Java RMI Chat App with GUI

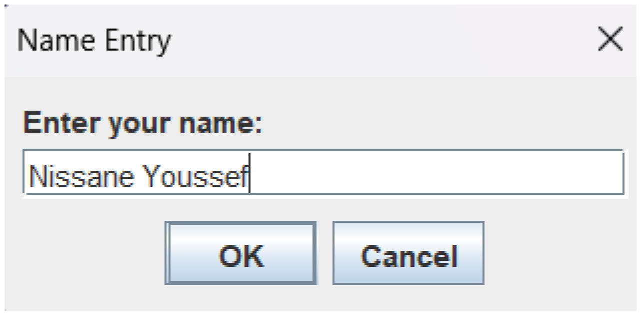
Nissane Youssef

Introduction

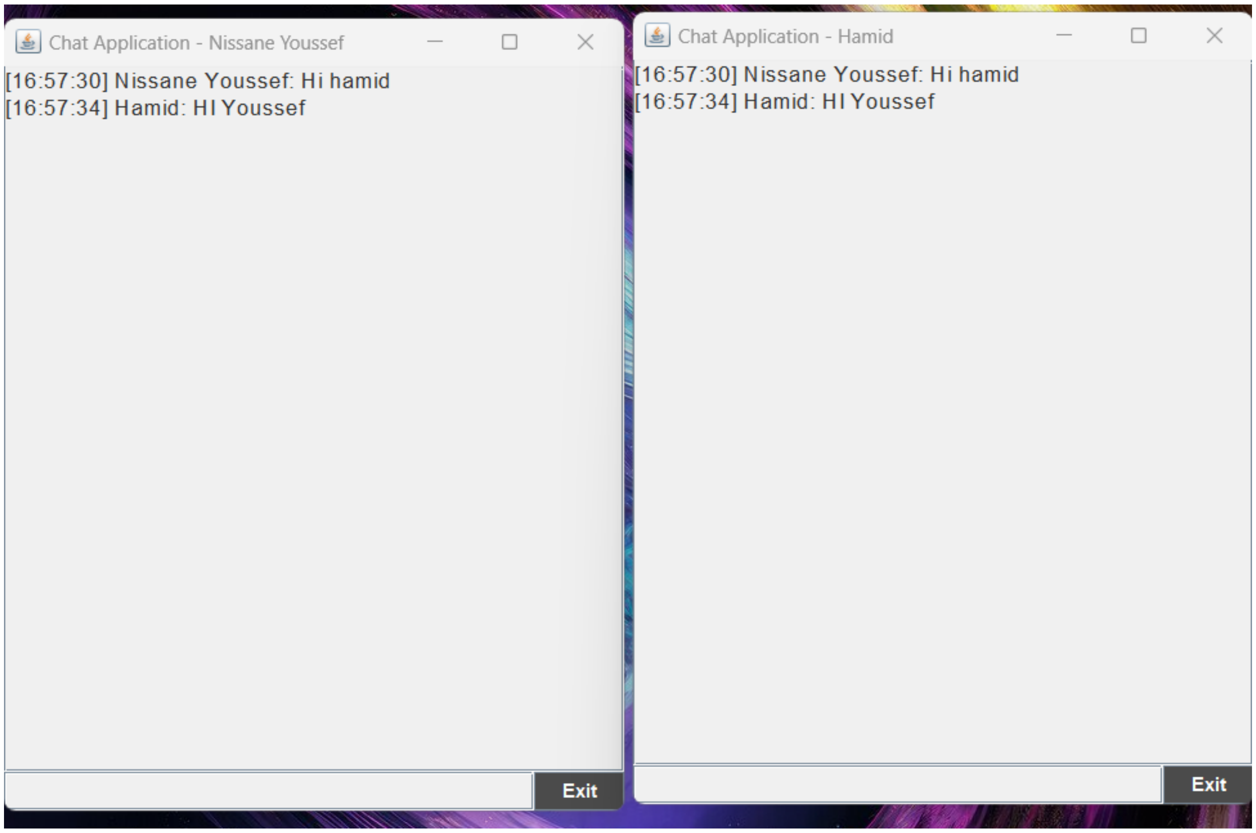
The Java RMI Chat App with GUI is a project that enables real-time communication between multiple users over a network. It combines Java's Remote Method Invocation (RMI) technology with a graphical user interface (GUI) built using Swing. This project provides a platform for users to engage in chat conversations seamlessly, offering a user-friendly interface for an enhanced chat experience.

Project Objective

The primary objective of the Java RMI Chat App with GUI is to facilitate real-time communication between multiple users over a network. By leveraging RMI technology, the project aims to provide a reliable and scalable solution for chat applications. The GUI enhances the user experience by providing intuitive controls for sending and receiving messages.

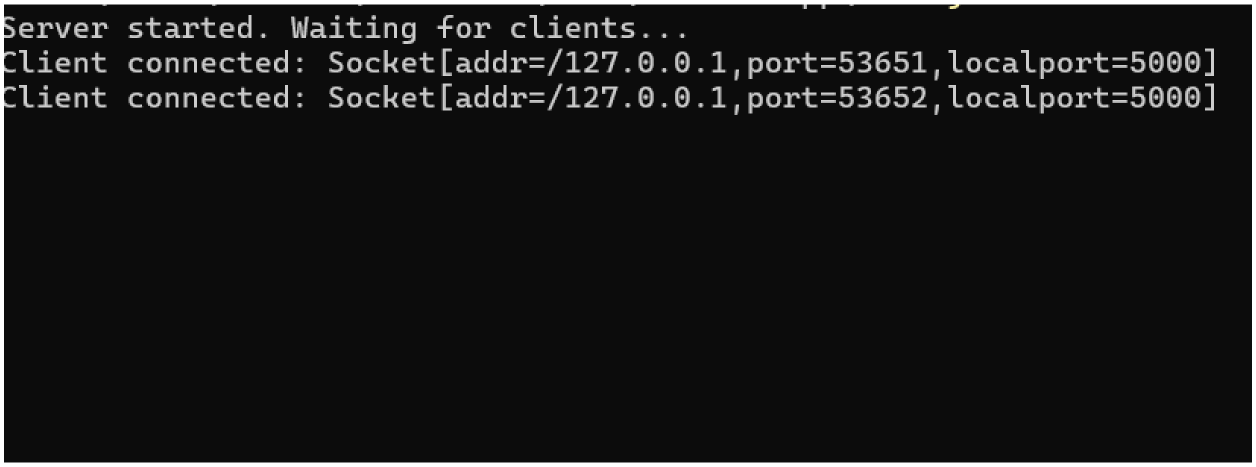


Enter your name: Nissane Youssef



```
[16:57:30] Nissane Youssef: Hi hamid
[16:57:34] Hamid: HI Youssef
```

```
[16:57:30] Nissane Youssef: Hi hamid
[16:57:34] Hamid: HI Youssef
```



```
Server started. Waiting for clients...
Client connected: Socket[addr=/127.0.0.1,port=53651,localport=5000]
Client connected: Socket[addr=/127.0.0.1,port=53652,localport=5000]
```

Stack Used

- Java RMI: For enabling communication between clients and the server.
- Swing (GUI): For creating the graphical user interface for the chat client.

How to Run the Code

To run the code, follow these steps:

Run the Server Code:

- Compile and run the `ChatServer.java` file.

  ```
  javac ChatServer.java
  java ChatServer
  ```

- **Run the Client GUI Code:**
  - Compile and run the `ChatClientGUI.java` file.

    ```
    javac ChatClientGUI.java
    java ChatClientGUI
    ```

Alternatively, you can run the code in your favorite text editor or IDE by compiling and executing the respective Java files.

Code :

ChatClientGUI.java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ChatClientGUI extends JFrame {
    private JTextArea messageArea;
    private JTextField textField;
```

```
    private JButton exitButton;
    private ChatClient client;

    public ChatClientGUI() {
        super("Chat Application"); // Set the title of the GU
I window
        setSize(400, 500); // Set the size of the GUI window
        setDefaultCloseOperation(EXIT_ON_CLOSE); // Set defau
lt close operation

        // Define colors and fonts for GUI elements
        Color backgroundColor = new Color(240, 240, 240);
        Color buttonColor = new Color(75, 75, 75);
        Color textColor = new Color(50, 50, 50);
        Font textFont = new Font("Arial", Font.PLAIN, 14);
        Font buttonFont = new Font("Arial", Font.BOLD, 12);

        // Create and configure message area for displaying c
hat messages
        messageArea = new JTextArea();
        messageArea.setEditable(false);
        messageArea.setBackground(backgroundColor);
        messageArea.setForeground(textColor);
        messageArea.setFont(textFont);
        JScrollPane scrollPane = new JScrollPane(messageAre
a);
        add(scrollPane, BorderLayout.CENTER);

        // Prompt user to enter their name
        String name = JOptionPane.showInputDialog(this, "Ente
r your name:", "Name Entry", JOptionPane.PLAIN_MESSAGE);
        this.setTitle("Chat Application - " + name); // Set w
indow title with user's name

        // Create and configure text field for typing message
s
```

```
        textField = new JTextField();
        textField.setFont(textFont);
        textField.setForeground(textColor);
        textField.setBackground(backgroundColor);
        textField.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Send message to server when Enter key is p
ressed
                String message = "[" + new SimpleDateFormat
("HH:mm:ss").format(new Date()) + "] " + name + ": "
                        + textField.getText();
                client.sendMessage(message);
                textField.setText(""); // Clear text field af
ter sending message
            }
        });

        // Create and configure exit button to gracefully exi
t the chat
        exitButton = new JButton("Exit");
        exitButton.setFont(buttonFont);
        exitButton.setBackground(buttonColor);
        exitButton.setForeground(Color.WHITE);
        exitButton.addActionListener(e -> {
            // Notify server about client departure and exit
the application
            String departureMessage = name + " has left the c
hat.";
            client.sendMessage(departureMessage);
            try {
                Thread.sleep(1000); // Wait for 1 second befo
re exiting to ensure message delivery
            } catch (InterruptedException ie) {
                Thread.currentThread().interrupt();
            }
            System.exit(0);
        });

        // Create panel for text field and exit button
        JPanel bottomPanel = new JPanel(new BorderLayout());
        bottomPanel.setBackground(backgroundColor);
        bottomPanel.add(textField, BorderLayout.CENTER);
        bottomPanel.add(exitButton, BorderLayout.EAST);
        add(bottomPanel, BorderLayout.SOUTH);

        // Initialize ChatClient instance to establish connec
tion with server
        try {
            this.client = new ChatClient("127.0.0.1", 5000, t
his::onMessageReceived);
            client.startClient(); // Start client communicati
on
        } catch (IOException e) {
            // Handle connection error and display error mess
age
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error connec
ting to the server", "Connection error",
                    JOptionPane.ERROR_MESSAGE);
            System.exit(1); // Exit application on connection
error
        }
    }

    // Callback method to handle received messages and update
GUI
    private void onMessageReceived(String message) {
        SwingUtilities.invokeLater(() -> messageArea.append(m
essage + "\\n")); // Append message to message area
    }

    // Main method to launch the application
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new ChatClientGUI().setVisible(true); // Create a
nd show GUI window
        });
    }
}
```

Explanation:

- **GUI Initialization:**
    - The `ChatClientGUI` class extends `JFrame` to create a graphical user interface for the chat client.
    - It sets the title, size, and default close operation for the GUI window.
- **GUI Components:**
    - It creates a text area (`messageArea`) for displaying chat messages, a text field (`textField`) for typing messages, and an exit button (`exitButton`) to gracefully exit the chat.
- **User Input:**
    - It prompts the user to enter their name using a dialog box.
    - The user's name is used to customize the window title and identify messages sent by the user.
- **Event Handling:**
    - It attaches an action listener to the text field to send messages to the server when the Enter key is pressed.
    - It attaches an action listener to the exit button to send a departure message to the server and exit the application.
- **Message Display:**
    - It provides a callback method (`onMessageReceived`) to handle received messages from the server and update the message area.

- **Client Initialization:**
  - It initializes a `ChatClient` instance to establish a connection with the server.
  - The `ChatClient` instance is started to initiate client-server communication.
- **Main Method:**
  - The `main` method creates and displays an instance of `ChatClientGUI` using the event dispatch thread (EDT) to ensure thread safety.

Overall, the `ChatClientGUI` class creates a user-friendly interface for the chat client, allowing users to send and receive messages while interacting with the server in real-time.

ChatServer.java:

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

public class ChatServer {
    private static List<ClientHandler> clients = new ArrayLis
t<>();

    public static void main(String[] args) throws IOException
{
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server started. Waiting for clien
ts...");

        while (true) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientS
ocket);
```

```
            ClientHandler clientThread = new ClientHandler(cl
ientSocket, clients);
            clients.add(clientThread);
            new Thread(clientThread).start();
        }
    }
}

class ClientHandler implements Runnable {
    private Socket clientSocket;
    private List<ClientHandler> clients;
    private PrintWriter out;
    private BufferedReader in;

    public ClientHandler(Socket socket, List<ClientHandler> c
lients) throws IOException {
        this.clientSocket = socket;
        this.clients = clients;
        this.out = new PrintWriter(clientSocket.getOutputStre
am(), true);
        this.in = new BufferedReader(new InputStreamReader(cl
ientSocket.getInputStream()));
    }

    public void run() {
        try {
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                for (ClientHandler aClient : clients) {
                    aClient.out.println(inputLine);
                }
            }
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getM
essage());
        } finally {
            try {
                in.close();
                out.close();
                clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Explanation:

- **ChatServer.java:**
    - The `ChatServer` class listens for incoming client connections on port 5000.
    - When a client connects, it creates a new `ClientHandler` object to handle communication with that client.
    - Each `ClientHandler` object is started in a new thread to handle multiple clients concurrently.
    - The server continuously listens for incoming connections in a loop.
- **ClientHandler.java:**
    - The `ClientHandler` class manages communication with an individual client.
    - It initializes input and output streams for the client's socket connection.
    - The `run` method of `ClientHandler` continuously reads messages from the client and broadcasts them to all connected clients.
    - If an error occurs during communication, it is caught and handled, ensuring the graceful termination of the client's connection.

This setup enables the server to handle multiple client connections simultaneously, allowing for real-time communication between clients in a chat-like environment. Each client's messages are broadcasted to all other connected clients.