

**应用基础实践（一）网络与JAVA**

**课程设计实验报告**

|  |  |
| --- | --- |
| 学生姓名 | **胡琛** |
| 学 号 | **L208220432** |
| 专业班级 | **计算机与技术** |
| 指导教师 | **刘嫔** |
| 学 院 | **计算机学院** |
| 完成时间 | **2024-07-10** |

计算机学院

2024年7月

Table of Contents

[**（一）** **用JavaSocket编程开发聊天室** 3](#_Toc171561836)

[**1.系统描述** 3](#_Toc171561837)

[**2.基本要求** 3](#_Toc171561838)

[**2.1 系统内容** 3](#_Toc171561839)

[**2.1. 服务器模块：** 3](#_Toc171561840)

[**2.2 客户端模块：** 3](#_Toc171561841)

[**2.3具体功能描述** 3](#_Toc171561842)

[**2.4. 服务器端功能：** 3](#_Toc171561843)

[**2.5 客户端功能：** 3](#_Toc171561844)

[**3.功能模块结构** 4](#_Toc171561845)

[**3.1服务器模块：** 4](#_Toc171561846)

[**3.2客户端模块：** 4](#_Toc171561847)

[**3.3结构图** 4](#_Toc171561848)

[**3.4流程图：** 5](#_Toc171561849)

[**3.5各模块的功能描述** 6](#_Toc171561850)

[**4.详细功能描述** 6](#_Toc171561851)

[**4.1ChatServer.java** 6](#_Toc171561852)

[**4.2客户端：ChatClient.java** 9](#_Toc171561853)

[**4,3主程序模块（Main）** 14](#_Toc171561854)

[**5..主要模块的算法说明** 14](#_Toc171561855)

[**5.1服务器端：ChatServer.java** 14](#_Toc171561856)

[**5.2客户端：ChatClient.java** 17](#_Toc171561857)

[**5.3 主程序模块（Main）** 21](#_Toc171561858)

[**6.** **运行结果：** 22](#_Toc171561859)

[6.1运行 ChatServer 22](#_Toc171561860)

[6.2运行 ChatClient 23](#_Toc171561861)

[**7.运行结果总结** 27](#_Toc171561862)

[**8.课程设计总结** 28](#_Toc171561863)

[**8.1遇到的问题** 28](#_Toc171561864)

[**8.2解决问题的过程** 28](#_Toc171561865)

[**8.3体会与收获** 32](#_Toc171561866)

[**8.4思考** 32](#_Toc171561867)

[**9.参考文献** 32](#_Toc171561868)

[**附录代码：主要程序清单** 33](#_Toc171561869)

[**10.1 ChatServer.java** 33](#_Toc171561870)

[**10.2ChatClient.java:** 36](#_Toc171561871)

[10.3Main.java 40](#_Toc171561872)

1. **用JavaSocket编程开发聊天室**

# **1.系统描述**

本系统是一个基于Java Socket编程的简易聊天室，旨在实现多用户实时通信的基本功能。系统通过Socket连接来实现客户端与服务器之间的消息传递，支持公聊和私聊功能，并提供用户界面（GUI）以提高用户体验。以下是系统的基本要求和内容：

# **2.基本要求**

1. 多用户支持：允许多个用户同时连接到服务器，并能够相互通信。

2. 消息广播：用户发送的消息可以广播给所有连接的用户，确保消息的实时性。

3. 私聊功能：用户可以选择向特定用户发送私聊消息，确保信息的私密性。

4. 用户管理：服务器能够管理连接的用户，包括显示在线用户列表、处理用户的连接和断开。

5. 图形用户界面（GUI）：客户端提供一个简单直观的图形界面，用户可以方便地输入和查看消息。

6. 用户输入处理：客户端能够处理用户的输入，包括发送消息、发送私聊消息、显示系统消息等。

## **2.1 系统内容**

## **2.1. 服务器模块：**

- 监听客户端的连接请求。

- 管理已连接的用户。

- 接收并广播用户消息。

- 转发私聊消息给指定用户。

- 在用户连接和断开时更新在线用户列表。

## **2.2 客户端模块：**

- 连接到服务器。

- 提供图形用户界面，用户可以通过界面发送和接收消息。

- 处理用户输入，包括发送公聊和私聊消息。

- 接收服务器广播的消息并显示在界面上。

- 显示系统通知，例如用户加入或离开。

## **2.3具体功能描述**

## **2.4. 服务器端功能：**

- 监听特定端口，等待客户端连接。

- 为每个连接的客户端分配一个线程处理通信。

- 接收客户端消息并区分是公聊还是私聊。

- 将公聊消息广播给所有在线用户。

- 将私聊消息转发给指定用户。

- 管理用户连接和断开事件，更新用户列表。

## **2.5 客户端功能：**

- 连接到服务器并进行身份验证（输入用户名）。

- 发送公聊消息到服务器。

- 发送私聊消息到服务器，指定目标用户。

- 接收并显示来自服务器的公聊和私聊消息。

- 显示用户连接和断开的系统通知。

# **3.功能模块结构**

**功能模块划分**

本系统主要分为两个核心模块：客户端模块和服务器模块。每个模块进一步细分为若干子模块，以实现各自的功能。

## **3.1服务器模块：**

-连接管理模块：负责监听客户端连接请求。

-用户管理模块：负责管理已连接用户，包括用户的添加和移除。

-消息处理模块：负责接收并处理客户端发送的消息，包括广播消息和私聊消息。

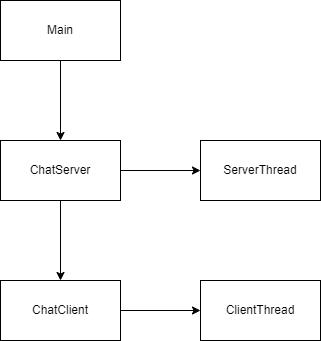
## **3.2客户端模块：**

-连接模块：负责与服务器建立连接。

-用户界面模块：提供图形用户界面供用户输入和查看消息。

-消息处理模块：处理用户输入的消息并发送给服务器，同时接收服务器发送的消息并显示。

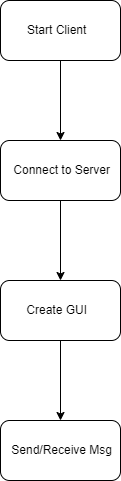
## **3.3结构图**

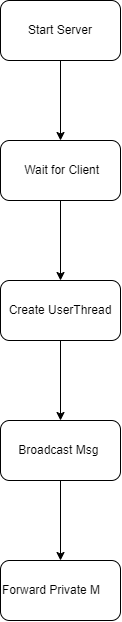


## **3.4流程图：**

**1.服务器处理流程图**

**2.客户端连接流程图：**



****

## **3.5各模块的功能描述**

**1.服务器模块：**

**连接管理模块：**

**功能：**监听客户端的连接请求。

**实现：**使用ServerSocket类监听特定端口，接受连接请求并创建用户线程。

**用户管理模块：**

**功能**：管理已连接的用户，维护用户列表。

**实现：**使用线程安全的集合来存储和管理用户线程。

**消息处理模块：**

**功能**：接收并处理客户端发送的消息。

**实现**：解析消息类型（公聊或私聊），调用相应的方法进行处理和转发。

**2.客户端模块：**

**连接模块：**

功能：与服务器建立连接。

实现：使用Socket类连接服务器，获取输入输出流。

**用户界面模块：**

**功能**：提供图形用户界面，供用户输入和查看消息。

**实现**：使用Swing组件创建窗口、文本区域、输入框、按钮等。

**消息处理模块：**

**功能**：处理用户输入的消息，发送给服务器，并接收服务器发送的消息显示在界面上。

**实现**：使用PrintWriter和BufferedReader处理消息的发送和接收，通过新线程监听服务器消息。

# **4.详细功能描述**

## **4.1ChatServer.java**

* 设置服务器端口和用户线程集合
* 定义服务器端口号（例如12345）。
* 使用线程安全的集合（Collections.synchronizedSet）来管理用户线程。

private static final int PORT = 12345;

private static Set<UserThread> userThreads = Collections.synchronizedSet(new HashSet<>());

* 启动服务器
* 创建ServerSocket对象来监听指定端口。
* 使用while循环持续等待客户端连接。

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, userThreads);

        userThreads.add(newUser);

        newUser.start();

    }

} catch (IOException ex) {

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

    ex.printStackTrace();

}

* 管理用户线程
* UserThread类负责处理每个用户的连接。
* 在run方法中，读取用户消息并进行相应处理，包括广播公聊消息和转发私聊消息。

class UserThread extends Thread {

    private Socket socket;

    private PrintWriter writer;

    private static Set<UserThread> userThreads;

    private String userName;

    public UserThread(Socket socket, Set<UserThread> userThreads) {

        this.socket = socket;

        UserThread.userThreads = userThreads;

    }

    public void run() {

        try {

            InputStream input = socket.getInputStream();

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

            OutputStream output = socket.getOutputStream();

            writer = new PrintWriter(output, true);

            userName = reader.readLine();

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

            broadcast(serverMessage, this);

            printUsers(); // 向新用户发送用户列表

            String clientMessage;

            do {

                clientMessage = reader.readLine();

                if (clientMessage != null) {

                    System.out.println("Received message: " + clientMessage); // 调试信息

                    if (clientMessage.startsWith("/w ")) {

                        String[] messageParts = clientMessage.split(" ", 3);

                        if (messageParts.length == 3) {

                            String recipient = messageParts[1];

                            String privateMessage = messageParts[2];

                            if (!ChatServer.sendPrivateMessage(recipient, privateMessage, userName)) {

                                writer.println("User " + recipient + " not found.");

                            }

                        }

                    } else {

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

                        broadcast(serverMessage, this);

                    }

                }

            } while (clientMessage != null && !clientMessage.equalsIgnoreCase("bye"));

            removeUser(this);

            socket.close();

            serverMessage = userName + " has quitted.";

            broadcast(serverMessage, this);

        } catch (IOException ex) {

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

            ex.printStackTrace();

        }

    }

    // 其他方法：printUsers、broadcast、removeUser、sendPrivateMessage

}

* 广播和私聊消息
* broadcast方法用于将公聊消息发送给所有用户。
* sendPrivateMessage方法用于发送私聊消息给特定用户。

void broadcast(String message, UserThread excludeUser) {

    for (UserThread aUser : userThreads) {

        if (aUser != excludeUser) {

            aUser.writer.println(message);

        }

    }

}

static boolean sendPrivateMessage(String recipient, String message, String sender) {

    for (UserThread aUser : userThreads) {

        if (aUser.getUserName().equals(recipient)) {

            aUser.sendPrivateMessage(sender, message);

            return true;

        }

    }

    return false;

}

## **4.2客户端：ChatClient.java**

* 1：设置服务器地址和端口
* 定义服务器的IP地址和端口号。

private String hostname;

private int port;

public ChatClient(String hostname, int port) {

    this.hostname = hostname;

    this.port = port;

}

* 2.连接服务器
* 使用Socket类连接服务器，获取输入输出流。
* 在连接后创建用户界面。

public void execute() {

    try {

        socket = new Socket(hostname, port);

        InputStream input = socket.getInputStream();

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

        OutputStream output = socket.getOutputStream();

        writer = new PrintWriter(output, true); // 自动刷新

        createGUI();

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

    } catch (IOException ex) {

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

        ex.printStackTrace();

    }

}

* 3：创建用户界面
* 使用Swing组件创建窗口、文本区域、输入框、按钮等。
* 添加事件监听器处理用户输入和按钮点击事件。

private void createGUI() {

    frame = new JFrame("Chat Client");

    frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    frame.setSize(400, 400);

    JPanel panel = new JPanel();

    panel.setLayout(new BorderLayout());

    textArea = new JTextArea();

    textArea.setEditable(false);

    panel.add(new JScrollPane(textArea), BorderLayout.CENTER);

    JPanel inputPanel = new JPanel();

    inputPanel.setLayout(new BorderLayout());

    textField = new JTextField();

    textField.setEditable(true);

    textField.setColumns(25);

    textField.setForeground(Color.BLACK);

    textField.setBackground(Color.LIGHT\_GRAY);

    textField.setText("Write a message...");

    textField.addFocusListener(new FocusListener() {

        @Override

        public void focusGained(FocusEvent e) {

            if (textField.getText().equals("Write a message...")) {

                textField.setText("");

            }

        }

        @Override

        public void focusLost(FocusEvent e) {

            if (textField.getText().isEmpty()) {

                textField.setText("Write a message...");

            }

        }

    });

    textField.addActionListener(new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            sendMessage();

        }

    });

    inputPanel.add(textField, BorderLayout.CENTER);

    sendButton = new JButton("Send");

    inputPanel.add(sendButton, BorderLayout.EAST);

    sendButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            sendMessage();

        }

    });

    privateMessageButton = new JButton("Private Message");

    inputPanel.add(privateMessageButton, BorderLayout.WEST);

    privateMessageButton.addActionListener(new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            showPrivateMessageDialog();

        }

    });

    panel.add(inputPanel, BorderLayout.SOUTH);

    frame.add(panel);

    frame.setVisible(true);

    textField.requestFocusInWindow(); // 请求焦点

    userName = JOptionPane.showInputDialog(frame, "Enter your username:");

    writer.println(userName);

    writer.flush(); // 确保立即发送用户名

}

* 4：发送和接收消息
* sendMessage方法用于发送公聊消息。
* showPrivateMessageDialog和sendPrivateMessage方法用于发送私聊消息。
* ReadThread类用于接收服务器发送的消息并更新界面。

private void sendMessage() {

    String message = textField.getText();

    System.out.println("Attempting to send message: " + message); // 调试信息

    if (message != null && !message.trim().isEmpty() && !message.equals("Write a message...")) {

        writer.println(message);

        writer.flush(); // 确保立即发送消息

        System.out.println("Message sent: " + message); // 调试信息

        textArea.append("You: " + message + "\n"); // 显示已发送消息

        textField.setText(""); // 发送后清空输入框

    }

}

private void showPrivateMessageDialog() {

    String recipient = JOptionPane.showInputDialog(frame, "Enter the username :");

    if (recipient != null && !recipient.trim().isEmpty()) {

        String message = JOptionPane.showInputDialog(frame, "Enter your private message:");

        if (message != null && !message.trim().isEmpty()) {

            sendPrivateMessage(recipient, message);

        }

    }

}

private void sendPrivateMessage(String recipient, String message) {

    writer.println("/w " + recipient + " " + message);

    writer.flush(); // 确保立即发送消息

    System.out.println("Private message sent to " + recipient + ": " + message); // 调试信息

    textArea.append("To " + recipient + ": " + message + "\n"); // 显示已发送消息

}

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();

                if (response != null) {

                    client.displayMessage(response);

                    System.out.println("Received message: " + response); // 调试信息

                }

            } catch (IOException ex) {

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

                ex.printStackTrace();

                break;

            }

        }

    }

}

## **4,3主程序模块（Main）**

**启动入口：**

负责启动服务器和客户端，提供系统的统一入口。

可以在同一台机器上启动服务器和客户端进行测试，也可以在不同的机器上分别启动。

public class Main {

    public static void main(String[] args) {

        // 启动服务器和客户端的入口

        new Thread(() -> {

            try {

                ChatServer.main(args);

            } catch (Exception e) {

                e.printStackTrace();

            }

        }).start();

        new Thread(() -> {

            try {

                ChatClient.main(args);

            } catch (Exception e) {

                e.printStackTrace();

            }

        }).start();

    }

}

# **5..主要模块的算法说明**

本节将详细说明服务器端和客户端各个主要模块的实现思路和算法。

## **5.1服务器端：ChatServer.java**

**1. 连接管理模块**

实现思路：

使用ServerSocket在指定端口上监听客户端连接请求。

每当有客户端连接时，接受连接请求并创建一个新的线程UserThread来处理该连接。

**算法：**

public static void main(String[] args) {

    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, userThreads);

            userThreads.add(newUser);

            newUser.start();

        }

    } catch (IOException ex) {

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

        ex.printStackTrace();

    }

}

**2. 用户管理模块**

实现思路：

使用线程安全的集合（Collections.synchronizedSet）来存储和管理所有连接的用户线程。

在用户连接时，将用户线程添加到集合中。

在用户断开时，从集合中移除用户线程，并通知其他用户。

**算法：**

class UserThread extends Thread {

    private Socket socket;

    private PrintWriter writer;

    private static Set<UserThread> userThreads;

    private String userName;

    public UserThread(Socket socket, Set<UserThread> userThreads) {

        this.socket = socket;

        UserThread.userThreads = userThreads;

    }

    public void run() {

        try {

            InputStream input = socket.getInputStream();

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

            OutputStream output = socket.getOutputStream();

            writer = new PrintWriter(output, true);

            userName = reader.readLine();

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

            broadcast(serverMessage, this);

            printUsers();

            String clientMessage;

            do {

                clientMessage = reader.readLine();

                if (clientMessage != null) {

                    System.out.println("Received message: " + clientMessage); // 调试信息

                    if (clientMessage.startsWith("/w ")) {

                        String[] messageParts = clientMessage.split(" ", 3);

                        if (messageParts.length == 3) {

                            String recipient = messageParts[1];

                            String privateMessage = messageParts[2];

                            if (!ChatServer.sendPrivateMessage(recipient, privateMessage, userName)) {

                                writer.println("User " + recipient + " not found.");

                            }

                        }

                    } else {

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

                        broadcast(serverMessage, this);

                    }

                }

            } while (clientMessage != null && !clientMessage.equalsIgnoreCase("bye"));

            removeUser(this);

            socket.close();

            serverMessage = userName + " has quitted.";

            broadcast(serverMessage, this);

        } catch (IOException ex) {

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

            ex.printStackTrace();

        }

    }

    void removeUser(UserThread aUser) {

        boolean removed = userThreads.remove(aUser);

        if (removed) {

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

        }

    }

}

**3. 消息处理模块**

实现思路：

接收客户端发送的消息，判断消息类型（公聊或私聊）。

如果是公聊消息，调用broadcast方法将消息发送给所有在线用户。

如果是私聊消息，调用sendPrivateMessage方法将消息发送给指定用户。

**算法：**

void broadcast(String message, UserThread excludeUser) {

    for (UserThread aUser : userThreads) {

        if (aUser != excludeUser) {

            aUser.writer.println(message);

        }

    }

}

static boolean sendPrivateMessage(String recipient, String message, String sender) {

    for (UserThread aUser : userThreads) {

        if (aUser.getUserName().equals(recipient)) {

            aUser.sendPrivateMessage(sender, message);

            return true;

        }

    }

    return false;

}

void sendPrivateMessage(String sender, String message) {

    writer.println("Private message from " + sender + ": " + message);

}

## **5.2客户端：ChatClient.java**

**1. 连接模块**

实现思路：

使用Socket类连接服务器，指定服务器的IP地址和端口。

获取服务器的输入输出流，使用BufferedReader和PrintWriter进行消息的读取和发送。

**算法：**

public void execute() {

    try {

        socket = new Socket(hostname, port);

        InputStream input = socket.getInputStream();

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

        OutputStream output = socket.getOutputStream();

        writer = new PrintWriter(output, true); // 自动刷新

        createGUI();

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

    } catch (IOException ex) {

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

        ex.printStackTrace();

    }

}

**2. 用户界面模块**

实现思路：

使用Swing组件创建图形用户界面，包括窗口、文本区域、输入框、按钮等。

在窗口中添加事件监听器，处理用户的输入和按钮点击事件。

**算法：**

private void createGUI() {

    frame = new JFrame("Chat Client");

    frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    frame.setSize(400, 400);

    JPanel panel = new JPanel();

    panel.setLayout(new BorderLayout());

    textArea = new JTextArea();

    textArea.setEditable(false);

    panel.add(new JScrollPane(textArea), BorderLayout.CENTER);

    JPanel inputPanel = new JPanel();

    inputPanel.setLayout(new BorderLayout());

    textField = new JTextField();

    textField.setEditable(true);

    textField.setColumns(25);

    textField.setForeground(Color.BLACK);

    textField.setBackground(Color.LIGHT\_GRAY);

    textField.setText("Write a message...");

    textField.addFocusListener(new FocusListener() {

        @Override

        public void focusGained(FocusEvent e) {

            if (textField.getText().equals("Write a message...")) {

                textField.setText("");

            }

        }

        @Override

        public void focusLost(FocusEvent e) {

            if (textField.getText().isEmpty()) {

                textField.setText("Write a message...");

            }

        }

    });

    // 按回车键发送消息

    textField.addActionListener(new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            sendMessage();

        }

    });

    inputPanel.add(textField, BorderLayout.CENTER);

    sendButton = new JButton("Send");

    inputPanel.add(sendButton, BorderLayout.EAST);

    sendButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            sendMessage();

        }

    });

    privateMessageButton = new JButton("Private Message");

    inputPanel.add(privateMessageButton, BorderLayout.WEST);

    privateMessageButton.addActionListener(new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            showPrivateMessageDialog();

        }

    });

    panel.add(inputPanel, BorderLayout.SOUTH);

    frame.add(panel);

    frame.setVisible(true);

    textField.requestFocusInWindow(); // 请求焦点

    userName = JOptionPane.showInputDialog(frame, "Enter your username:");

    writer.println(userName);

    writer.flush(); // 确保立即发送用户名

}

**3. 消息处理模块**

发送消息时，从输入框获取用户输入的消息，并使用PrintWriter将消息发送到服务器。

接收消息时，通过新线程持续监听服务器发送的消息，并更新文本区域显示收到的消息。

**算法：**

private void sendMessage() {

    String message = textField.getText();

    System.out.println("Attempting to send message: " + message); // 调试信息

    if (message != null && !message.trim().isEmpty() && !message.equals("Write a message...")) {

        writer.println(message);

        writer.flush(); // 确保立即发送消息

        System.out.println("Message sent: " + message); // 调试信息

        textArea.append("You: " + message + "\n"); // 显示已发送消息

        textField.setText(""); // 发送后清空输入框

    }

}

private void showPrivateMessageDialog() {

    String recipient = JOptionPane.showInputDialog(frame, "Enter the username :");

    if (recipient != null && !recipient.trim().isEmpty()) {

        String message = JOptionPane.showInputDialog(frame, "Enter your private message:");

        if (message != null && !message.trim().isEmpty()) {

            sendPrivateMessage(recipient, message);

        }

    }

}

private void sendPrivateMessage(String recipient, String message) {

    writer.println("/w " + recipient + " " + message);

    writer.flush(); // 确保立即发送消息

    System.out.println("Private message sent to " + recipient + ": " + message); // 调试信息

    textArea.append("To " + recipient + ": " + message + "\n"); // 显示已发送消息

}

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();

                if (response != null) {

                    client.displayMessage(response);

                    System.out.println("Received message: " + response); // 调试信息

                }

            } catch (IOException ex) {

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

                ex.printStackTrace();

                break;

            }

        }

    }

}

void displayMessage(String message) {

    textArea.append(message + "\n");

}

## **5.3 主程序模块（Main）**

实现思路：

**1.启动入口：**

主程序负责启动服务器和客户端，提供系统的统一入口。

可以在同一台机器上启动服务器和客户端进行测试，也可以在不同的机器上分别启动。

使用多线程技术，分别启动服务器和客户端。

**算法：**

public class Main {

    public static void main(String[] args) {

        // 启动服务器和客户端的入口

        new Thread(() -> {

            try {

                ChatServer.main(args);

            } catch (Exception e) {

                e.printStackTrace();

            }

        }).start();

        new Thread(() -> {

            try {

                ChatClient.main(args);

            } catch (Exception e) {

                e.printStackTrace();

            }

        }).start();

}

}

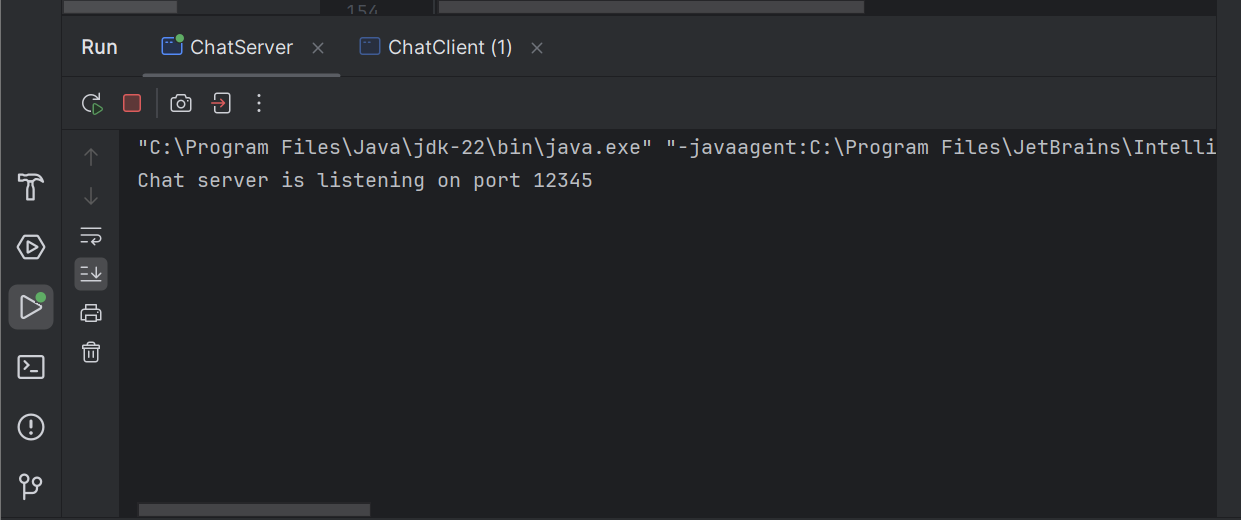
# **6.** **运行结果：**

**以下是运行这个聊天程序时的典型界面、输入和输出数据**

## 6.1运行 ChatServer

在命令行中

运行 ChatServer:

****

## 6.2运行 ChatClient

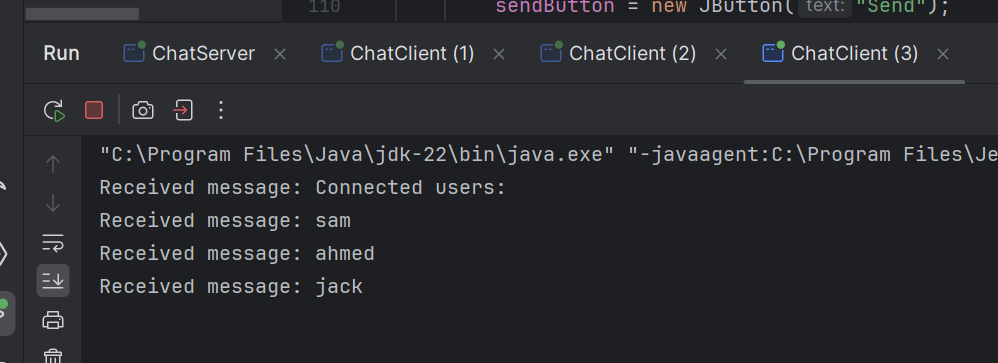
在三个不同的命令行窗口中分别运行 ChatClient:

**服务器端**：显示每个用户的连接、接收到的消息以及私信内容。

**客户端1 (sam)：**

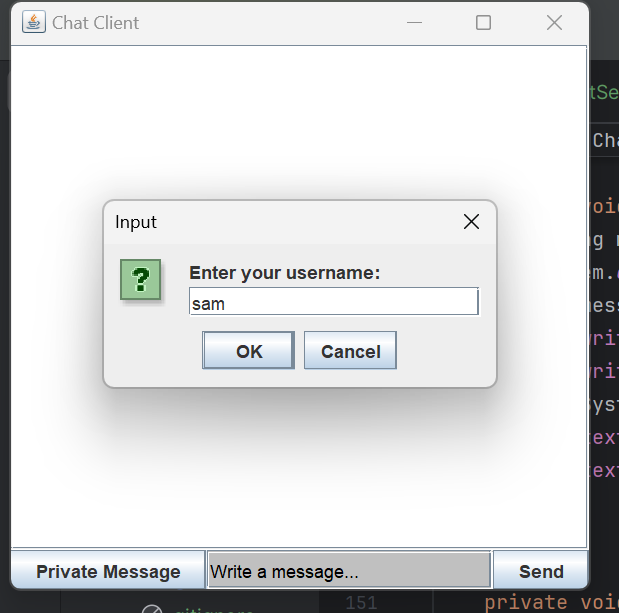
**客户端2 (ahmed)：**

**客户端3 (jack)：**

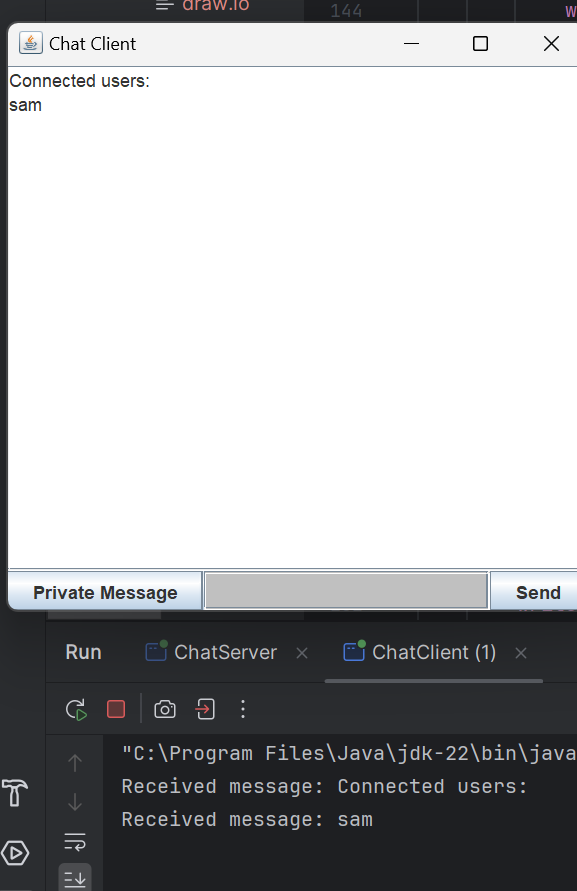
****

**客户端1连接**

**启动客户端1，连接到服务器并输入用户名sam。**

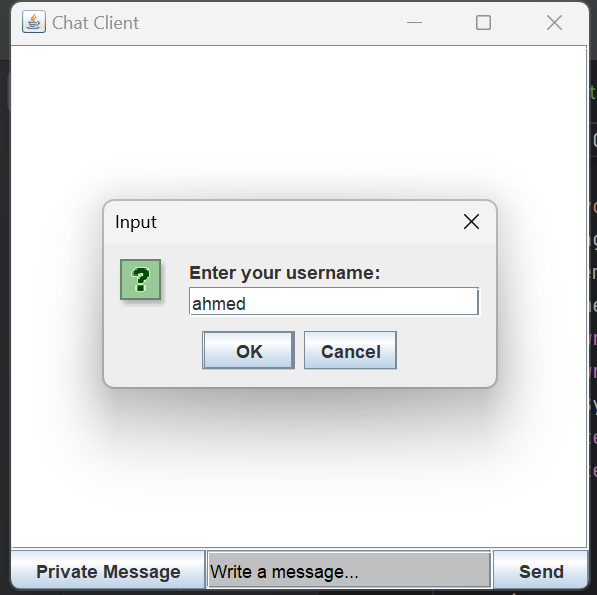
****

**客户端1 (sam)界面：**

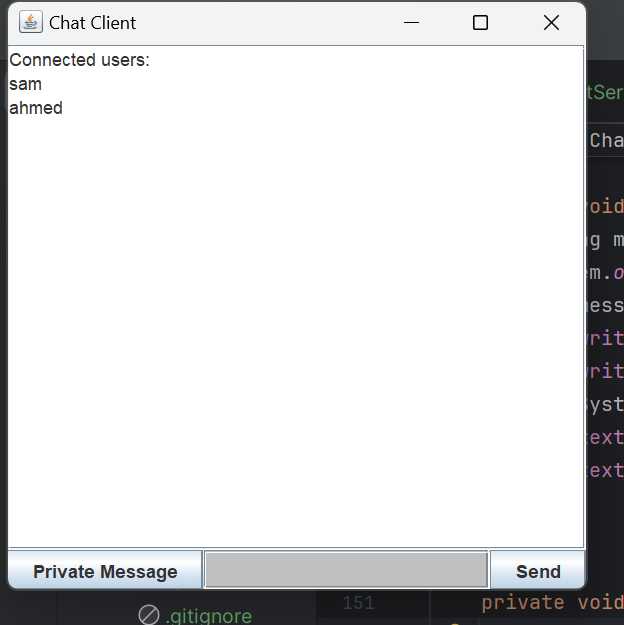
****

**客户端2连接**

**启动客户端2，连接到服务器并输入用户名**

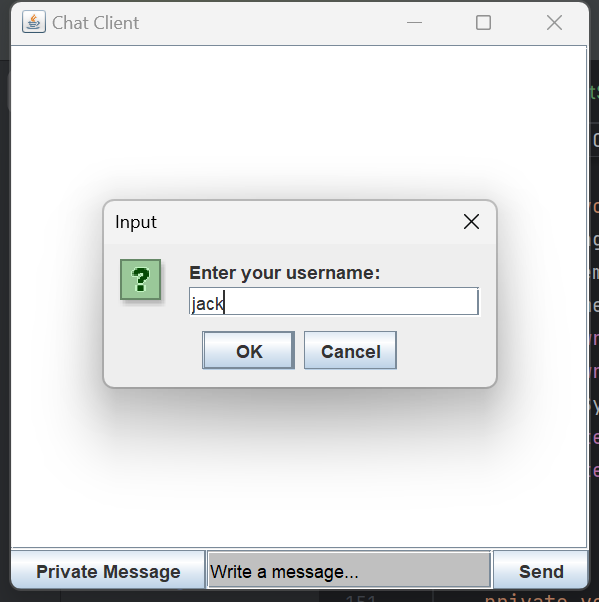


**客户端2 (ahmed)界面：**

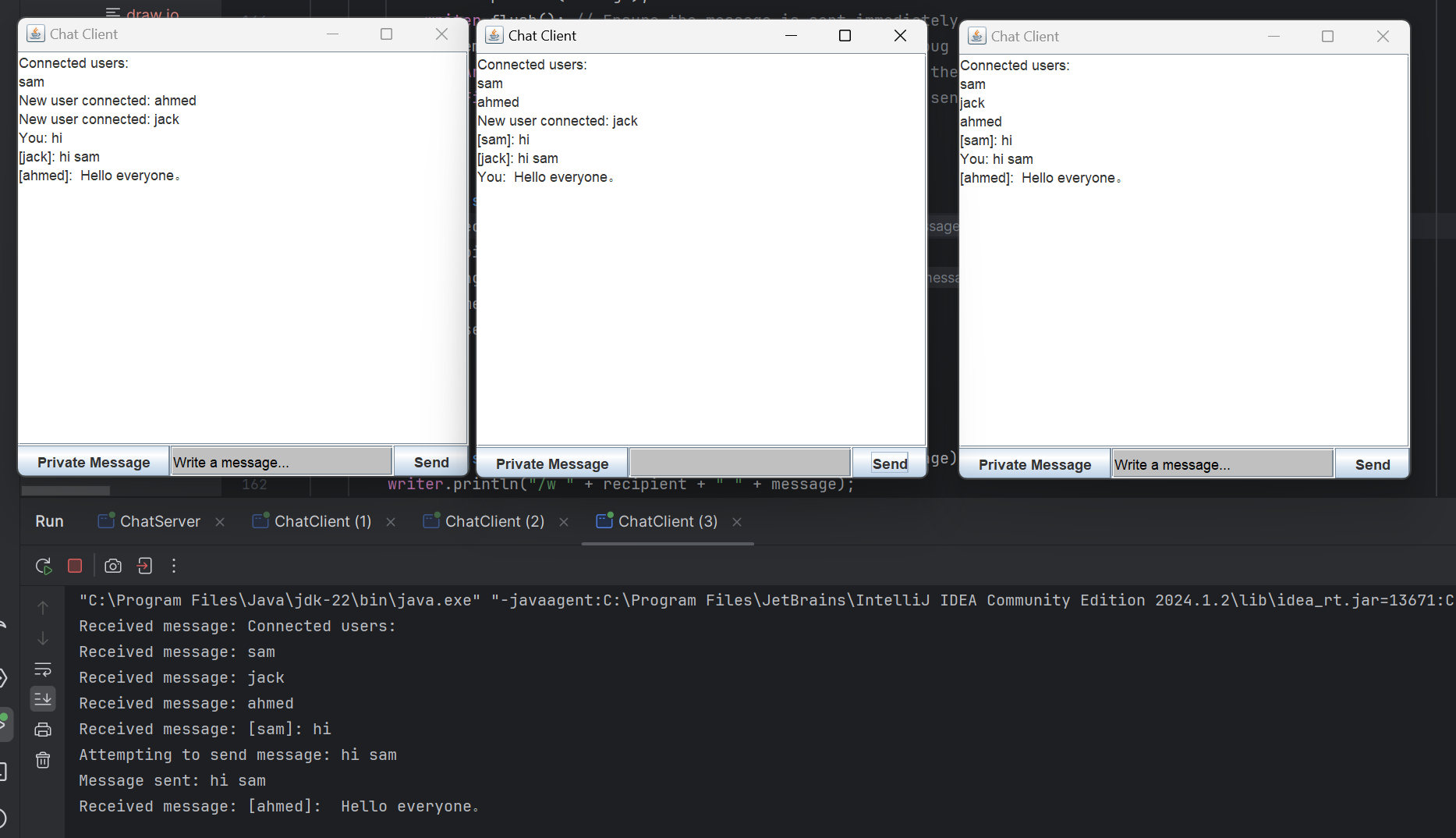
****

**客户端3 (jack)：**

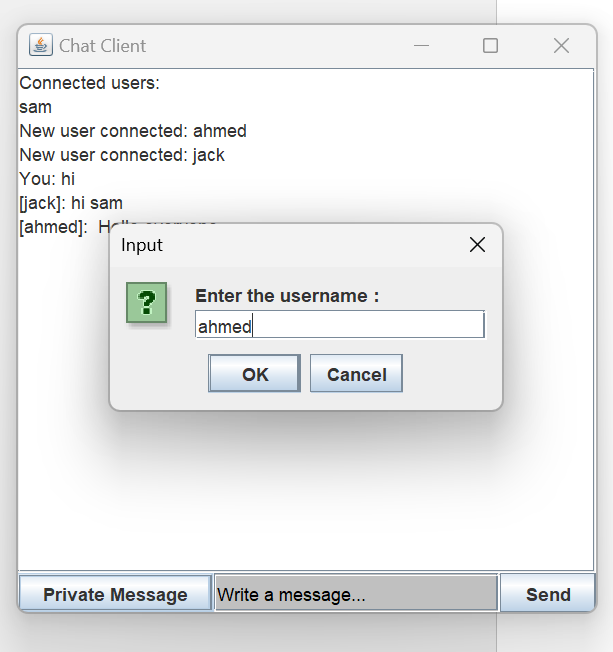
**启动客户端3，连接到服务器并输入用户名**

****

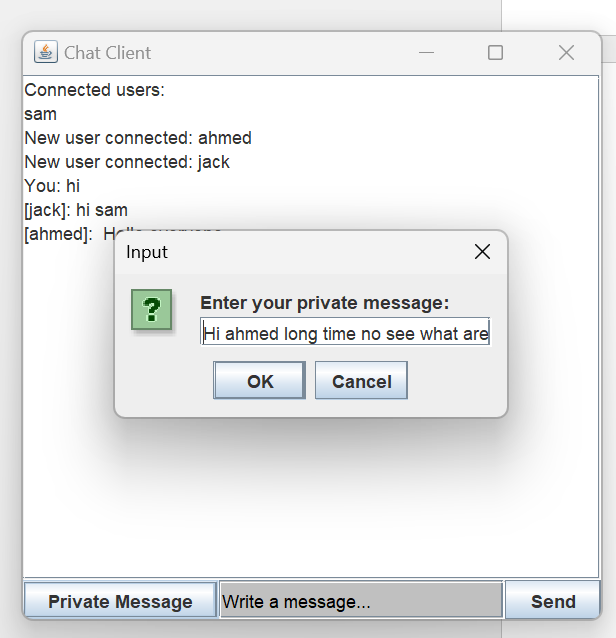
**根据最新的截图，我们可以看到三个客户端连接到服务器并相互发送消息。**



**用户可以通过点击“Private Message”按钮发送私信然后写他的名字然后写消息**

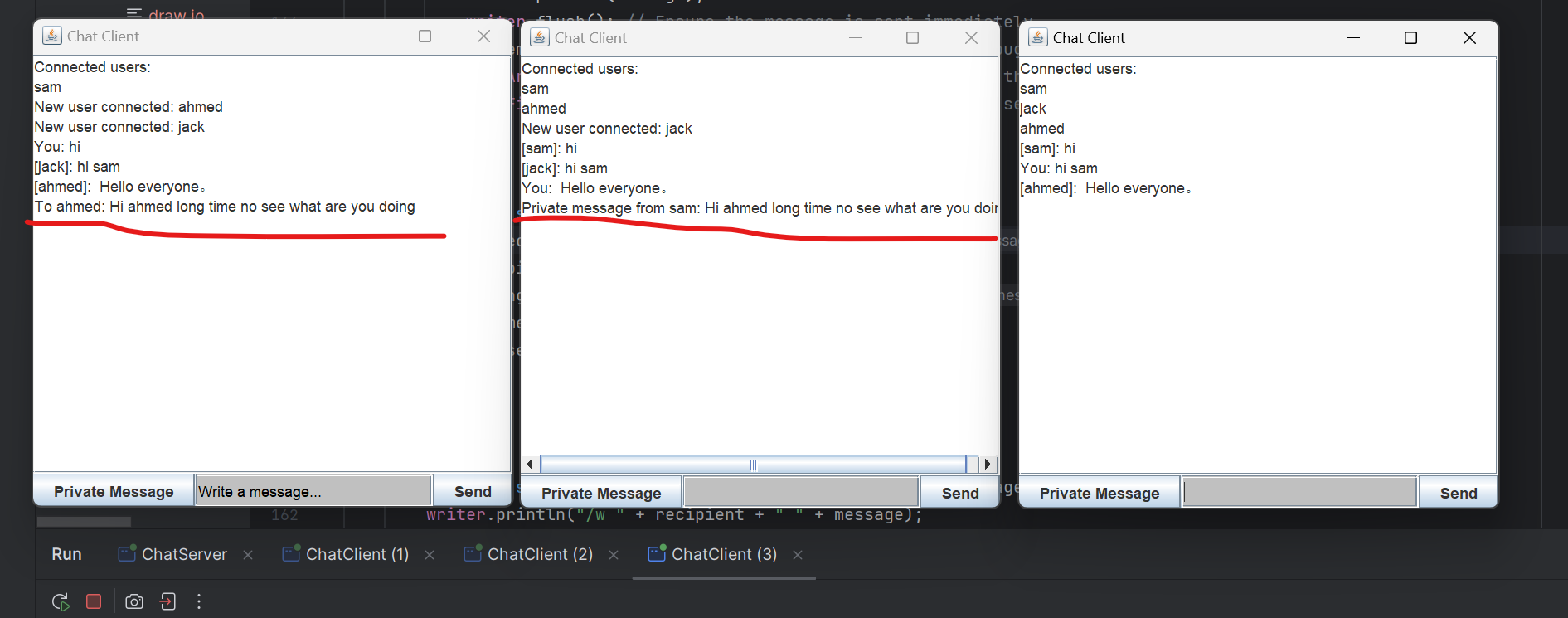


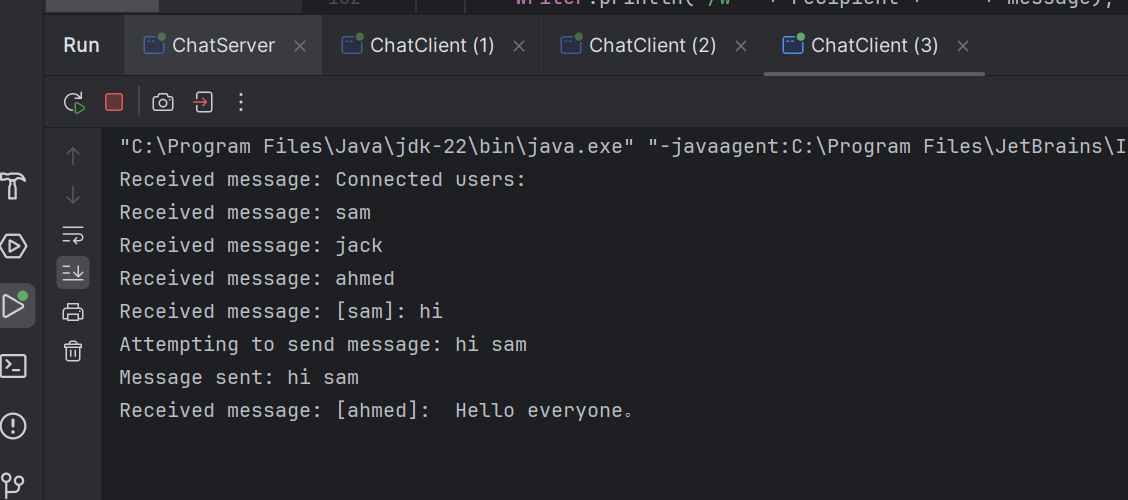
**消息内容**



**客户端1. sam发送私信 "Hi ahmed long time no see what are you doing" 给用户 ahmed**

**Ahmed 成功收到了消息**





# **7.运行结果总结**

根据最新的截图，我们可以看到三个客户端连接到服务器，并且用户之间发送了私信。下面是详细的运行步骤和每个客户端的界面与消息传输。

**客户端界面**

每个客户端都有一个聊天窗口，显示连接的用户、系统消息和聊天消息。

服务器端：显示每个用户的连接、接收到的消息以及私信内容。

用户可以输入消息并发送给所有连接的客户端。

用户可以通过点击“Private Message”按钮发送私信。然后写他的名字然后写消息

# **8.课程设计总结**

## **8.1遇到的问题**

在本次课程设计中，主要遇到了以下几个问题：

**并发处理：**

如何有效管理多个用户的并发连接，确保服务器能够正确处理每个用户的消息。

**消息同步：**

如何保证消息能够实时发送和接收，确保各个客户端之间的消息同步。

**私聊功能的实现：**

如何准确识别并转发私聊消息，确保私聊消息能够正确发送到指定的用户。

**用户界面设计：**

如何设计一个友好的用户界面，使用户能够方便地发送和接收消息。

## **8.2解决问题的过程**

**问题描述：**在多人聊天的情况下，服务器需要同时处理多个用户的连接和消息，这要求服务器能够支持并发操作。

**解决方案**：采用多线程技术，每个用户连接创建一个独立的线程进行处理。使用Collections.synchronizedSet来管理用户线程，确保线程安全。

**具体实现：**在服务器端，使用ServerSocket监听连接请求，每当有新的连接请求时，创建一个新的UserThread来处理该连接，并将其添加到用户线程集合中。

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, userThreads);

        userThreads.add(newUser);

        newUser.start();

    }

} catch (IOException ex) {

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

    ex.printStackTrace();

}

**消息同步：**

**问题描述：**在多人聊天中，消息需要在所有客户端之间实时同步，避免出现延迟或消息丢失。

**解决方案：**在客户端发送消息时，使用PrintWriter的自动刷新功能（autoFlush），确保消息立即发送到服务器；在服务器端，接收到消息后立即进行广播。

**具体实现：**客户端发送消息时，使用PrintWriter进行发送，确保消息实时发送；服务器端接收消息后，通过调用broadcast方法将消息发送给所有在线用户。

// Client-side message sending

private void sendMessage() {

    String message = textField.getText();

    if (message != null && !message.trim().isEmpty() && !message.equals("Write a message...")) {

        writer.println(message);

        writer.flush(); // Ensure the message is sent immediately

        textArea.append("You: " + message + "\n"); // Display the sent message

        textField.setText(""); // Clear the input field after sending

    }

}

// Server-side message broadcasting

void broadcast(String message, UserThread excludeUser) {

    for (UserThread aUser : userThreads) {

        if (aUser != excludeUser) {

            aUser.writer.println(message);

        }

    }

}

**私聊功能的实现：**

**问题描述**：需要实现用户之间的私聊功能，确保私聊消息能够准确发送到指定用户。

**解决方案：**在客户端输入私聊命令时，解析命令并发送到服务器；服务器接收到私聊消息后，根据目标用户名找到对应的用户线程并转发消息。

**具体实现：**客户端输入私聊命令时，解析命令格式并发送；服务器接收消息后，判断是否为私聊消息，如果是则调用sendPrivateMessage方法进行转发。

// Client-side private message sending

private void sendPrivateMessage(String recipient, String message) {

    writer.println("/w " + recipient + " " + message);

    writer.flush(); // Ensure the message is sent immediately

    textArea.append("To " + recipient + ": " + message + "\n"); // Display the sent message

}

// Server-side private message forwarding

static boolean sendPrivateMessage(String recipient, String message, String sender) {

    for (UserThread aUser : userThreads) {

        if (aUser.getUserName().equals(recipient)) {

            aUser.sendPrivateMessage(sender, message);

            return true;

        }

    }

    return false;

}

**用户界面设计：**

**问题描述：**需要设计一个友好的用户界面，使用户能够方便地输入和查看消息。

**解决方案：**使用Java Swing创建用户界面，包含文本区域、输入框、发送按钮等组件；添加事件监听器，处理用户的输入和按钮点击事件。

**具体实现：**使用JFrame、JTextArea、JTextField、JButton等组件创建用户界面，并添加相应的事件监听器。

private void createGUI() {

    frame = new JFrame("Chat Client");

    frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    frame.setSize(400, 400);

    JPanel panel = new JPanel();

    panel.setLayout(new BorderLayout());

    textArea = new JTextArea();

    textArea.setEditable(false);

    panel.add(new JScrollPane(textArea), BorderLayout.CENTER);

    JPanel inputPanel = new JPanel();

    inputPanel.setLayout(new BorderLayout());

    textField = new JTextField();

    textField.setEditable(true);

    textField.setColumns(25);

    textField.setForeground(Color.BLACK);

    textField.setBackground(Color.LIGHT\_GRAY);

    textField.setText("Write a message...");

    textField.addFocusListener(new FocusListener() {

        @Override

        public void focusGained(FocusEvent e) {

            if (textField.getText().equals("Write a message...")) {

                textField.setText("");

            }

        }

        @Override

        public void focusLost(FocusEvent e) {

            if (textField.getText().isEmpty()) {

                textField.setText("Write a message...");

            }

        }

    });

    textField.addActionListener(new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            sendMessage();

        }

    });

    inputPanel.add(textField, BorderLayout.CENTER);

    sendButton = new JButton("Send");

    inputPanel.add(sendButton, BorderLayout.EAST);

    sendButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            sendMessage();

        }

    });

    privateMessageButton = new JButton("Private Message");

    inputPanel.add(privateMessageButton, BorderLayout.WEST);

    privateMessageButton.addActionListener(new ActionListener() {

        @Override

        public void actionPerformed(ActionEvent e) {

            showPrivateMessageDialog();

        }

    });

    panel.add(inputPanel, BorderLayout.SOUTH);

    frame.add(panel);

    frame.setVisible(true);

    textField.requestFocusInWindow(); // 请求焦点

    userName = JOptionPane.showInputDialog(frame, "Enter your username:");

    writer.println(userName);

    writer.flush(); // 确保立即发送用户名

}

## **8.3体会与收获**

在这次课程设计中，我学会了用Java进行Socket编程，理解了基本的网络通信原理，并掌握了多线程编程技巧。在实践中，我遇到了并发处理、消息同步和实现私聊功能等多个问题，通过查阅资料和反复调试，最终成功解决了这些问题。我还意识到了设计用户友好界面的重要性，并使用Swing组件创建了一个简洁直观的聊天界面。

网络编程技能：通过这次设计，我掌握了Java网络编程的基本技能，包括Socket编程和输入输出流的处理等。

多线程编程：学会了用多线程处理并发连接，确保服务器能够稳定运行。

GUI设计：通过使用Swing组件，掌握了基本的GUI设计技能，能够设计出简洁实用的用户界面。

问题解决能力：在遇到问题时，通过查阅资料、讨论和实验，培养了分析和解决问题的能力。

## **8.4思考**

需求分析的重要性：在开发过程中，充分的需求分析和设计是项目顺利进行的基础。只有清楚了解需求，才能设计出合理的系统架构。

模块化设计：采用模块化设计思想，将系统分解为多个功能模块，有助于提高代码的可维护性和可扩展性。

异常处理：在网络编程中，异常情况不可避免。通过完善的异常处理机制，确保系统在遇到错误时能够优雅地处理，并尽量减少对用户的影响。

用户体验：设计一个用户友好的界面，提升用户体验，是软件开发中的重要环节。良好的用户界面能够显著提高用户的使用满意度。

通过这次课程设计，我对程序开发有了更深的认识和理解，也积累了宝贵的实践经验。在未来的学习和工作中，我将继续提升自己的编程技能，不断学习和实践，争取成为一名优秀的软件开发工程师。

# **9.参考文献**

1. Java官方文档Oracle，

<https://docs.oracle.com/javase/8/docs/>] https://docs.oracle.com/javase/8/docs/

2. Java TutorialsOracle

<https://docs.oracle.com/javase/tutorial/>] (https://docs.oracle.com/javase/tutorial/

3. Stack Overflow

- [Stack Overflow]

https://stackoverflow.com/

4. GitHub

- [GitHub]

https://github.com/

通过查阅以上参考文献，我在本次课程设计中获得了丰富的知识和技术支持，成功完成了多人聊天室系统的开发和实现。这些文献不仅为我提供了理论基础，还给予了实践指导，使我在Java编程和网络通信方面有了显著提升。

# **附录代码：主要程序清单**

## **10.1 ChatServer.java**

此程序是服务器端的主程序，负责监听客户端的连接，并处理客户端发送的消息，包括广播消息和私聊消息。

package com.example.chatserver;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Collections;  
import java.util.HashSet;  
import java.util.Set;  
  
public class ChatServer {  
 private static final int *PORT* = 12345;  
 private static Set<UserThread> *userThreads* = Collections.*synchronizedSet*(new HashSet<>());  
  
 // 服务器主方法  
 public static void main(String[] args) {  
 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, *userThreads*);  
 *userThreads*.add(newUser);  
 newUser.start();  
 }  
 } catch (IOException ex) {  
 System.*out*.println("Error in the server: " + ex.getMessage());  
 ex.printStackTrace();  
 }  
 }  
  
 // 发送私聊消息  
 static boolean sendPrivateMessage(String recipient, String message, String sender) {  
 for (UserThread aUser : *userThreads*) {  
 if (aUser.getUserName().equals(recipient)) {  
 aUser.sendPrivateMessage(sender, message);  
 return true;  
 }  
 }  
 return false;  
 }  
}  
  
class UserThread extends Thread {  
 private Socket socket;  
 private PrintWriter writer;  
 private static Set<UserThread> *userThreads*;  
 private String userName;  
  
 public UserThread(Socket socket, Set<UserThread> userThreads) {  
 this.socket = socket;  
 UserThread.*userThreads* = userThreads;  
 }  
  
 public void run() {  
 try {  
 InputStream input = socket.getInputStream();  
 BufferedReader reader = new BufferedReader(new InputStreamReader(input));  
  
 OutputStream output = socket.getOutputStream();  
 writer = new PrintWriter(output, true);  
  
 userName = reader.readLine();  
 String serverMessage = "New user connected: " + userName;  
 broadcast(serverMessage, this);  
  
 printUsers(); // 向新用户发送用户列表  
  
 String clientMessage;  
  
 do {  
 clientMessage = reader.readLine();  
 if (clientMessage != null) {  
 System.*out*.println("Received message: " + clientMessage); // 调试信息  
 if (clientMessage.startsWith("/w ")) {  
 String[] messageParts = clientMessage.split(" ", 3);  
 if (messageParts.length == 3) {  
 String recipient = messageParts[1];  
 String privateMessage = messageParts[2];  
 if (!ChatServer.*sendPrivateMessage*(recipient, privateMessage, userName)) {  
 writer.println("User " + recipient + " not found.");  
 }  
 }  
 } else {  
 serverMessage = "[" + userName + "]: " + clientMessage;  
 broadcast(serverMessage, this);  
 }  
 }  
 } while (clientMessage != null && !clientMessage.equalsIgnoreCase("bye"));  
  
 removeUser(this);  
 socket.close();  
  
 serverMessage = userName + " has quitted.";  
 broadcast(serverMessage, this);  
  
 } catch (IOException ex) {  
 System.*out*.println("Error in UserThread: " + ex.getMessage());  
 ex.printStackTrace();  
 }  
 }  
  
 void printUsers() {  
 if (*userThreads*.isEmpty()) {  
 writer.println("No other users connected");  
 } else {  
 writer.println("Connected users: ");  
 for (UserThread userThread : *userThreads*) {  
 if (userThread.userName != null) {  
 writer.println(userThread.userName);  
 }  
 }  
 }  
 }  
  
 void broadcast(String message, UserThread excludeUser) {  
 for (UserThread aUser : *userThreads*) {  
 if (aUser != excludeUser) {  
 aUser.writer.println(message);  
 }  
 }  
 }  
  
 void removeUser(UserThread aUser) {  
 boolean removed = *userThreads*.remove(aUser);  
 if (removed) {  
 System.*out*.println("The user " + userName + " quitted");  
 }  
 }  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void sendPrivateMessage(String sender, String message) {  
 writer.println("Private message from " + sender + ": " + message);  
 }  
}

## **10.2ChatClient.java:**

此程序是客户端的主程序，实现了一个简单的聊天客户端，通过套接字与服务器通信，并且有一个图形用户界面（GUI）来显示和发送消息。

package com.example.chatclient;  
  
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.FocusEvent;  
import java.awt.event.FocusListener;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.io.PrintWriter;  
import java.net.Socket;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JOptionPane;  
import javax.swing.JPanel;  
import javax.swing.JScrollPane;  
import javax.swing.JTextArea;  
import javax.swing.JTextField;  
  
public class ChatClient {  
 private String hostname;  
 private int port;  
 private String userName;  
 private Socket socket;  
 private PrintWriter writer;  
 private BufferedReader reader;  
 private JTextArea textArea;  
 private JTextField textField;  
 private JButton sendButton;  
 private JButton privateMessageButton;  
 private JFrame frame;  
  
 // 构造函数，初始化服务器地址和端口  
 public ChatClient(String hostname, int port) {  
 this.hostname = hostname;  
 this.port = port;  
 }  
  
 // 连接服务器并创建GUI界面  
 public void execute() {  
 try {  
 socket = new Socket(hostname, port);  
  
 InputStream input = socket.getInputStream();  
 reader = new BufferedReader(new InputStreamReader(input));  
  
 OutputStream output = socket.getOutputStream();  
 writer = new PrintWriter(output, true); // 自动刷新  
  
 createGUI();  
  
 new ReadThread(socket, this).start();  
  
 } catch (IOException ex) {  
 System.*out*.println("Error connecting to the server: " + ex.getMessage());  
 ex.printStackTrace();  
 }  
 }  
  
 // 创建图形用户界面  
 private void createGUI() {  
 frame = new JFrame("Chat Client");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(400, 400);  
  
 JPanel panel = new JPanel();  
 panel.setLayout(new BorderLayout());  
  
 textArea = new JTextArea();  
 textArea.setEditable(false);  
 panel.add(new JScrollPane(textArea), BorderLayout.*CENTER*);  
  
 JPanel inputPanel = new JPanel();  
 inputPanel.setLayout(new BorderLayout());  
  
 textField = new JTextField();  
 textField.setEditable(true);  
 textField.setColumns(25);  
 textField.setForeground(Color.*BLACK*);  
 textField.setBackground(Color.*LIGHT\_GRAY*);  
 textField.setText("Write a message...");  
 textField.addFocusListener(new FocusListener() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (textField.getText().equals("Write a message...")) {  
 textField.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (textField.getText().isEmpty()) {  
 textField.setText("Write a message...");  
 }  
 }  
 });  
  
 // 按回车键发送消息  
 textField.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 sendMessage();  
 }  
 });  
  
 inputPanel.add(textField, BorderLayout.*CENTER*);  
  
 sendButton = new JButton("Send");  
 inputPanel.add(sendButton, BorderLayout.*EAST*);  
  
 sendButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 sendMessage();  
 }  
 });  
  
 privateMessageButton = new JButton("Private Message");  
 inputPanel.add(privateMessageButton, BorderLayout.*WEST*);  
 privateMessageButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 showPrivateMessageDialog();  
 }  
 });  
  
 panel.add(inputPanel, BorderLayout.*SOUTH*);  
  
 frame.add(panel);  
 frame.setVisible(true);  
 textField.requestFocusInWindow(); // 请求焦点  
  
 userName = JOptionPane.*showInputDialog*(frame, "Enter your username:");  
 writer.println(userName);  
 writer.flush(); // 确保立即发送用户名  
 }  
  
 // 发送消息方法  
 private void sendMessage() {  
 String message = textField.getText();  
 System.*out*.println("Attempting to send message: " + message); // 调试信息  
 if (message != null && !message.trim().isEmpty() && !message.equals("Write a message...")) {  
 writer.println(message);  
 writer.flush(); // 确保立即发送消息  
 System.*out*.println("Message sent: " + message); // 调试信息  
 textArea.append("You: " + message + "\n"); // 显示已发送消息  
 textField.setText(""); // 发送后清空输入框  
 }  
 }  
  
 // 显示私聊对话框  
 private void showPrivateMessageDialog() {  
 String recipient = JOptionPane.*showInputDialog*(frame, "Enter the username :");  
 if (recipient != null && !recipient.trim().isEmpty()) {  
 String message = JOptionPane.*showInputDialog*(frame, "Enter your private message:");  
 if (message != null && !message.trim().isEmpty()) {  
 sendPrivateMessage(recipient, message);  
 }  
 }  
 }  
  
 // 发送私聊消息  
 private void sendPrivateMessage(String recipient, String message) {  
 writer.println("/w " + recipient + " " + message);  
 writer.flush(); // 确保立即发送消息  
 System.*out*.println("Private message sent to " + recipient + ": " + message); // 调试信息  
 textArea.append("To " + recipient + ": " + message + "\n"); // 显示已发送消息  
 }  
  
 void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 String getUserName() {  
 return this.userName;  
 }  
  
 void displayMessage(String message) {  
 textArea.append(message + "\n");  
 }  
  
 public static void main(String[] args) {  
 String hostname = "localhost";  
 int port = 12345;  
  
 ChatClient client = new ChatClient(hostname, port);  
 client.execute();  
 }  
}  
  
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();  
 if (response != null) {  
 client.displayMessage(response);  
 System.*out*.println("Received message: " + response); // 调试信息  
 }  
 } catch (IOException ex) {  
 System.*out*.println("Error reading from server: " + ex.getMessage());  
 ex.printStackTrace();  
 break;  
 }  
 }  
 }  
}

## 10.3Main.java

// TIP To run the code, press <shortcut actionId="Run"/> or  
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.  
  
public class Main {  
 public static void main(String[] args) {  
 // TIP Press <shortcut actionId="ShowIntentionActions"/> with your caret at the highlighted text  
 // to see how IntelliJ IDEA suggests fixing it.  
 System.*out*.printf("Hello and welcome!");  
  
 for (int i = 1; i <= 5; i++) {  
 // TIP Press <shortcut actionId="Debug"/> to start debugging your code. We have set one <icon src="AllIcons.Debugger.Db\_set\_breakpoint"/> breakpoint  
 // for you, but you can always add more by pressing <shortcut actionId="ToggleLineBreakpoint"/>.  
 System.*out*.println("i = " + i);  
 }  
 }  
}