UNIVERSIDADE FEDERAL DE SANTA MARIA - UFSM

DESENVOLVIMENTO DE APLICATIVO PARA A DISCIPLINA DE POO
UFSM00274 - PROGRAMAÇÃO ORIENTADA A OBJETOS

APLICATIVO DE CONVERSA DESENVOLVIDO COM JAVA.

THÁSSIO GOMES SILVA
2023510194

Santa Maria - RS
Novembro / 2023

THÁSSIO GOMES SILVA
2023510194

IMPLEMENTAÇÃO DE APLICATIVO DE CONVERSA, USANDO JAVA PARA FINS ACADÊMICOS.

Relatório de prática submetido como parte da avaliação parcial da disciplina PROGRAMAÇÃO ORIENTADA A OBJETOS, orientado pelo Professor Dr. Osmar Marchi dos Santos.

Santa Maria - RS
Novembro / 2023

APLICATIVO DE CONVERSA USANDO JAVA

IDEALIZAÇÃO

Desenvolvimento de um aplicativo de chat em Java, utilizando a biblioteca Swing, que será uma aplicação de desktop. Esse aplicativo permitirá que os usuários troquem mensagens em tempo real de forma eficaz e intuitiva.

Palavras-chave: Java, JavaFX, Java Swing

BACKLOG

(i) Familiarização com Swing ou JavaFX.

(ii) Estruturação do Projeto.

(iii) Desenvolvimento da Logica e testes.

(iv) Implementação do Java Swing ou JavaFX.

Palavras-chave: JAVA, JAVAFX, JAVA Swing

PARTE 2

foi criado uma classe Server implementa a interface ActionListener e contém a lógica principal do servidor de chat, onde nos possibilita o envio de mensagem e recepção de mensagem. Foi implementaldo alguns metodos usando swing:

Server(): Este é o construtor da classe. Ele inicializa a janela do servidor e chama o método criarInterfaceGrafica().

criarInterfaceGrafica(): Este método configura a interface gráfica do servidor. Ele chama vários outros métodos para criar diferentes componentes da interface gráfica.

criarPainelSuperior(): Este método cria o painel superior da interface gráfica, que inclui o botão de voltar e os ícones de perfil, vídeo e telefone.

criarBotaoVoltar(): Este método cria o botão de voltar no painel superior. Quando clicado, ele fecha o aplicativo.

criarPainelMensagens(): Este método cria o painel onde as mensagens serão exibidas.

criarCampoTexto(): Este método cria o campo de texto onde o usuário pode digitar suas mensagens.

criarBotaoEnviar(): Este método cria o botão “Enviar”. Quando clicado, ele envia a mensagem digitada no campo de texto.

actionPerformed(ActionEvent): Este método é chamado quando o botão “Enviar” é clicado. Ele lida com o envio da mensagem.

formatarMensagem(String): Este método formata a mensagem para ser exibida no painel de mensagens.

atualizarPainelMensagens(JPanel): Este método atualiza o painel de mensagens com a nova mensagem formatada.

enviarMensagemServidor(String): Este método envia a mensagem para o cliente.

limparCampoTexto(): Este método limpa o campo de texto após o envio da mensagem.

atualizarJanela(): Este método atualiza a janela após o envio da mensagem.

main(String[]): Este é o ponto de entrada do programa. Ele inicia o servidor e aceita conexões do cliente. Ele também inicia uma nova thread para cada cliente conectado para lidar com a recepção de mensagens.

Observações

A parte do servidor ainda não esta operando. Está sendo implementado a parte do cliente do pacote do projeto. Estou adicionando os seguintes metodos tambem:

Interface do Cliente: Atualmente, o código implementa apenas a interface do servidor. Estou criando uma interface semelhante para o cliente para permitir que os usuários enviem e recebam mensagens.

Autenticação de Usuário: Para permitir que múltiplos usuários usem o aplicativo,

Persistência de Mensagens: Atualmente, as mensagens são perdidas quando o servidor é desligado.

Palavras-chave: JAVA, JAVAFX, JAVA Swing

PARTE 3 - Entrega Final

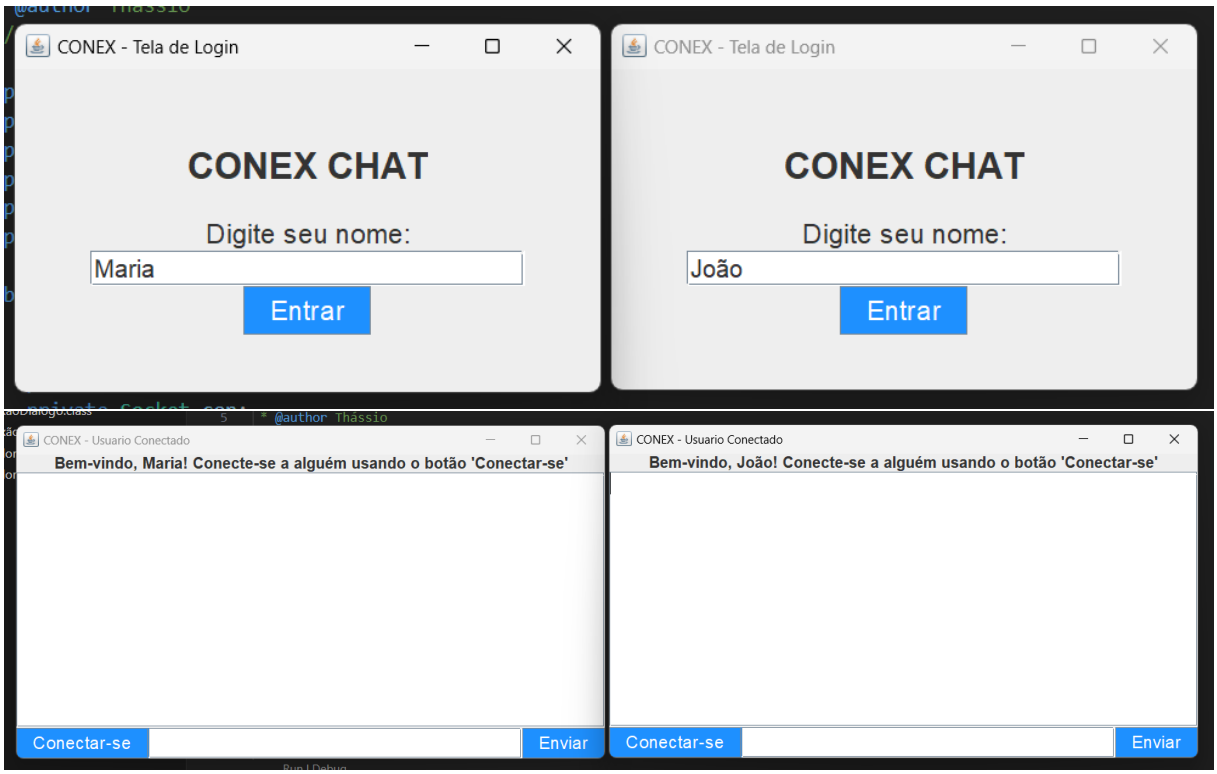
0.1 Visão Geral

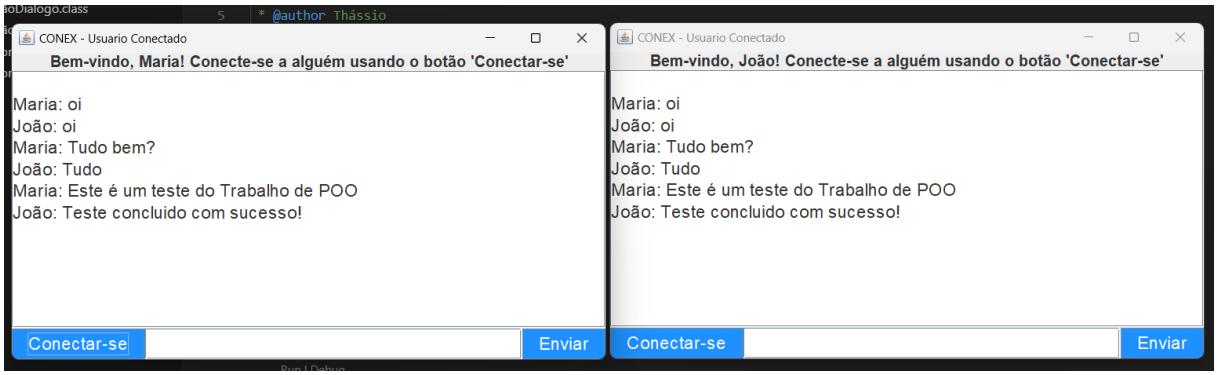
O código do aplicativo de conversas em Java é bem estruturado e modular, composto por três classes principais: `App`, `ConexãoDialogo`, e `Servidor`. Cada classe desempenha um papel específico, contribuindo para a clareza e manutenção do código. O codigo desde inicio veio sofrendo bastante alterações em sua logica, no final consegui usar alguns metodos e documentos da biblioteca swing; De inicio foi utilizado a IDE NetBeans, mas, por questão de facilidade de desenvolvimento, resolvi migrar para o vscode, pois é o ambiente que eu já estava acostumado a desenvolver, estou utilizando Socket para conseguir fazer a troca de dados no aplicativo.

0.1.1 Cliente (`App`)

O cliente é responsável pela interface gráfica (GUI) e pela comunicação com o servidor. Destacam-se as seguintes características:

- A GUI é construída usando o Swing, proporcionando uma experiência interativa.
- O código gerencia a conexão ao servidor e a comunicação em tempo real de forma eficiente.
- A funcionalidade de conexão direta é implementada de maneira clara, proporcionando uma interação amigável para o usuário.

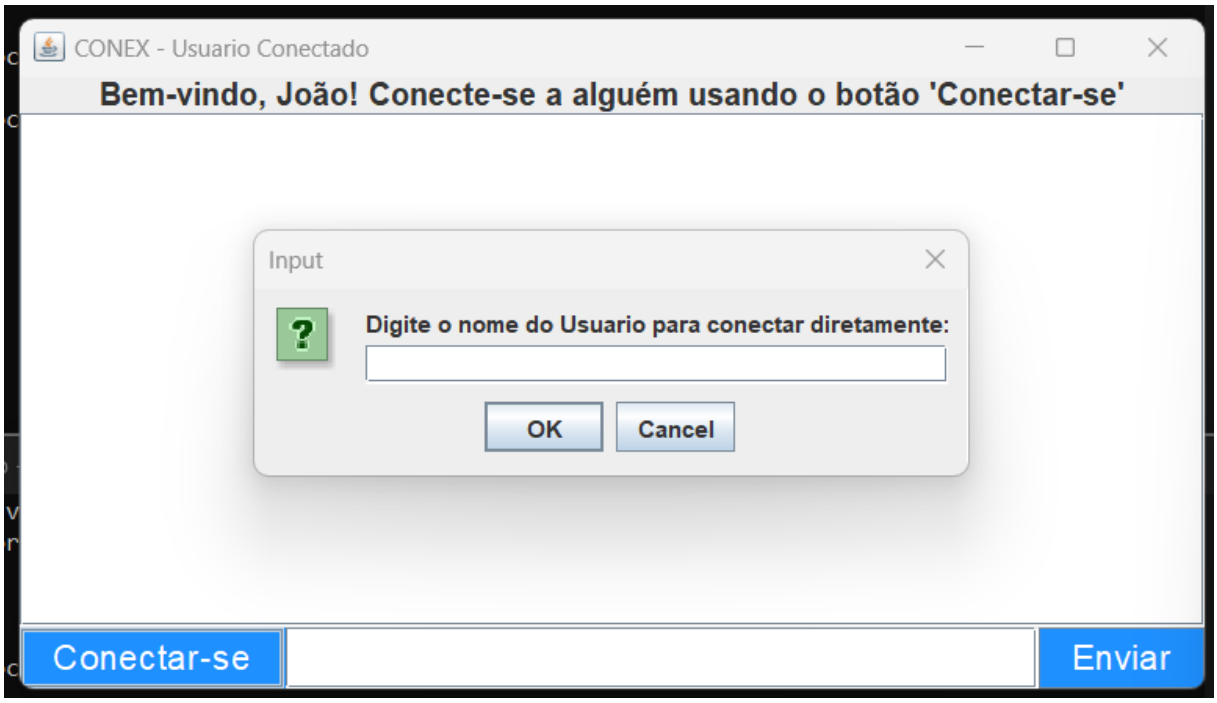




0.1.2 Diálogo de Conexão Direta (`ConexãoDialogo`)

Este componente fornece um diálogo simples para os usuários inserirem o nome do usuário alvo. Destacam-se as seguintes características:

- A classe é bem projetada para cumprir seu propósito específico.
- O método `getTargetUsername()` é eficaz para fornecer o nome de usuário inserido pelo usuário.



0.1.3 Servidor (`Servidor`)

O servidor gerencia as conexões entre clientes, permitindo comunicação direta ou através do servidor. Destacam-se as seguintes características:

- A classe gerencia eficientemente conexões, comunicação e solicitações de conexão direta.

- O código utiliza mapas para rastrear clientes e solicitações de conexão direta, promovendo uma estrutura limpa.
- A lógica para conexões diretas é bem implementada, fornecendo uma base sólida para futuras extensões.

0.2 Estrutura do Projeto

O projeto é composto por três classes principais: `App`, `ConexãoDialogo`, e `Servidor`. Cada classe desempenha um papel específico no sistema.

0.2.1 `App`

A classe `App` representa a interface do usuário e a lógica de comunicação com o servidor.

0.2.1.1 Membros Principais

- `mainFrame`: Janela principal da aplicação.
- `textField`: Campo de entrada de texto.
- `textArea`: Área de exibição de mensagens.
- `con`: Socket para comunicação com o servidor.
- `clientName`: Nome do cliente.
- `output`: Fluxo de saída para enviar mensagens.
- `serverInput`: Fluxo de entrada para receber mensagens do servidor.

0.2.1.2 Métodos Principais

- `connectToServer()`: Estabelece a conexão com o servidor.
- `sendMessage()`: Envia mensagens para o servidor.
- `requestDirectConnection()`: Solicita uma conexão direta com outro usuário.
- `connectToClient(String targetClient)`: Conecta-se diretamente a outro usuário.
- `closeConnection()`: Fecha a conexão com o servidor.

0.2.1.3 Funcionalidades

- Conexão ao servidor e comunicação em tempo real.
- Interface gráfica interativa com botões para enviar mensagens e conectar-se a outros usuários.
- Suporte para conexões diretas entre usuários.

0.2.2 `ConexãoDialogo`

A classe `ConexãoDialogo` fornece um diálogo para os usuários inserirem o nome do usuário ao qual desejam se conectar diretamente.

0.2.2.1 Membros Principais

- `usernameField`: Campo de texto para inserir o nome de usuário alvo.
- `connectButton`: Botão para confirmar a conexão.
- `targetUsername`: Variável que armazena o nome de usuário alvo.

0.2.2.2 Métodos Principais

- `ConexãoDialogo(JFrame parent)`: Construtor que recebe a janela principal como parâmetro.
- `initialize()`: Inicializa o diálogo e seus componentes.
- `getTargetUsername()`: Retorna o nome de usuário alvo inserido pelo usuário.

0.2.2.3 Funcionalidades

- Fornecer um diálogo simples para inserir o nome do usuário alvo.
- Permitir que o usuário confirme a conexão direta.

0.2.3 `Servidor`

A classe `Servidor` gerencia as conexões entre os clientes, permitindo comunicação direta ou através do servidor.

0.2.3.1 Membros Principais

- `serverSocket`: Socket do servidor.
- `clientMap`: Mapa para mapear nomes de usuários para seus respectivos sockets.
- `directConnectionRequests`: Mapa para armazenar solicitações de conexão direta.

0.2.3.2 Métodos Principais

- `Servidor()`: Construtor que inicializa mapas para rastrear clientes e solicitações de conexão direta.
- `startServer()`: Inicia o servidor e aguarda conexões.
- `handleClient(Socket clientSocket)`: Lida com a comunicação contínua de um cliente.
- `startDirectCommunication(String client1, String client2)`: Inicia a comunicação direta entre dois clientes.
- `forwardMessage(String sender, String message)`: Encaminha mensagens para todos os clientes conectados.

0.2.3.3 Funcionalidades

- Aceitar conexões de clientes e gerenciar suas comunicações.
- Permitir solicitações e aceitação de conexões diretas.
- Encaminhar mensagens entre clientes.

0.3 Mapa de Classes

0.3.1 `App`

- **Atributos:**
  - `- mainFrame: JFrame`
  - `- textField: JTextField`
  - `- textArea: JTextArea`
  - `- con: Socket`
  - `- clientName: String`
  - `- output: DataOutputStream`

  - - serverInput: DataInputStream

- **Métodos:**
  - + connectToServer(): void
  - + sendMessage(): void
  - + requestDirectConnection(): void
  - + connectToClient(targetClient: String): void
  - + closeConnection(): void

0.3.2 `ConexãoDialogo`

- **Atributos:**
  - - usernameField: JTextField
  - - connectButton: JButton
  - - targetUsername: String

- **Métodos:**
  - + ConexãoDialogo(parent: JFrame): void
  - + initialize(): void
  - + getTargetUsername(): String

0.3.3 `Servidor`

- **Atributos:**
  - - serverSocket: ServerSocket
  - - clientMap: Map<String, Socket>
  - - directConnectionRequests: Map<String, Socket>

- **Métodos:**
  - + Servidor(): void
  - + startServer(): void
  - + handleClient(clientSocket: Socket): void
  - + startDirectCommunication(client1: String, client2: String): void
  - + forwardMessage(sender: String, message: String): void

0.4 Conclusão

Em resumo, o aplicativo de conversas em Java representa uma ferramenta de comunicação interativa, desenvolvida com base nos princípios da Programação Orientada a Objetos (POO). A estrutura do projeto é organizada em três partes principais: o lado do usuário (`App`), um diálogo para conexões diretas (`ConexãoDialogo`), e o servidor (`Servidor`).

Ao utilizar a POO, conseguimos criar uma aplicação modular e fácil de entender, dividindo-a em partes distintas com responsabilidades específicas. Isso facilita a manutenção do código e permite adicionar novos recursos de forma mais eficiente no futuro.

O usuário interage com a aplicação através de uma interface gráfica amigável, podendo enviar mensagens, se conectar a outros usuários, e até mesmo estabelecer comunicações diretas. Tudo isso é feito de maneira intuitiva, com a aplicação cuidando dos detalhes complexos nos bastidores.

No entanto, durante a análise mais profunda, identificamos áreas que poderiam ser aprimoradas, como o tratamento de erros em situações específicas. Isso garantiria uma experiência mais robusta para o usuário, especialmente em ambientes de rede menos previsíveis.

Em suma, este projeto ilustra a aplicação prática dos conceitos de Programação Orientada a Objetos, proporcionando uma visão clara de como os programas podem ser estruturados e organizados para serem eficientes, expansíveis e fáceis de manter.

0.5 Como Rodar o Código

Para executar o aplicativo de conversas em Java, siga as etapas abaixo:

1. Certifique-se de ter o ambiente de desenvolvimento Java (JDK) instalado em sua máquina.
2. Abra um terminal ou prompt de comando.
3. Navegue até o diretório onde o arquivo Java do servidor está localizado.
4. Compile o código Java do servidor usando o seguinte comando:

```
javac Servidor.java
```

   Isso criará os arquivos de classe necessários.

5. Inicie o servidor com o seguinte comando:

```
java Servidor
```

   O servidor estará pronto para receber conexões.

6. Agora, navegue até o diretório onde o arquivo Java do cliente (App) está localizado.

7. Compile o código Java do cliente usando o seguinte comando:

```
    javac App.java

```

8. Agora, execute o aplicativo do cliente com o seguinte comando:

```
    java App

```

   O aplicativo será iniciado, e a interface gráfica do usuário será exibida.

Lembre-se de que essas instruções podem variar dependendo do sistema operacional que você está usando. Certifique-se de ajustar as instruções conforme necessário para o ambiente em que está executando o código Java.

App

```
package conexodireta;

/**
 * Classe principal que representa a aplica  o de chat.
 * @author Th ssio
 */
import javax.swing.*;
import java.awt.*;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class App {
    private JFrame mainFrame;  // A janela principal da aplica  o
    private JTextField textField;  // Campo de texto para entrada de
    mensagens
    private static JTextArea textArea;  //  rea de exibi  o das
    mensagens
    private Socket con;  // Socket para a conex o
    private String clientName;  // Nome do cliente
    private DataOutputStream output;  // Stream de sa da para enviar
    mensagens
    private DataInputStream serverInput;  // Stream de entrada para receber
     mensagens do servidor

    /**
     * M todo principal que inicia a aplica  o.
     * @param args Argumentos da linha de comando (n o utilizado aqui)
```

```
26      */
27     public static void main(String[] args) {
28         SwingUtilities.invokeLater(() -> {
29             try {
30                 App window = new App();
31                 window.showLogin();
32             } catch (Exception e) {
33                 e.printStackTrace();
34             }
35         });
36     }
37 
38     /**
39      * Construtor da classe App que inicializa a interface gr fica.
40      */
41     public App() {
42         initializeMain();
43     }
44 
45     /**
46      * Inicializa a janela principal da aplica  o.
47      */
48     private void initializeMain() {
49         mainFrame = new JFrame();
50         // Configura  es da janela
51         mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
52         mainFrame.setTitle("CONEX - Usuario Conectado");
53         mainFrame.setSize(620, 350);
54         mainFrame.setLocationRelativeTo(null);
55         mainFrame.setLayout(new BorderLayout());
56 
57         //  rea  de exibi  o das mensagens
58         textArea = new JTextArea();
59         stylizeComponent(textArea, 18);
60         textArea.setEditable(false);
61         JScrollPane scrollPane = new JScrollPane(textArea);
62         mainFrame.add(scrollPane, BorderLayout.CENTER);
63 
64         // Painel para entrada de mensagens
65         JPanel inputPanel = new JPanel(new BorderLayout());
66         textField = new JTextField();
67         stylizeComponent(textField, 18);
68         inputPanel.add(textField, BorderLayout.CENTER);
69 
70         // Bot o para enviar mensagem
71         JButton sendButton = new JButton("Enviar");
72         stylizeComponent(sendButton, 18);
73         sendButton.setBackground(new Color(30, 144, 255));
```

```
74          sendButton.setForeground(Color.WHITE);
75          sendButton.addActionListener(e -> sendMessage());
76          inputPanel.add(sendButton, BorderLayout.EAST);
77
78          // Bot o para conectar-se a outro usu rio
79          JButton connectButton = new JButton("Conectar-se");
80          stylizeComponent(connectButton, 18);
81          connectButton.setBackground(new Color(30, 144, 255));
82          connectButton.setForeground(Color.WHITE);
83          connectButton.addActionListener(e -> requestDirectConnection());
84          inputPanel.add(connectButton, BorderLayout.WEST);
85
86          mainFrame.add(inputPanel, BorderLayout.SOUTH);
87
88          // Listener para fechar a conex o ao fechar a janela
89          mainFrame.addWindowListener(new java.awt.event.WindowAdapter() {
90              @Override
91              public void windowClosing(java.awt.event.WindowEvent
    windowEvent) {
92                  closeConnection();
93              }
94          });
95
96          // Adiciona a mensagem de boas-vindas
97          JLabel welcomeLabel = new JLabel("Bem-vindo, " + clientName + "!
    Conecte-se a algu m usando o bot o 'Conectar-se'");
98          stylizeComponent(welcomeLabel, Font.BOLD, 16);
99          welcomeLabel.setHorizontalAlignment(JLabel.CENTER);
100         mainFrame.add(welcomeLabel, BorderLayout.NORTH);
101     }
102
103     /**
104      * Exibe a tela de login para que o usu rio informe seu nome.
105      */
106     private void showLogin() {
107         JFrame loginFrame = new JFrame();
108         // Configura es da janela de login
109         loginFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
110         loginFrame.setTitle("CONEX - Tela de Login");
111         loginFrame.setSize(400, 250);
112         loginFrame.setLocationRelativeTo(null);
113
114         // Painel de login
115         JPanel loginPanel = new JPanel();
116         loginPanel.setLayout(new BoxLayout(loginPanel, BoxLayout.Y_AXIS));
117         loginPanel.setBorder(BorderFactory.createEmptyBorder(50, 50, 50,
    50));
118
```

```
        // T tulo
        JLabel titleLabel = new JLabel("CONEX CHAT");
        stylizeComponent(titleLabel, Font.BOLD, 24);
        titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        loginPanel.add(titleLabel);

        loginPanel.add(Box.createRigidArea(new Dimension(0, 20)));

        // Solicita  o de nome
        JLabel nameLabel = new JLabel("Digite seu nome:");
        stylizeComponent(nameLabel, Font.PLAIN, 18);
        nameLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        loginPanel.add(nameLabel);

        // Campo para inser  o do nome
        JTextField loginTextField = new JTextField();
        stylizeComponent(loginTextField, 16);
        loginTextField.setAlignmentX(Component.CENTER_ALIGNMENT);
        loginPanel.add(loginTextField);

        // Bot o de login
        JButton loginButton = new JButton("Entrar");
        stylizeComponent(loginButton, 18);
        loginButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        loginButton.setBackground(new Color(30, 144, 255));
        loginButton.setForeground(Color.WHITE);
        loginButton.addActionListener(e -> {
            clientName = loginTextField.getText();
            if (!clientName.isEmpty()) {
                loginFrame.dispose();
                initializeMain();
                connectToServer();
                SwingUtilities.invokeLater(() -> mainFrame.setVisible(true)
);
            } else {
                JOptionPane.showMessageDialog(loginFrame, "Por favor,
digite seu nome.");
            }
        });
        loginPanel.add(loginButton);

        loginFrame.setContentPane(loginPanel);
        loginFrame.setVisible(true);
    }

    /**
     * Conecta o cliente ao servidor.
     */
```

```
165     private void connectToServer() {
166         try {
167             con = new Socket("127.0.0.1", 8080);
168             output = new DataOutputStream(con.getOutputStream());
169             serverInput = new DataInputStream(con.getInputStream());
170 
171             // Enviar o nome do cliente para o servidor
172             output.writeUTF(clientName);
173 
174             // Inicia uma thread para receber mensagens do servidor
175             new Thread(() -> {
176                 while (true) {
177                     try {
178                         String string = serverInput.readUTF();
179                         updateInterface(string);
180                     } catch (IOException e) {
181                         updateInterface("Problema de conex o...");
182                         closeConnection();
183                         break;
184                     }
185                 }
186             }).start();
187         } catch (IOException e) {
188             e.printStackTrace();
189         }
190     }
191 
192     /**
193      * Atualiza a interface gr fica com a mensagem recebida.
194      * @param message Mensagem recebida do servidor.
195      */
196     private static void updateInterface(String message) {
197         SwingUtilities.invokeLater(() -> textArea.append("\n" + message));
198     }
199 
200     /**
201      * Envia a mensagem digitada para o servidor.
202      */
203     private void sendMessage() {
204         if (con != null && con.isConnected()) {
205             String message = textField.getText();
206             if (!message.isEmpty()) {
207                 updateInterface(clientName + ": " + message);
208                 try {
209                     output.writeUTF(message);
210                 } catch (IOException e) {
211                     updateInterface("Problema de conex o...");
212                     closeConnection();
```

```
213                 }
214                 textField.setText("");
215             }
216         } else {
217             JOptionPane.showMessageDialog(mainFrame, "Conex o perdida.
    Reconecte-se!");
218             closeConnection();
219         }
220     }
221 
222     /**
223      * Solicita a conex o direta a outro usu rio.
224      */
225     private void requestDirectConnection() {
226         String targetClient = JOptionPane.showInputDialog(mainFrame, "
    Digite o nome do Usuario para conectar diretamente:");
227         if (targetClient != null && !targetClient.isEmpty()) {
228             connectToClient(targetClient);
229         }
230     }
231 
232     /**
233      * Conecta-se diretamente a outro usu rio.
234      * @param targetClient Nome do usu rio alvo.
235      */
236     private void connectToClient(String targetClient) {
237         if (con != null && con.isConnected()) {
238             try {
239                 output.writeUTF("/connect " + targetClient);
240             } catch (IOException e) {
241                 updateInterface("Problema de conex o...");
242                 closeConnection();
243             }
244         } else {
245             JOptionPane.showMessageDialog(mainFrame, "Conex o perdida.
    Reconecte-se!");
246             closeConnection();
247         }
248     }
249 
250     /**
251      * Fecha a conex o com o servidor.
252      */
253     private void closeConnection() {
254         if (con != null && !con.isClosed()) {
255             try {
256                 con.close();
257             } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
        //System.exit(0);
    }

    /**
     * Aplica um estilo de fonte a um componente Swing.
     * @param component Componente Swing a ser estilizado.
     * @param fontSize Tamanho da fonte.
     */
    private void stylizeComponent(JComponent component, int fontSize) {
        component.setFont(new Font("Arial", Font.PLAIN, fontSize));
    }

    /**
     * Aplica um estilo de fonte e estilo a um componente Swing.
     * @param component Componente Swing a ser estilizado.
     * @param style Estilo da fonte (e.g., Font.BOLD).
     * @param fontSize Tamanho da fonte.
     */
    private void stylizeComponent(JComponent component, int style, int
    fontSize) {
        component.setFont(new Font("Arial", style, fontSize));
    }
}
```