VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ASSIGNMENT REPORT

LAB 1c

COMPUTER NETWORK

Instructor: PROF. NGUYEN MANH THIN

HO CHI MINH CITY, APRIL 2024

Contents

1 Objectives

- Practice with Socket programming in Java.
- Build a simple chat application using client-server model.
- Multithreaded application.

2 Content

2.1 Socket programming in Java

Exercise 1: Create a program that connects to a web server and downloads the homepage of this website to local computer.

The idea to solve the problem is that first of all, the program must be able to connect to the homepage of the website. In this exercise, I choose to connect to **mybk** website. The next step is to read the contents of the whole homepage and finally is to download those content and save it to a text file.

Here is the program code in Java:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class Get_Home_Page {
    public static void main(String[] args) {
        String url = "https://mybk.hcmut.edu.vn/my/index.action"; // test with mybk
        String filePath = "output.txt";

        try {
            // Create URL object
            URL obj = new URL(url);

            // Open connection
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();

            // Request GET method
            con.setRequestMethod("GET");

            // Get response code
            int responseCode = con.getResponseCode();
            System.out.println("Response Code: " + responseCode);

            // Read response content
            BufferedReader in = new BufferedReader(new InputStreamReader(con.
    getInputStream()));
            StringBuilder response = new StringBuilder();
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
                response.append("\n");
            }
            in.close();

            // Write content to file
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
```

```
                writer.write(response.toString());
                System.out.println("Content saved to: " + filePath);
            }

        System.out.println("Download successfully !");
          } catch (Exception e) {
              e.printStackTrace();
          }
    }
}
```

Here is the output of the program:

```
olkmphy@knammm MINGW64 /d/BKU - K21 - Computer Engineering/Computer Network/Lab/Lab 1/Pictures/Lab 1c (main)
$ java get_web_content.java
Response Code: 200
Content saved to: output.txt
Download successfully !
```

Figure 1: Response code of the program.

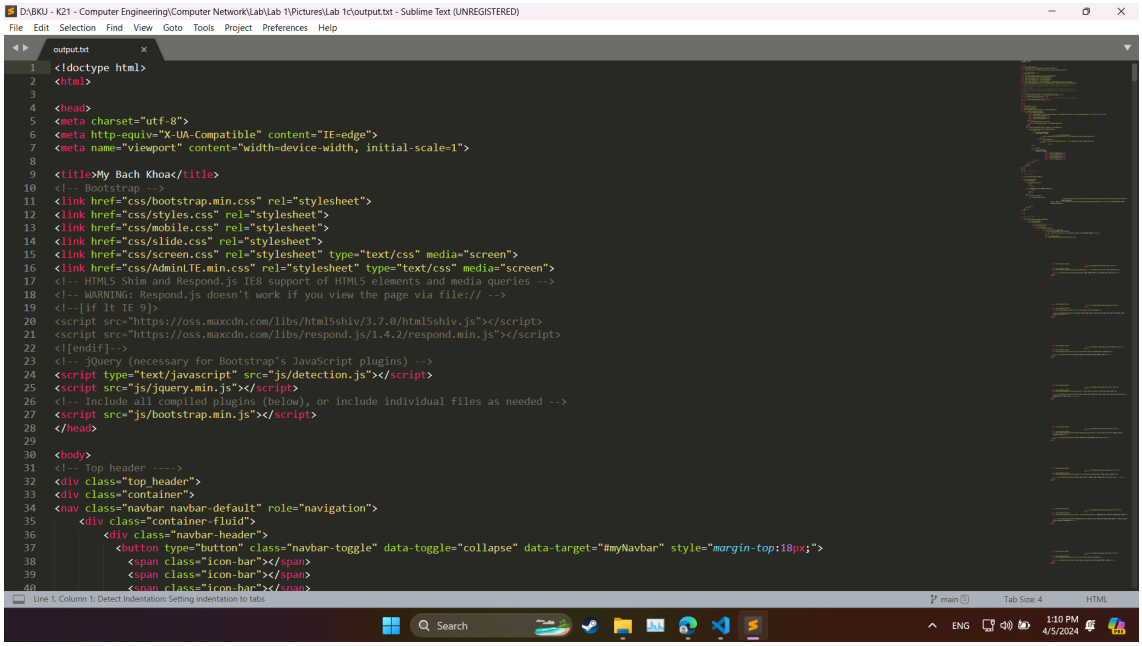

Figure 2: MyBK homepage content.

2.2 Develop a simple chat application using client-server model

Exercise 2: Design the user interface for the chat application.

In this section, my main target is to create an interface for the chat application using client-server model. This is my program's interface using Java language.

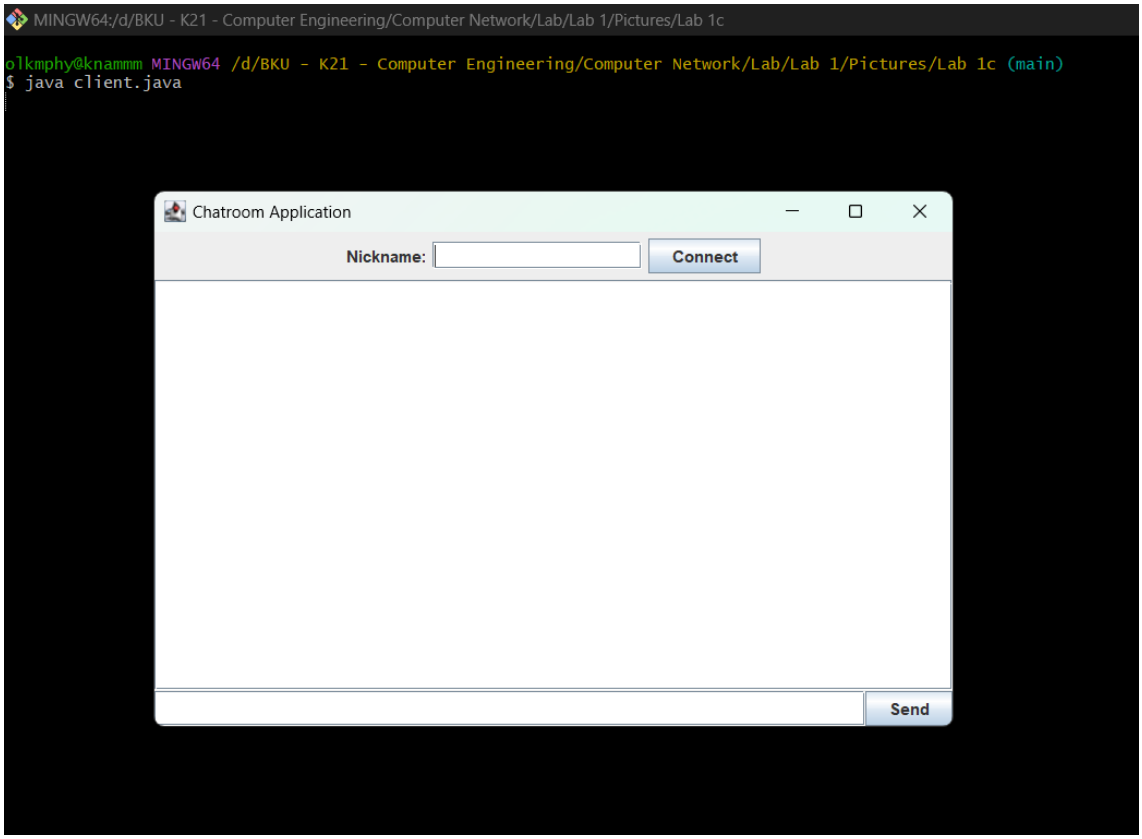

Figure 3: Application interface.

The code to create this interface is below:

```
public client() {
    setTitle("Chatroom Application");
    setSize(600, 400); // Adjusted height
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // Panel
    JPanel mainPanel = new JPanel(new BorderLayout());

    // Top panel for nickname, connect button, and quit button
    JPanel topPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
    JLabel nameLabel = new JLabel("Nickname:");
    nicknameField = new JTextField(15);

    // Connect button
    JButton connectButton = new JButton("Connect");
    connectButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            connectToServer(connectButton);
        }
    });

    topPanel.add(nameLabel);
    topPanel.add(nicknameField);
```

```
    topPanel.add(connectButton);
    mainPanel.add(topPanel, BorderLayout.NORTH);

    // Message area
    chatArea = new JTextArea();
    chatArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(chatArea);
    mainPanel.add(scrollPane, BorderLayout.CENTER);

    // Bottom panel for message input and send button
    JPanel bottomPanel = new JPanel(new BorderLayout());
    messageField = new JTextField();
    messageField.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            sendMessage();
        }
    });
    JButton sendButton = new JButton("Send");
    sendButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            sendMessage();
        }
    });
    bottomPanel.add(messageField, BorderLayout.CENTER);
    bottomPanel.add(sendButton, BorderLayout.EAST);
    mainPanel.add(bottomPanel, BorderLayout.SOUTH);

    add(mainPanel);
    setVisible(true);
}
```

In other word, the interface of the program is created within the constructor of the client class.

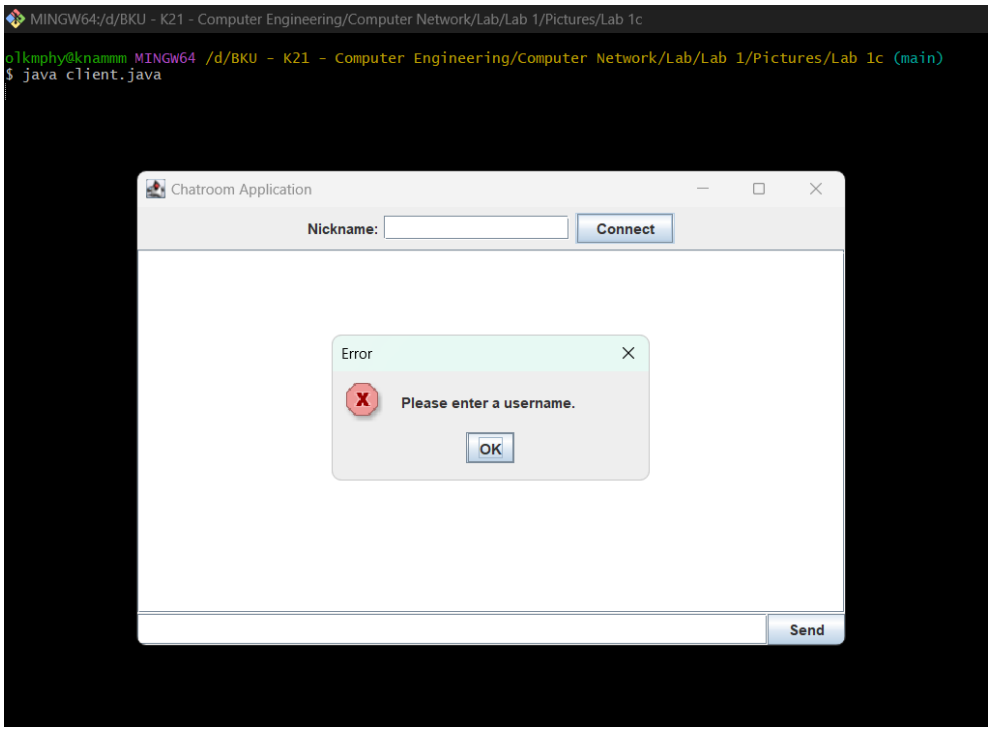

Figure 4: Handling invalid username.

2.3 Multithread in Java

Exercise 3: Using multi-thread programming model to make the chat application can talk to many different users concurrently.

In this section, I will implement the multi-thread programming model into my chatroom application so that the application can handle several users concurrently. Here is my program's functioning output

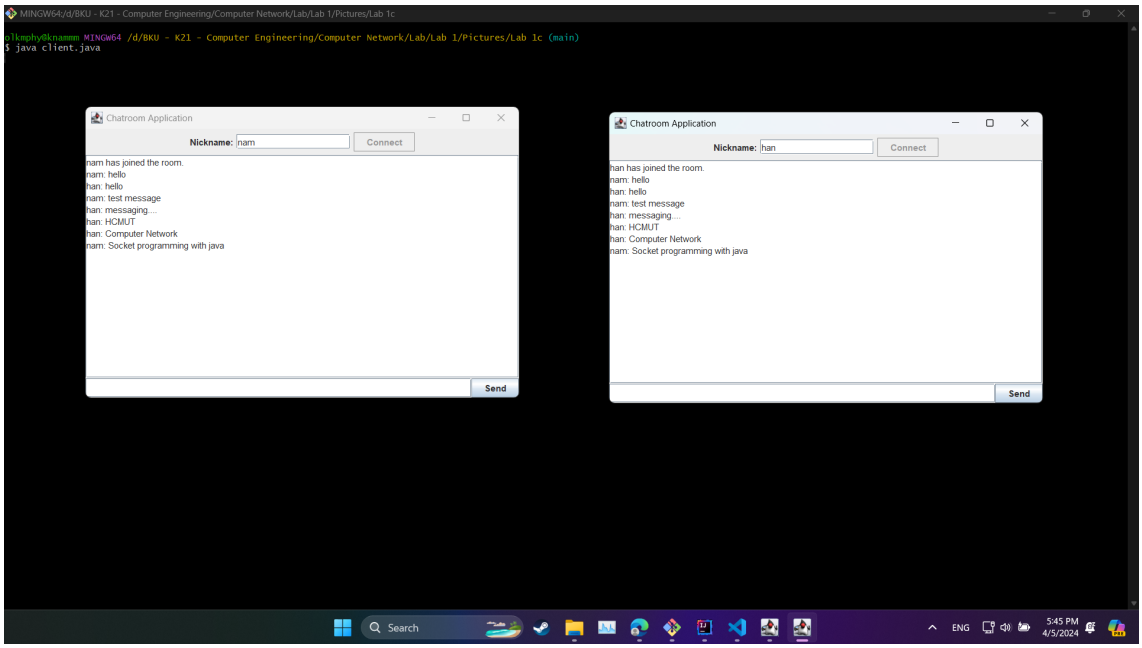

Figure 5: Chatroom Application with users.

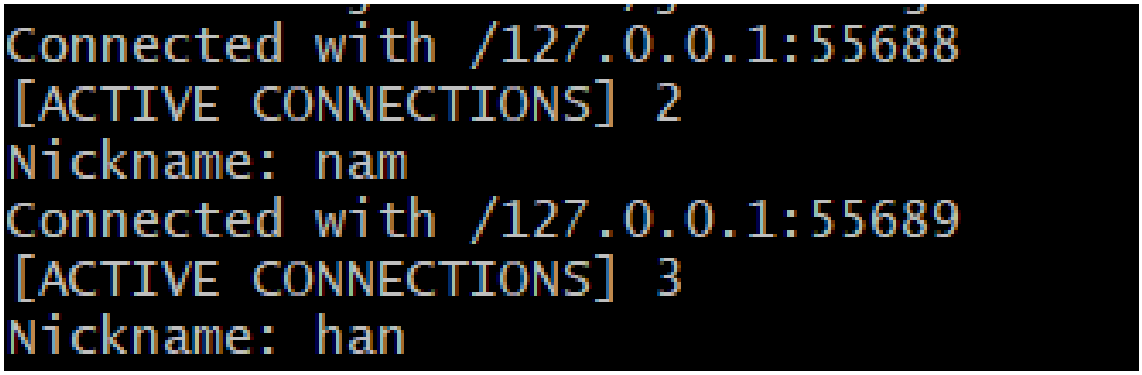

Figure 6: Server side.

Here is the source code of the **server-side**:

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

public class server {
    private static String host;
    private static final int PORT = 12345;

    private static final List<Socket> clients = new ArrayList<>();
    private static final List<String> nicknames = new ArrayList<>();

    private static void sendMessageToAllClients(String message) {
        for (Socket client : clients) {
            try {
                PrintWriter writer = new PrintWriter(client.getOutputStream(), true);
                writer.println(message); // sending message to clients
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    private static void handleClient(Socket client) {
        String nickname = null;
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(client.
    getInputStream()));
            nickname = reader.readLine();
            nicknames.add(nickname);
            clients.add(client);

            System.out.println("Nickname: " + nickname);

            while (true) {
                String message = reader.readLine();
                if (message != null) {
                    sendMessageToAllClients(message);
                }
            }
        } catch (SocketException e) {
            // Client disconnected unexpectedly
            System.out.println("Client disconnected: " + e.getMessage());
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // When client disconnects, remove from lists
            if (nickname != null) {
                int index = nicknames.indexOf(nickname);
                if (index != -1) {
                    nicknames.remove(index);
                    clients.remove(index);
                    sendMessageToAllClients(nickname + " has left the room.");
                }
            }
        }
    }

    private static void startServer() {
```

```
        try {
            ServerSocket serverSocket = new ServerSocket(PORT);
            System.out.println("Server is running on " + host + ":" + PORT);

            while (true) {
                Socket client = serverSocket.accept();
                System.out.println("Connected with " + client.getRemoteSocketAddress());

                // Multi-thread programming
                Thread thread = new Thread(() -> handleClient(client));
                thread.start();
                System.out.println("[ACTIVE CONNECTIONS] " + (Thread.activeCount() - 1));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        try {
            host = InetAddress.getLocalHost().getHostAddress();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
        startServer();
    }
}
```

Here is the source code of the **client-side**:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class ChatClient extends JFrame {
    private static final int PORT = 12345;
    private Socket socket;
    private BufferedReader reader;
    private PrintWriter writer;
    private JTextField messageInputField;
    private JTextArea chatDisplayArea;
    private JTextField nicknameInputField;

    public ChatClient() {
        setTitle("Chatroom Application");
        setSize(600, 400); // Adjusted height
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Panel
        JPanel mainPanel = new JPanel(new BorderLayout());

        // Top panel for nickname, connect button, and quit button
        JPanel topPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        JLabel nameLabel = new JLabel("Nickname:");
        nicknameInputField = new JTextField(15);

        // Connect button
        JButton connectButton = new JButton("Connect");
        connectButton.addActionListener(new ActionListener() {
```

```
            @Override
            public void actionPerformed(ActionEvent e) {
                connectToServer(connectButton);
            }
        });

        topPanel.add(nameLabel);
        topPanel.add(nicknameInputField);
        topPanel.add(connectButton);
        mainPanel.add(topPanel, BorderLayout.NORTH);

        // Message area
        chatDisplayArea = new JTextArea();
        chatDisplayArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(chatDisplayArea);
        mainPanel.add(scrollPane, BorderLayout.CENTER);

        // Bottom panel for message input and send button
        JPanel bottomPanel = new JPanel(new BorderLayout());
        messageInputField = new JTextField();
        messageInputField.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                sendMessage();
            }
        });
        JButton sendButton = new JButton("Send");
        sendButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                sendMessage();
            }
        });
        bottomPanel.add(messageInputField, BorderLayout.CENTER);
        bottomPanel.add(sendButton, BorderLayout.EAST);
        mainPanel.add(bottomPanel, BorderLayout.SOUTH);

        add(mainPanel);
        setVisible(true);
    }

    private void connectToServer(JButton connectButton) {
        // Check if the username is empty
        if (nicknameInputField.getText().isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter a username.", "Error",
    JOptionPane.ERROR_MESSAGE);
            return;
        }
        try {
            String nickname = nicknameInputField.getText();
            // Connect to server
            socket = new Socket("localhost", PORT);
            reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            writer = new PrintWriter(socket.getOutputStream(), true);
            // Send username to server
            writer.println(nickname);
            connectButton.setEnabled(false); // Don't give the user to click the connect
    button anymore
            // Display username
            chatDisplayArea.append(nickname + " has joined the room.\n");
```

```
            // Listen from server
            new Thread(new IncomingReader()).start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void sendMessage() {
        String message = messageInputField.getText();
        if (!message.isEmpty()) {
            writer.println(nicknameInputField.getText() + ": " + message);
            messageInputField.setText("");
        }
    }

    private class IncomingReader implements Runnable {
        @Override
        public void run() {
            try {
                String msg;
                while ((msg = reader.readLine()) != null) {
                    final String message = msg;
                    SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                            chatDisplayArea.append(message + "\n");
                        }
                    });
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new ChatClient();
            }
        });
    }
}
```

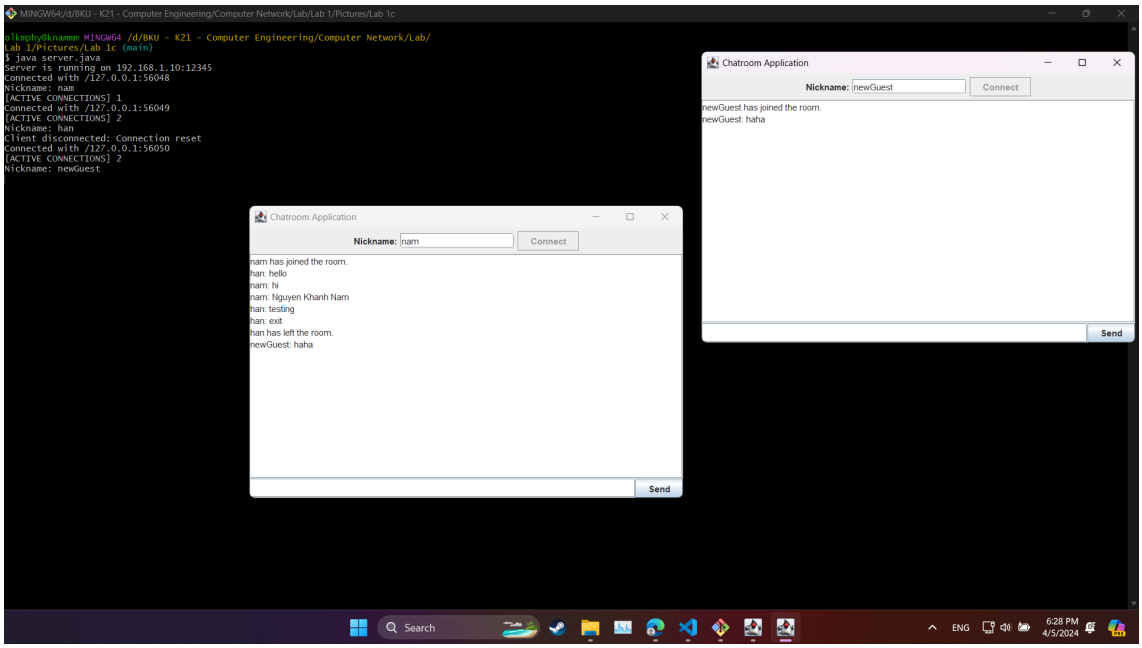

Figure 7: Full implementation.

Here is the link to the source code: GitHub