class IncomingMessageHandler implements Runnable {  
    public void run() {  
        String message;  
        try {  
            while ((message = in.readLine()) != null) {  
                System.out.println("Server: " + message);  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
public static void main(String[] args) {  
    new ChatClient("localhost", 12345);  
}
```

Output:

ServerA screenshot of a Java console window titled "Console". The window shows the following output:

```
Server (3) [Java Application] C:\Users\Admin1\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (04-Sep-2022 10:00:00 AM)  
Server is started...  
User1 has joined chat-room.  
User2 has joined chat-room.
```

Client:A screenshot of a Java console window titled "Client (3) [Java Application]". The window shows the following output:

```
Client (3) [Java Application] C:\Users\Admin1\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (04-Sep-2022 10:00:00 AM)  
Enter your name (Please enter your name to join the chat): User1  
User2 has joined chat-room.
```