1. **WELL-COMMENTED JAVA SOURCE CODE**

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

// ChatServer class
class ChatServer {
    private static final int PORT = 12345;
    private static int clientIdCounter = 1;
    private static final Map<Integer, ClientHandler> clients = new
ConcurrentHashMap<>();

    public static void main(String[] args) {
        System.out.println("Chat server started...");
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            while (true) {
                Socket clientSocket = serverSocket.accept();
                int clientId = clientIdCounter++;
                ClientHandler handler = new ClientHandler(clientSocket, clientId);
                clients.put(clientId, handler);
                new Thread(handler).start();
                broadcast("User " + clientId + " has joined the chat.");
            }
        } catch (IOException e) {
            System.out.println("Server error: " + e.getMessage());
        }
    }

    // Broadcast message to all connected clients
    public static void broadcast(String message) {
        for (ClientHandler client : clients.values()) {
            client.sendMessage(message);
        }
```

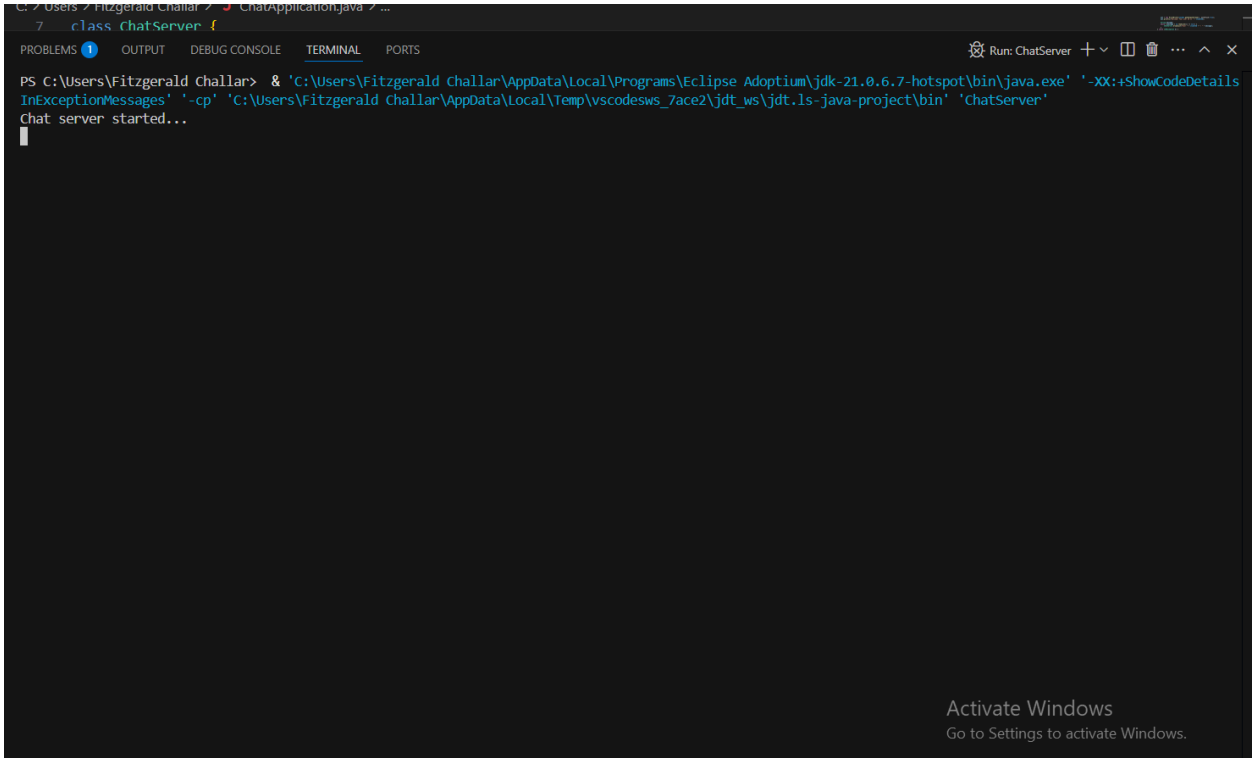
```
    }

    // Remove client when they disconnect
    public static void removeClient(int clientId) {
        clients.remove(clientId);
        broadcast("User " + clientId + " has left the chat.");
    }

    // Inner class for handling each client
    static class ClientHandler implements Runnable {
        private final Socket socket;
        private final int clientId;
        private PrintWriter out;

        public ClientHandler(Socket socket, int clientId) {
            this.socket = socket;
            this.clientId = clientId;
        }

        @Override
        public void run() {
            try (
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            ) {
                out = new PrintWriter(socket.getOutputStream(), true);
                out.println("Welcome! Your user ID is " + clientId);

                String message;
                while ((message = in.readLine()) != null) {
                    ChatServer.broadcast("User " + clientId + ": " + message);
                }
            } catch (IOException e) {
                System.out.println("User " + clientId + " disconnected.");
```

```
        } finally {
            try {
                socket.close();
            } catch (IOException ignored) {}
            ChatServer.removeClient(clientId);
        }
    }

    public void sendMessage(String message) {
        if (out != null) {
            out.println(message);
        }
    }
}

// ChatClient class
class ChatClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int PORT = 12345;

    public static void main(String[] args) {
        try (
            Socket socket = new Socket(SERVER_ADDRESS, PORT);
            BufferedReader serverIn = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter serverOut = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader userIn = new BufferedReader(new
InputStreamReader(System.in));
        ) {
            // Thread to read messages from server
            new Thread(() -> {
                String response;
                try {
```

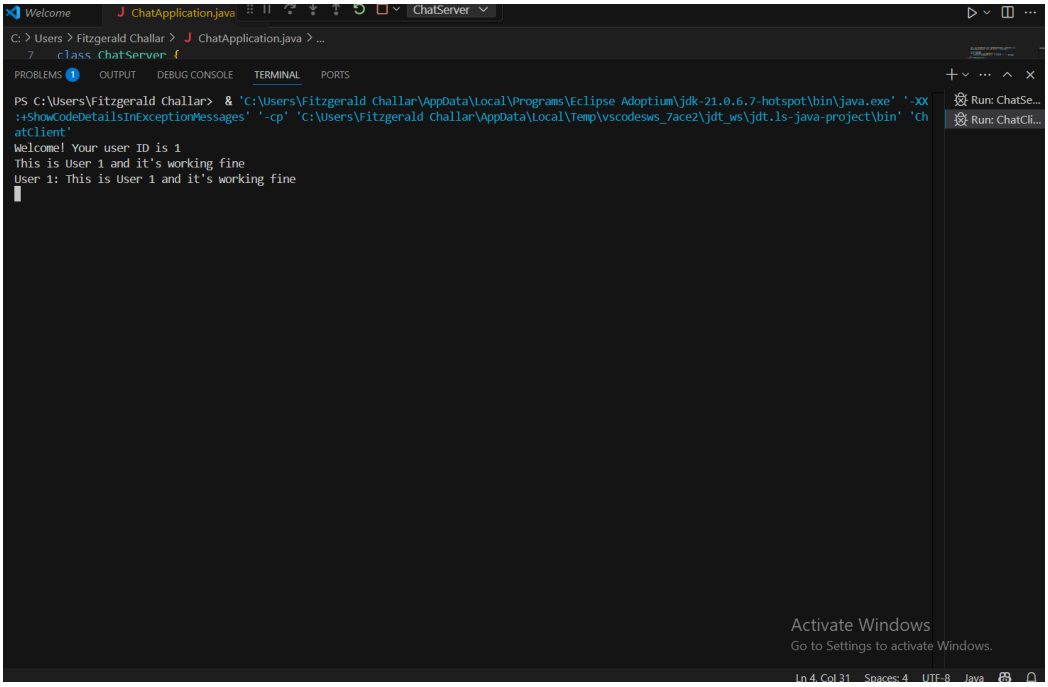
```
                while ((response = serverIn.readLine()) != null) {
                    System.out.println(response);
                }
            } catch (IOException e) {
                System.out.println("Disconnected from server.");
            }
        }).start();

        // Main thread reads user input and sends to server
        String input;
        while ((input = userIn.readLine()) != null) {
            serverOut.println(input);
        }
    } catch (IOException e) {
        System.out.println("Client error: " + e.getMessage());
    }
  }
}
```

2. SCREENSHOT OF TEXT-BASED USER INTERFACE

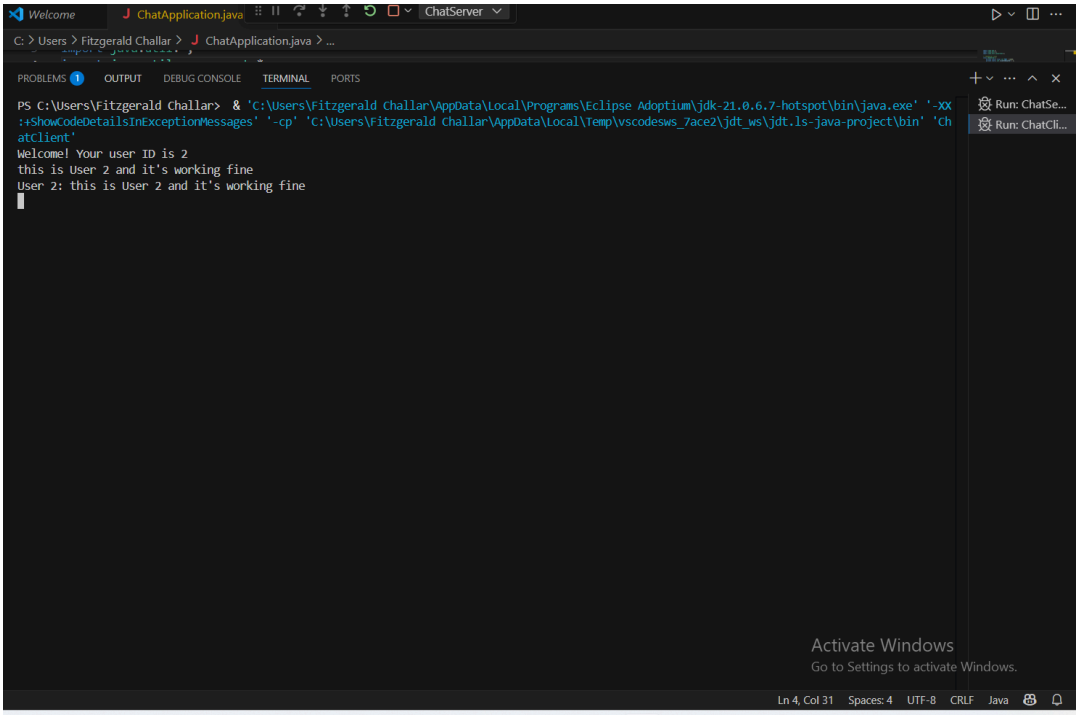
➔ **Screenshot of running server (ChatServer Class):**



➔ **Screenshot of Client 1 (User 1 from ChatClient class)**



➔ Screenshot of Client 2 (User 2 from ChatClient class)



3. **README FILE**

README Instructions

How to Compile and Run

1. Compile

javac ChatApplication.java

2. Run the Server

java ChatServer

3. Run Clients in Separate Terminals

java ChatClient

Usage:

- **Each client gets a unique user ID.**
- **Type messages in the terminal and press Enter to send.**
- **Messages are broadcasted to all connected clients.**