**18ISE01 – NETWORK PROTOCOLS**

**SEMESTER:** V **III M.Sc Software Systems** **DATE: 12-11-2020**

**PROJECT TITLE:** CHAT CONSOLE APPLICATION

**TEAM MEMBERS:**

KARTHI S – 18ISR019

SANKARANANTH S – 18ISR042

**PROGRAMS:**

1. **Chat Server Program:**
2. **ChatServer class:**

package net.codejava.networking.chat.server;

import java.io.\*;

import java.net.\*;

import java.util.\*;

/\*\*

 \* This is the chat server program.

 \* Press Ctrl + C to terminate the program.

 \*/

public class ChatServer {

    private int port;

    private Set<String> userNames = new HashSet<>();

    private Set<UserThread> userThreads = new HashSet<>();

    public ChatServer(int port) {

        this.port = port;

    }

    public void execute() {

        try (ServerSocket serverSocket = new ServerSocket(port)) {

            System.out.println("Chat Server is listening on port " + port);

            while (true) {

                Socket socket = serverSocket.accept();

                System.out.println("New user connected");

                UserThread newUser = new UserThread(socket, this);

                userThreads.add(newUser);

                newUser.start();

            }

        } catch (IOException ex) {

            System.out.println("Error in the server: " + ex.getMessage());

            ex.printStackTrace();

        }

    }

    public static void main(String[] args) {

        if (args.length < 1) {

            System.out.println("Syntax: java ChatServer <port-number>");

            System.exit(0);

        }

        int port = Integer.parseInt(args[0]);

        ChatServer server = new ChatServer(port);

        server.execute();

    }

    /\*\*

     \* Delivers a message from one user to others (broadcasting)

     \*/

    void broadcast(String message, UserThread excludeUser) {

        for (UserThread aUser : userThreads) {

            if (aUser != excludeUser) {

                aUser.sendMessage(message);

            }

        }

    }

    /\*\*

     \* Stores username of the newly connected client.

     \*/

    void addUserName(String userName) {

        userNames.add(userName);

    }

    /\*\*

     \* When a client is disconneted, removes the associated username and UserThread

     \*/

    void removeUser(String userName, UserThread aUser) {

        boolean removed = userNames.remove(userName);

        if (removed) {

            userThreads.remove(aUser);

            System.out.println("The user " + userName + " quitted");

        }

    }

    Set<String> getUserNames() {

        return this.userNames;

    }

    /\*\*

     \* Returns true if there are other users connected (not count the currently connected user)

     \*/

    boolean hasUsers() {

        return !this.userNames.isEmpty();

    }

}

1. **UserThread class:**

package net.codejava.networking.chat.server;

import java.io.\*;

import java.net.\*;

import java.util.\*;

/\*\*

 \* This thread handles connection for each connected client, so the server

 \* can handle multiple clients at the same time.

 \*/

public class UserThread extends Thread {

    private Socket socket;

    private ChatServer server;

    private PrintWriter writer;

    public UserThread(Socket socket, ChatServer server) {

        this.socket = socket;

        this.server = server;

    }

    public void run() {

        try {

            InputStream input = socket.getInputStream();

            BufferedReader reader = new BufferedReader(new InputStreamReader(input));

            OutputStream output = socket.getOutputStream();

            writer = new PrintWriter(output, true);

            printUsers();

            String userName = reader.readLine();

            server.addUserName(userName);

            String serverMessage = "New user connected: " + userName;

            server.broadcast(serverMessage, this);

            String clientMessage;

            do {

                clientMessage = reader.readLine();

                serverMessage = "[" + userName + "]: " + clientMessage;

                server.broadcast(serverMessage, this);

            } while (!clientMessage.equals("bye"));

            server.removeUser(userName, this);

            socket.close();

            serverMessage = userName + " has quitted.";

            server.broadcast(serverMessage, this);

        } catch (IOException ex) {

            System.out.println("Error in UserThread: " + ex.getMessage());

            ex.printStackTrace();

        }

    }

    /\*\*

     \* Sends a list of online users to the newly connected user.

     \*/

    void printUsers() {

        if (server.hasUsers()) {

            writer.println("Connected users: " + server.getUserNames());

        } else {

            writer.println("No other users connected");

        }

    }

    /\*\*

     \* Sends a message to the client.

     \*/

    void sendMessage(String message) {

        writer.println(message);

    }

}

1. **Chat Client Program:**
2. **ChatClient class:**

package net.codejava.networking.chat.client;

import java.net.\*;

import java.io.\*;

/\*\*

 \* This is the chat client program.

 \* Type 'bye' to terminte the program.

 \*/

public class ChatClient {

    private String hostname;

    private int port;

    private String userName;

    public ChatClient(String hostname, int port) {

        this.hostname = hostname;

        this.port = port;

    }

    public void execute() {

        try {

            Socket socket = new Socket(hostname, port);

            System.out.println("Connected to the chat server");

            new ReadThread(socket, this).start();

            new WriteThread(socket, this).start();

        } catch (UnknownHostException ex) {

            System.out.println("Server not found: " + ex.getMessage());

        } catch (IOException ex) {

            System.out.println("I/O Error: " + ex.getMessage());

        }

    }

    void setUserName(String userName) {

        this.userName = userName;

    }

    String getUserName() {

        return this.userName;

    }

    public static void main(String[] args) {

        if (args.length < 2) return;

        String hostname = args[0];

        int port = Integer.parseInt(args[1]);

        ChatClient client = new ChatClient(hostname, port);

        client.execute();

    }

}

1. **ReadThread class:**

package net.codejava.networking.chat.client;

import java.io.\*;

import java.net.\*;

/\*\*

 \* This thread is responsible for reading server's input and printing it

 \* to the console.

 \* It runs in an infinite loop until the client disconnects from the server.

 \*/

public class ReadThread extends Thread {

    private BufferedReader reader;

    private Socket socket;

    private ChatClient client;

    public ReadThread(Socket socket, ChatClient client) {

        this.socket = socket;

        this.client = client;

        try {

            InputStream input = socket.getInputStream();

            reader = new BufferedReader(new InputStreamReader(input));

        } catch (IOException ex) {

            System.out.println("Error getting input stream: " + ex.getMessage());

            ex.printStackTrace();

        }

    }

    public void run() {

        while (true) {

            try {

                String response = reader.readLine();

                System.out.println("\n" + response);

                // prints the username after displaying the server's message

                if (client.getUserName() != null) {

                    System.out.print("[" + client.getUserName() + "]: ");

                }

            } catch (IOException ex) {

                System.out.println("Error reading from server: " + ex.getMessage());

                ex.printStackTrace();

                break;

            }

        }

    }

}

1. **WriteThread class:**

package net.codejava.networking.chat.client;

import java.io.\*;

import java.net.\*;

/\*\*

 \* This thread is responsible for reading user's input and send it

 \* to the server.

 \* It runs in an infinite loop until the user types 'bye' to quit.

 \*/

public class WriteThread extends Thread {

    private PrintWriter writer;

    private Socket socket;

    private ChatClient client;

    public WriteThread(Socket socket, ChatClient client) {

        this.socket = socket;

        this.client = client;

        try {

            OutputStream output = socket.getOutputStream();

            writer = new PrintWriter(output, true);

        } catch (IOException ex) {

            System.out.println("Error getting output stream: " + ex.getMessage());

            ex.printStackTrace();

        }

    }

    public void run() {

        Console console = System.console();

        String userName = console.readLine("\nEnter your name: ");

        client.setUserName(userName);

        writer.println(userName);

        String text;

        do {

            text = console.readLine("[" + userName + "]: ");

            writer.println(text);

        } while (!text.equals("bye"));

        try {

            socket.close();

        } catch (IOException ex) {

            System.out.println("Error writing to server: " + ex.getMessage());

        }

    }

}

**OUTPUT:**

