NEWTON PAIVA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LUCAS AGENOR, MAURICIO SADALA, MATHEUS AUGUSTO, PAULO HAMILTON E VINICIUS EVERTON

**MINI SIMULADOR DE REDE SOCIAL**

BELO HORIZONTE – MG

2023

LUCAS AGENOR, MAURICIO SADALA, MATHEUS AUGUSTO, PAULO HAMILTON, VINICIUS EVERTON

**MINI SIMULADOR DE REDE SOCIAL**

Trabalho final da disciplina de Programação Orientada a Objetos, apresentado a Universidade Newton Paiva.

Professor(a): Michelle Hanne

BELO HORIZONTE – MG

2023

Sumário

[1. INTRODUÇÃO: 4](#_Toc137542179)

[1.2. Estrutura de dados: 4](#_Toc137542180)

[1.3. UML – Diagrama de classes: 4](#_Toc137542181)

[2. DESENVOLVIMENTO: 5](#_Toc137542182)

[2.2. Classe abstrata: 5](#_Toc137542183)

[2.3. Herança: 5](#_Toc137542184)

[2.4. Interface: 5](#_Toc137542185)

[2.5. Polimorfismo: 6](#_Toc137542186)

[2.6. Explicações das classes: 6](#_Toc137542187)

[2.6.1. Classe User: 6](#_Toc137542188)

[2.6.2. Interface SocialNetWork: 9](#_Toc137542189)

[2.6.3. Classe SocialNeworkImpl: 11](#_Toc137542190)

[2.6.4. Classe DatabaseConnection: 15](#_Toc137542191)

[2.6.5. Classe Main: 16](#_Toc137542192)

[3. CONCLUSÃO 21](#_Toc137542193)

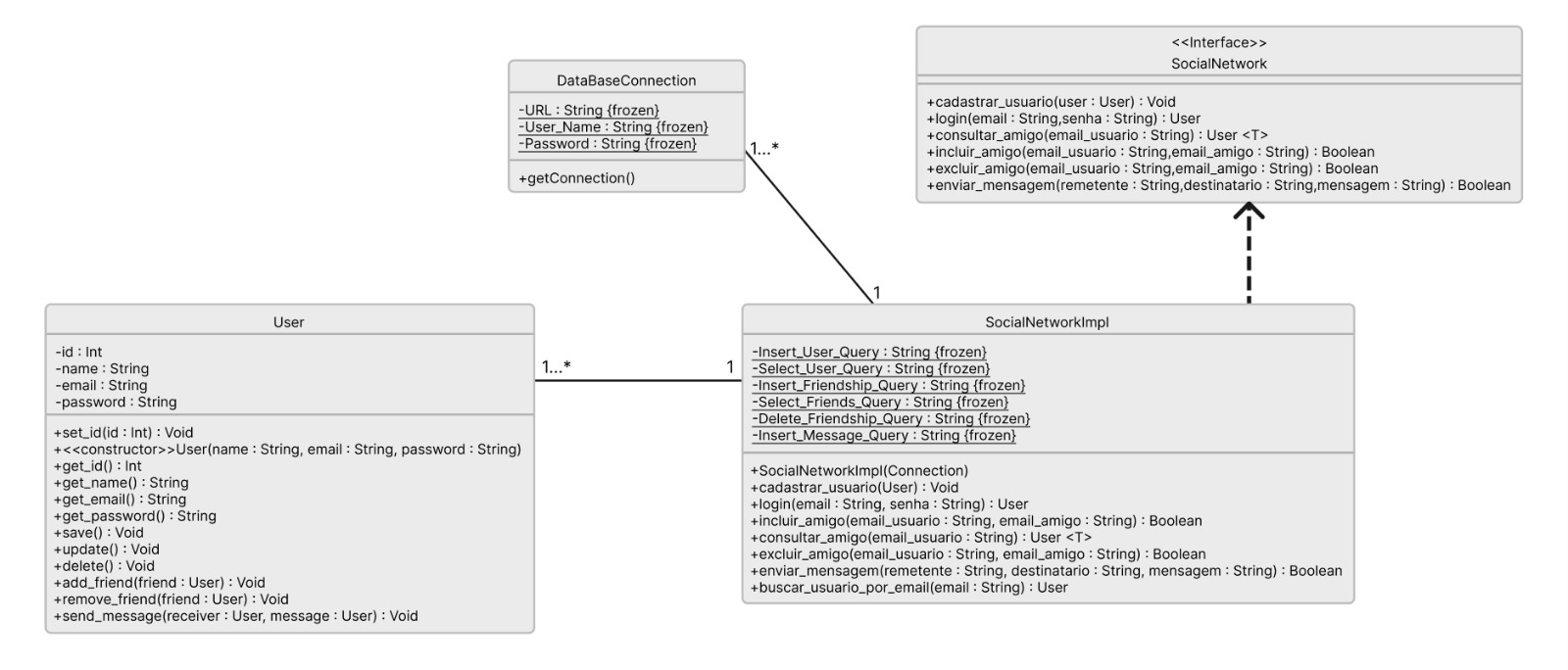
# INTRODUÇÃO:

O problema abordado neste projeto é a implementação de um Mini Simulador de Rede Social. O objetivo é criar um sistema capaz de cadastrar usuários, gerenciar amizades entre eles e permitir o envio de mensagens. O sistema deve ser capaz de lidar com um número indefinido de usuários.

## Estrutura de dados:

A estrutura de dados utilizada nesse projeto é baseada em classes, com foco na orientação a objetos. A classe principal é o “User”, que representa um usuário da rede social. Cada usuário possui um nome, e-mail, senha e uma lista de amigos. A lista de amigos é uma coleção de objetos do tipo “User”, permitindo a associação de um número indefinido de amigos a cada usuário.

## UML – Diagrama de classes:

A seguir, o diagrama de classes do Mini Simulador de Rede social:

# DESENVOLVIMENTO:

No desenvolvimento do Mini simulador de Rede Social, a utilização dos conceitos de classe abstrata, herança, interface e polimorfismo é fundamental para criar uma estrutura flexível, modular e de fácil manutenção. Segue a explicação desses conceitos:

## Classe abstrata:

É utilizada para definir uma estrutura comum e compartilhar comportamentos entre classes relacionadas. No caso do simulador de rede social, a classe “User” é utilizada para definir atributos e métodos básicos que são comuns a todos os usuários, como nome, e-mail, senha e a lista de amigos. Ao utilizar uma classe abstrata, seria possível criar novas classes que herdam essas características comuns e, ao memo tempo, implementam comportamentos específicos.

## Herança:

É utilizada para estabelecer uma relação de “é um” entre classes, onde uma classe mais específica herda características e comportamentos de uma classe mais genérica. No simulador de rede social, a classe “User” é a classe base de outras classes, como “RegisteredUser” e “AdminUser”, que podem ser criadas para representar tipos específicos de usuários. A herança permite reutilizar código e estabelecer hierarquias de classes, facilitando a organização e a manutenção do código.

## Interface:

A interface é utilizada para definir um conjunto de métodos que uma classe deve implementar. No simulador de rede social, a interface “SocialNetwork” define os métodos que devem ser implementados pelas classes que gerenciam a rede social, como cadastrar usuários, gerenciar amizades e enviar mensagens. O uso de interfaces permite a criação de classes diferentes que implementam esses métodos de maneiras distintas, mas compatíveis com a interface, fornecendo uma maior flexibilidade e possibilitando a substituição de implementações.

## **Polimorfismo**:

É a capacidade de um objeto ser tratado de diferentes maneiras, dependendo do contexto em que é utilizado. No contexto do mini simulador de rede social, o polimorfismo é aplicado quando se utiliza referências da classe base (User) para manipular objetos de classes derivadas. Isso permite tratar todos os usuários de uma forma uniforme, independente do tipo específico de usuário. O polimorfismo simplifica a lógica do programa, tornando-o mais flexível e facilitando futuras modificações.

Em síntese, o uso de classe abstrata, herança, interface e polimorfismo no desenvolvimento do projeto foi fundamental para criar uma estrutura modular, flexível e de fácil manutenção. Esses conceitos permitem reutilizar o código, estabelecer hierarquias de classes, definir comportamentos comuns e possibilitar a substituição de implementações. A seguir, será explicado os principais métodos de cada classe:

## Explicações das classes:

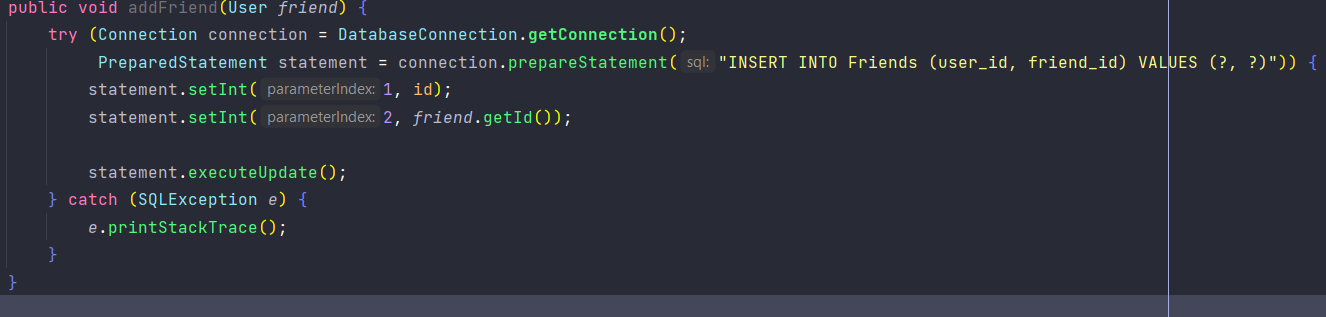
### Classe User:

Método “addFriend(User Friend)”: é responsável por adicionar um amigo à lista de amigos do usuário atual.

Primeiro, ele obtém uma conexão com o banco de dados usando o método getConnection() da classe DatabaseConnection. Em seguida, prepara uma declaração SQL para inserir um novo registro na tabela "Friends" utilizando a instrução INSERT INTO. Os valores a serem inseridos são definidos por meio de parâmetros na declaração preparada.O parâmetro id corresponde ao ID do usuário atual, enquanto o parâmetro friend.getId() corresponde ao ID do amigo a ser adicionado. Esses valores são definidos na declaração preparada usando os métodos setInt().

Após a definição dos parâmetros, o método executeUpdate() é chamado na declaração preparada para executar a instrução SQL e inserir o registro na tabela.

Em caso de exceção do tipo SQLException, o método printStackTrace() é chamado para imprimir informações detalhadas sobre a exceção ocorrida.

Dessa forma, o método addFriend(User friend) permite adicionar um amigo à lista de amigos do usuário atual no banco de dados.

Método “removeFriend(User Friend)”: é responsável por remover um amigo da lista de amigos do usuário atual.

Primeiro, ele obtém uma conexão com o banco de dados usando o método getConnection() da classe DatabaseConnection. Em seguida, prepara uma declaração SQL para excluir um registro da tabela "Friends" utilizando a instrução DELETE FROM. A condição para exclusão é especificada no WHERE, onde o ID do usuário atual deve corresponder ao parâmetro user\_id e o ID do amigo a ser removido deve corresponder ao parâmetro friend\_id.

Os valores dos parâmetros são definidos na declaração preparada usando os métodos setInt(). O parâmetro id corresponde ao ID do usuário atual, enquanto o parâmetro friend.getId() corresponde ao ID do amigo a ser removido.

Após a definição dos parâmetros, o método executeUpdate() é chamado na declaração preparada para executar a instrução SQL e remover o registro da tabela.

Em caso de exceção do tipo SQLException, o método printStackTrace() é chamado para imprimir informações detalhadas sobre a exceção ocorrida.

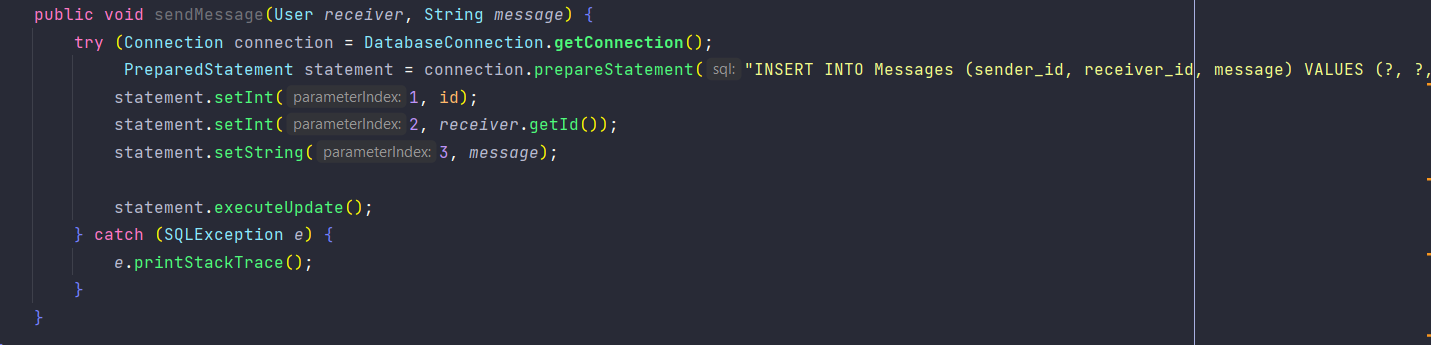
Método “sendMessage(User receiver, String message)”: é responsável por enviar uma mensagem de um usuário atual para outro usuário especificado como receptor. A mensagem consiste em uma string de texto.

O método utiliza uma conexão com o banco de dados para inserir a mensagem na tabela "Messages" do banco de dados. A inserção é feita por meio de uma consulta SQL preparada, que é construída usando um objeto PreparedStatement. A consulta SQL contém três parâmetros: "sender\_id" (identificador do remetente), "receiver\_id" (identificador do destinatário) e "message" (texto da mensagem). A conexão com o banco de dados e o PreparedStatement são criados em um bloco try-with-resources. Isso garante que os recursos sejam devidamente fechados, independentemente de ocorrer uma exceção ou não.

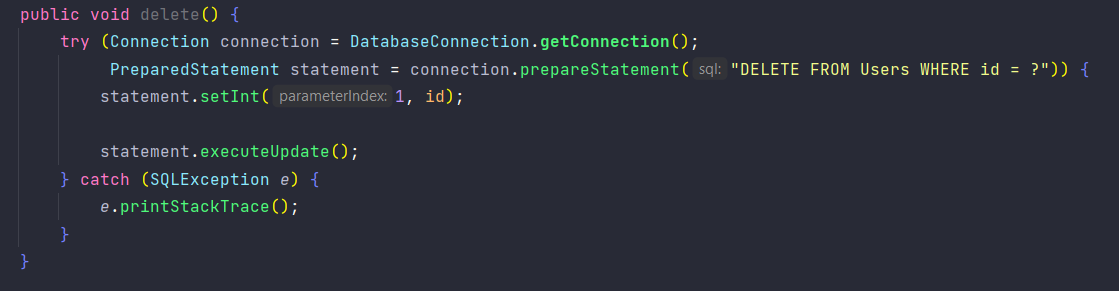
Dentro do bloco try, são definidos os valores dos parâmetros da consulta SQL usando os métodos setInt e setString do PreparedStatement. O valor do parâmetro "sender\_id" é definido como o ID do usuário atual (que é acessado através da variável id), o valor do parâmetro "receiver\_id" é definido como o ID do receptor (obtido através do método getId() do objeto User recebido como argumento) e o valor do parâmetro "message" é definido como a mensagem fornecida como argumento. Em seguida, o método executeUpdate() é chamado no PreparedStatement para executar a consulta SQL e inserir a mensagem no banco de dados.

Se ocorrer uma exceção do tipo SQLException, o bloco catch captura a exceção e imprime o rastreamento de pilha no console usando o método printStackTrace(). Esse tratamento de exceção é útil para depuração e identificação de problemas durante o acesso ao banco de dados.

No geral, o método sendMessage encapsula a lógica para inserir uma mensagem no banco de dados, fornecendo uma interface conveniente para a funcionalidade de envio de mensagens na aplicação.



Método “public void delete()”: é responsável por excluir um usuário da rede social. Ele é chamado em um objeto do tipo User para remover todas as informações relacionadas a esse usuário do sistema.



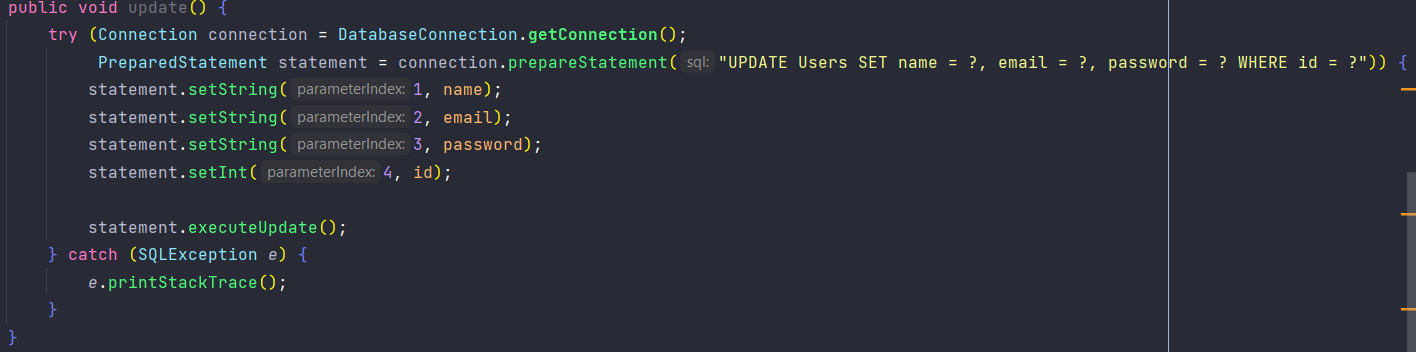
Método “public void update()”: é responsável por atualizar os dados de um usuário no banco de dados. Ele atualiza as informações de nome, email e senha do usuário com base no ID do usuário. O método utiliza uma conexão com o banco de dados para executar uma consulta SQL preparada que atualiza os campos relevantes na tabela "Users".

A conexão com o banco de dados e o PreparedStatement são criados em um bloco try-with-resources, garantindo que os recursos sejam fechados corretamente, independentemente de ocorrer uma exceção ou não.

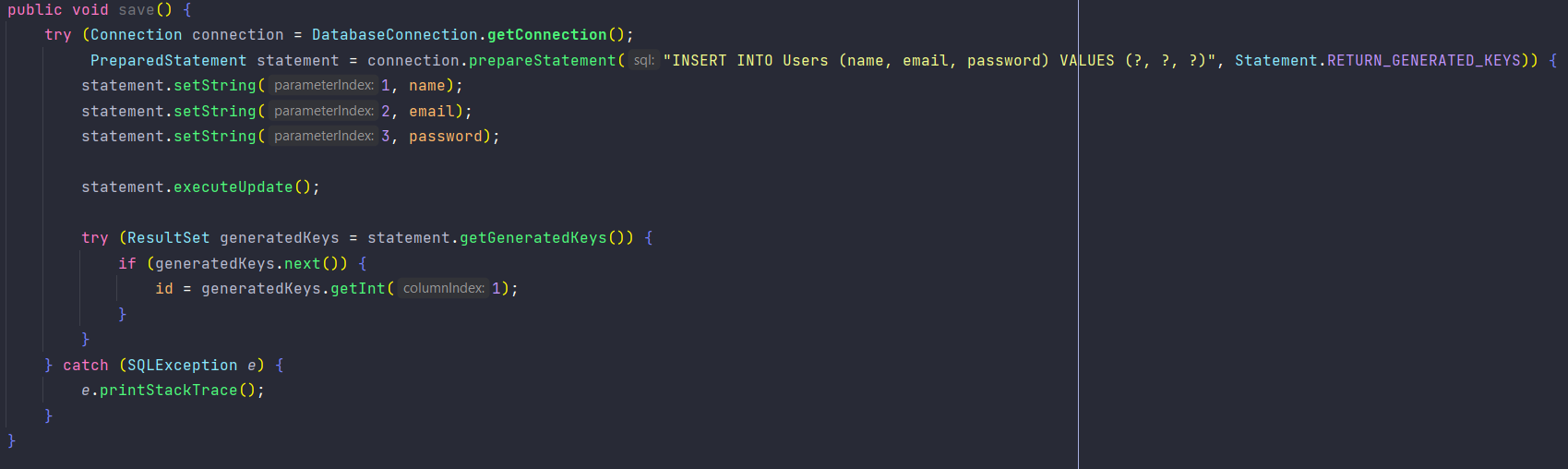
Dentro do bloco try, são definidos os valores dos parâmetros da consulta SQL usando os métodos setString e setInt do PreparedStatement. O valor do parâmetro "name" é definido como o novo nome do usuário (acessado pela variável name), o valor do parâmetro "email" é definido como o novo email do usuário (acessado pela variável email), o valor do parâmetro "password" é definido como a nova senha do usuário (acessada pela variável password) e o valor do parâmetro "id" é definido como o ID do usuário (acessado pela variável id).

Em seguida, o método executeUpdate() é chamado no PreparedStatement para executar a consulta SQL e atualizar os dados do usuário no banco de dados.

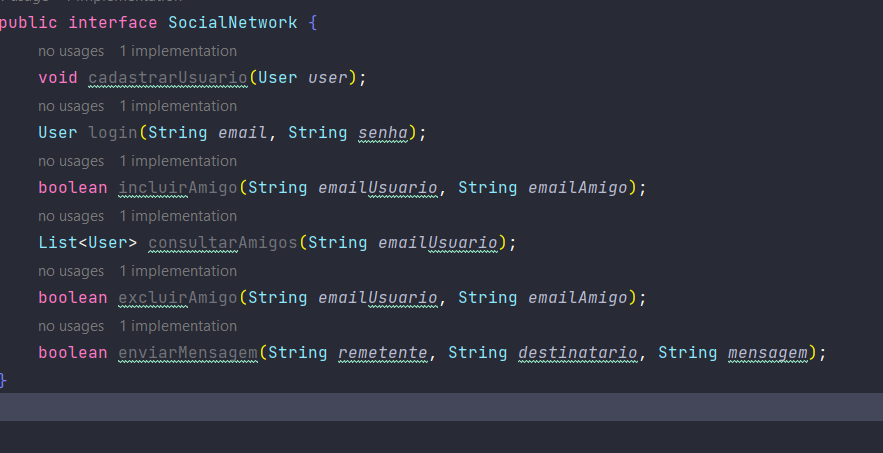
Se ocorrer uma exceção do tipo SQLException, o bloco catch captura a exceção e imprime o rastreamento de pilha no console usando o método printStackTrace(). Isso auxilia na depuração e identificação de problemas durante a atualização dos dados no banco de dados.



Método “public void save()”: é responsável por salvar um novo usuário na rede social. Ele é chamado em um objeto do tipo User para armazenar as informações do usuário no sistema.



### Interface SocialNetWork:



A interface SocialNetwork define um conjunto de métodos que representam as funcionalidades básicas de uma rede social. Vamos analisar cada um desses métodos em detalhes:

“void registerUser(User user)”: Este método é responsável por cadastrar um novo usuário na rede social. Ele recebe um objeto do tipo User como parâmetro, representando o usuário a ser registrado. A implementação desse método pode incluir a validação dos dados do usuário e a persistência dessas informações em um banco de dados ou outra forma de armazenamento.

“void logout()”: Esse método é responsável por realizar o logout do usuário da rede social. Ele não recebe parâmetros. A implementação desse método pode envolver a limpeza das informações de autenticação do usuário ou a atualização do status de login no sistema.

“List<User> getFriends(User user)”: Este método retorna uma lista de amigos de um usuário na rede social. Ele recebe um objeto do tipo User como parâmetro e retorna uma lista de objetos User representando os amigos desse usuário. A implementação desse método deve recuperar as informações dos amigos do usuário a partir do nosso banco de dados.

“void addFriend(User user, User friend)”: Este método permite adicionar um amigo para um usuário na rede social. Ele recebe dois objetos do tipo User: o usuário atual e o amigo a ser adicionado. A implementação desse método pode envolver a atualização das listas de amigos dos usuários envolvidos ou a persistência dessas informações no nosso banco de dados.

“void removeFriend(User user, User friend)”: Esse método permite remover um amigo da lista de amigos de um usuário. Ele recebe dois objetos do tipo User: o usuário atual e o amigo a ser removido. A implementação desse método pode envolver a exclusão da relação de amizade entre os usuários ou a atualização das listas de amigos.

“void sendMessage(User user, User friend, String message)”: Esse método permite que um usuário envie uma mensagem para outro usuário. Ele recebe três parâmetros: o remetente da mensagem (objeto User), o destinatário da mensagem (objeto User) e a mensagem em si (como uma String). A implementação desse método pode envolver o envio da mensagem por meio de um sistema de mensagens interno da rede social.

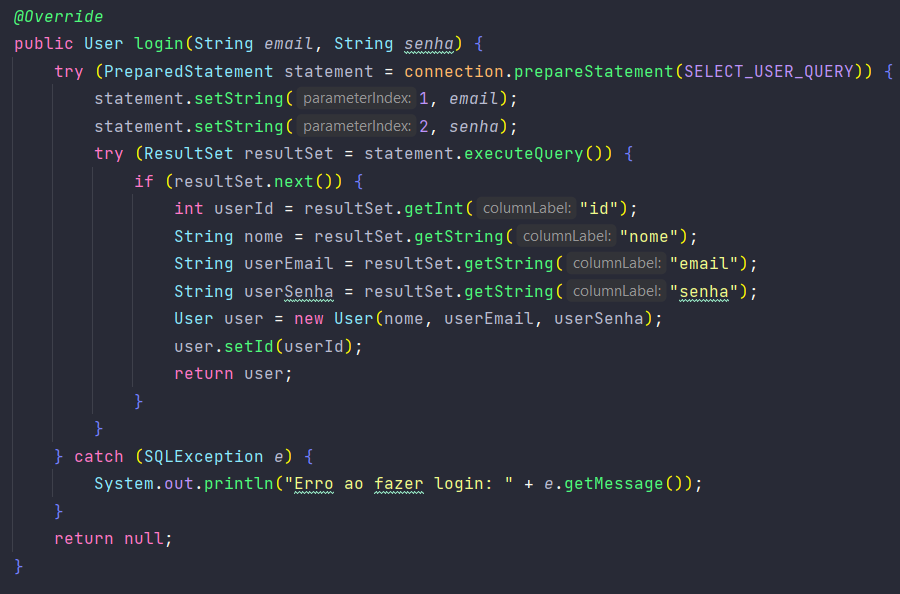
### Classe SocialNeworkImpl:

A classe SocialNetworkImpl implementa a interface SocialNetwork e fornece a implementação dos métodos definidos nessa interface.

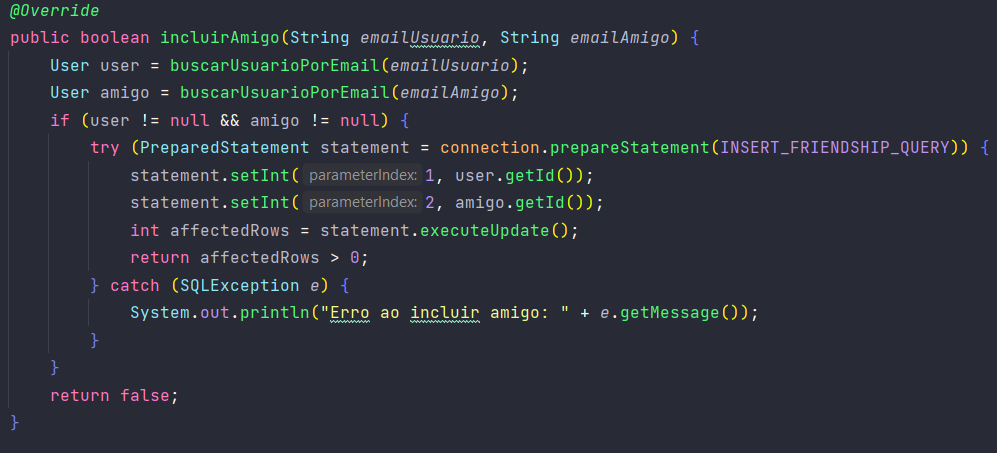
“cadastrarUsuario(User user)”: Esse método recebe um objeto do tipo User como parâmetro e realiza o cadastro desse usuário no banco de dados. Ele utiliza uma consulta preparada (PreparedStatement) para inserir os dados do usuário na tabela users. Após a execução da consulta, o método verifica se algum registro foi afetado e, em caso positivo, obtém o ID gerado automaticamente e o atribui ao objeto User.



“login(String email, String senha)”: Esse método realiza a autenticação de um usuário com base no email e senha fornecidos. Ele utiliza uma consulta preparada para selecionar os dados do usuário correspondente na tabela users. Se a consulta retornar algum resultado, o método cria um objeto User com as informações recuperadas e retorna esse objeto. Caso contrário, retorna null.



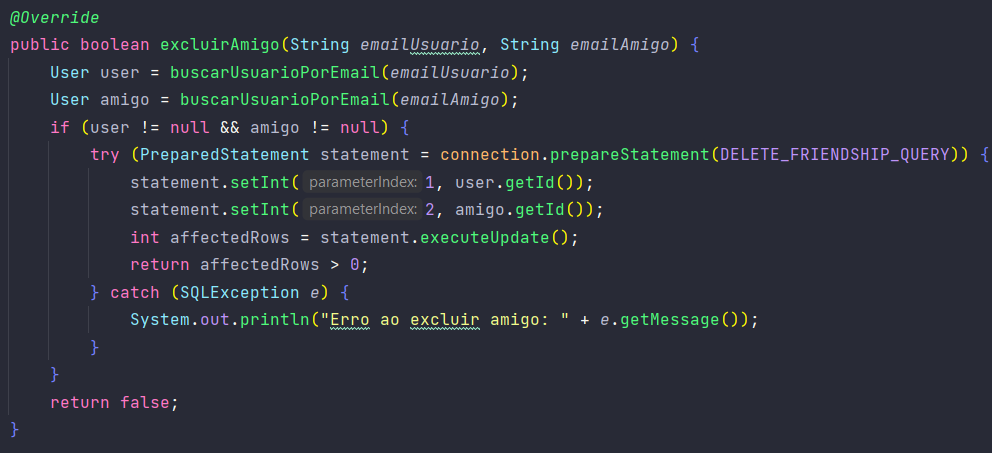
“incluirAmigo(String emailUsuario, String emailAmigo)”: Esse método permite incluir um amigo para um determinado usuário. Ele busca os objetos User correspondentes aos emails fornecidos e, se ambos existirem, utiliza uma consulta preparada para inserir a relação de amizade na tabela friendships. O método retorna true se a inserção for bem-sucedida e false caso contrário.



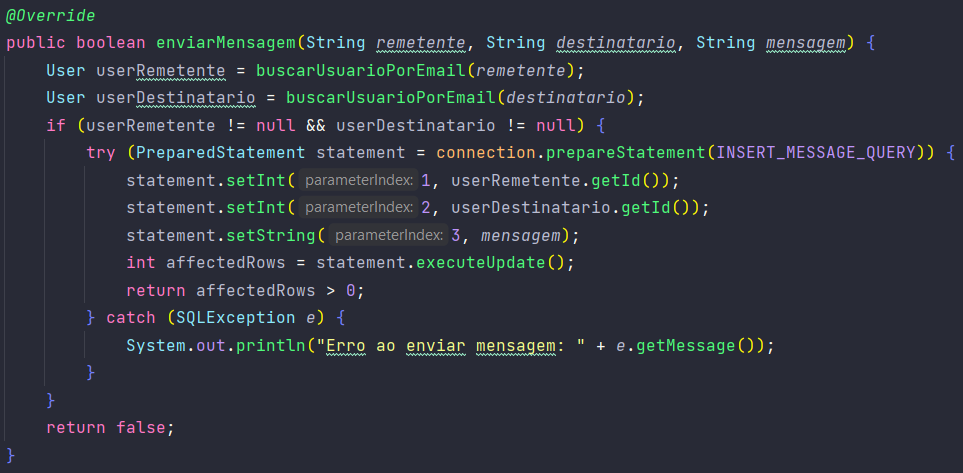
“consultarAmigos(String emailUsuario)”: Esse método retorna uma lista de amigos de um usuário com base no email fornecido. Ele busca o objeto User correspondente ao email fornecido e, se existir, utiliza uma consulta preparada para selecionar os amigos desse usuário na tabela users por meio de um JOIN com a tabela friendships. O método cria objetos User para cada amigo recuperado e os adiciona a uma lista, que é retornada ao final.



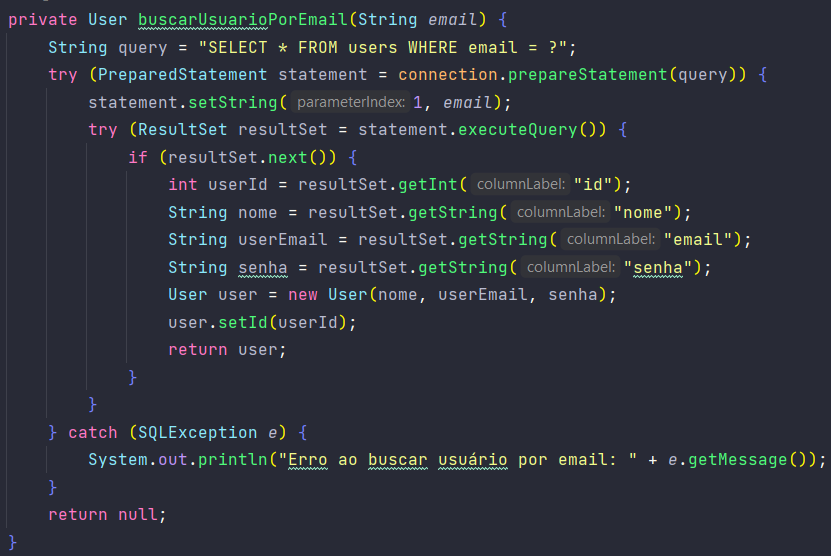
“excluirAmigo(String emailUsuario, String emailAmigo)”: Esse método permite excluir um amigo da lista de amigos de um usuário. Ele busca os objetos User correspondentes aos emails fornecidos e, se ambos existirem, utiliza uma consulta preparada para excluir a relação de amizade da tabela friendships. O método retorna true se a exclusão for bem-sucedida e false caso contrário.



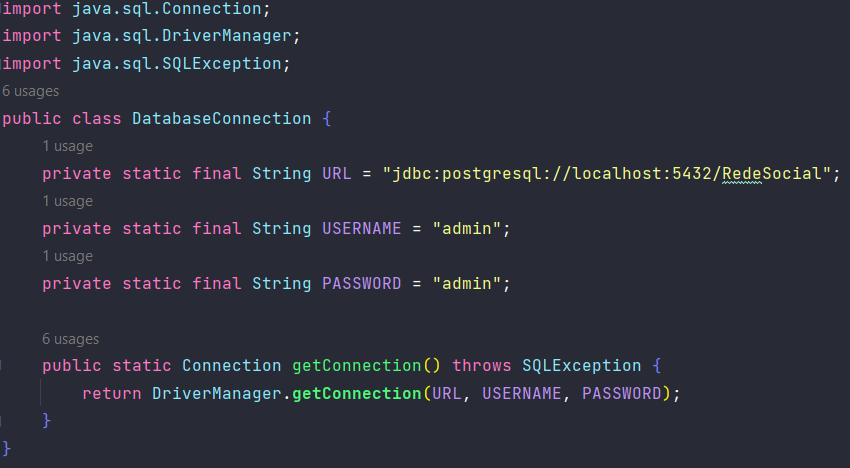
“enviarMensagem(String remetente, String destinatario, String mensagem)”: Esse método permite enviar uma mensagem de um remetente para um destinatário. Ele busca os objetos User correspondentes aos emails fornecidos e, se ambos existirem, utiliza uma consulta preparada para inserir a mensagem na tabela messages. O método retorna true se o envio for bem-sucedido e false caso contrário.



“buscarUsuarioPorEmail(String email)”: Esse método é um método auxiliar que busca um usuário no banco de dados com base no email fornecido. Ele utiliza uma consulta preparada para selecionar os dados do usuário correspondente na tabela users. Se a consulta retornar algum resultado, o método cria um objeto User com as informações recuperadas e o retorna. Caso contrário, retorna null.



### Classe DatabaseConnection:



A classe DatabaseConnection é responsável por estabelecer a conexão com o banco de dados PostgreSQL.

“getConnection()”: Esse método estático retorna uma instância de Connection que representa a conexão com o banco de dados. Ele utiliza o DriverManager para obter a conexão com base na URL, nome de usuário e senha fornecidos. O método lança uma exceção SQLException se ocorrer algum erro durante o processo de conexão.

Essa classe fornece um meio simples de obter uma conexão com o banco de dados usando as informações de URL, nome de usuário e senha definidas na classe. Ela encapsula a lógica de estabelecimento da conexão em um método estático, facilitando o uso em outras partes do código.

### Classe Main:

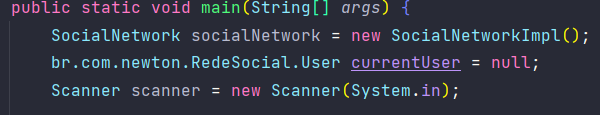
A classe Main é a classe principal do programa que representa um simulador básico de uma rede social. Vamos analisar seus principais componentes e funcionalidades:

Declaração de variáveis:

SocialNetwork socialNetwork: Declara uma variável do tipo SocialNetwork para representar a rede social.

br.com.newton.RedeSocial.User currentUser: Declara uma variável do tipo User para representar o usuário atualmente logado.

Scanner scanner: Declara uma variável do tipo Scanner para ler as entradas do usuário.



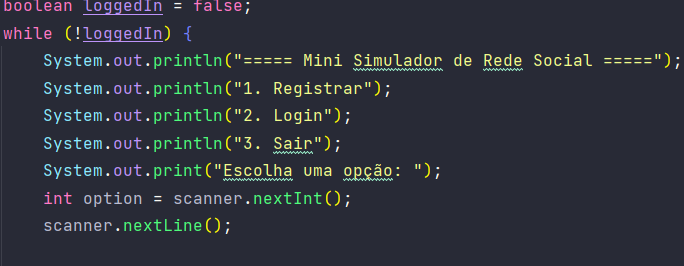
Loop de autenticação:

Um loop while é usado para autenticar o usuário até que o login seja bem-sucedido ou o usuário escolha sair.

O usuário pode escolher entre as opções de registrar, fazer login ou sair.

O usuário fornece as informações necessárias para registro ou login, e as operações correspondentes são executadas usando os métodos da classe SocialNetwork.

Se o login for bem-sucedido, a variável loggedIn é definida como true para sair do loop e continuar para o próximo menu.



Loop do menu principal:

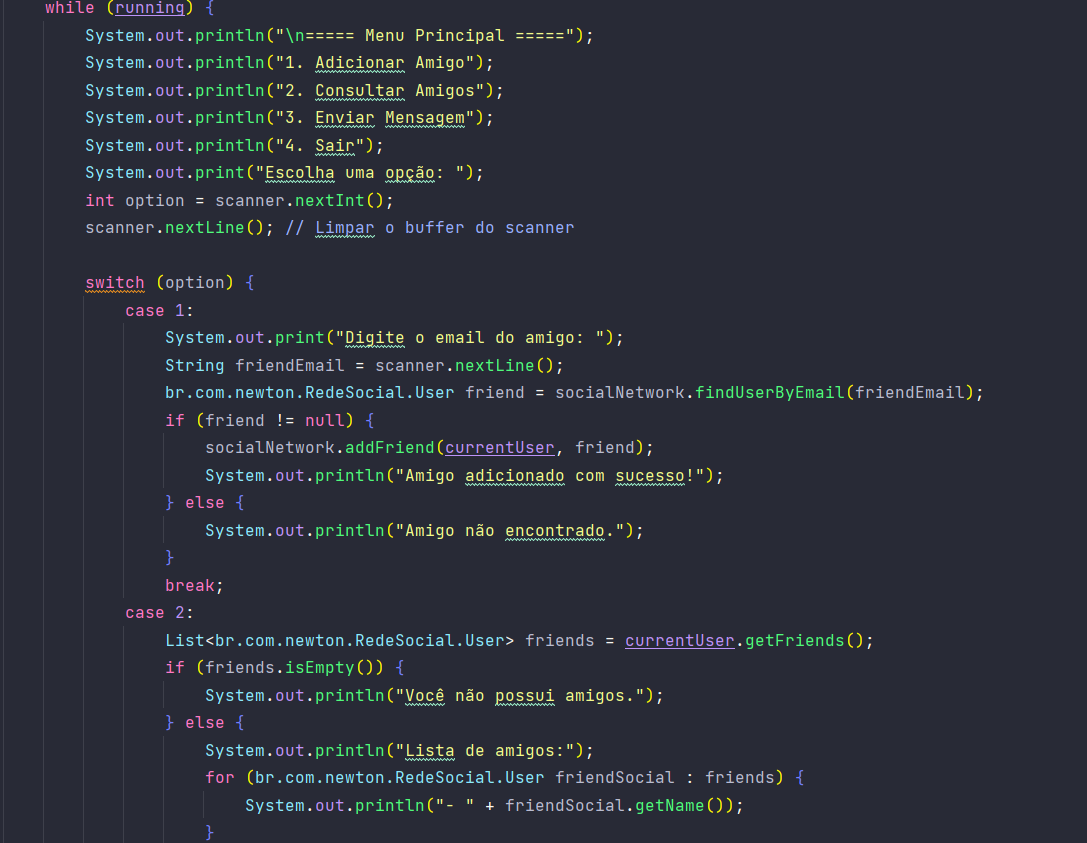
Um loop while é usado para exibir o menu principal e executar as operações escolhidas pelo usuário.

O usuário pode escolher entre as opções de adicionar amigo, consultar amigos, enviar mensagem ou sair.

Com base na escolha do usuário, as operações correspondentes são executadas usando os métodos da classe SocialNetwork e as informações são exibidas no console.

Se o usuário escolher sair, a variável running é definida como false para sair do loop e encerrar o programa.







Em resumo, a classe Main coordena a interação com o usuário, chamando os métodos apropriados da classe SocialNetwork para registrar usuários, fazer login, adicionar amigos, consultar amigos e enviar mensagens. Ela oferece uma interface básica para simular as funcionalidades de uma rede social.

# CONCLUSÃO

Durante o desenvolvimento deste projeto de um Mini Simulador de Rede Social, o grupo enfrentou desafios significativos na implementação da parte gráfica utilizando o JavaFX. Apesar dos esforços empreendidos, não foi possível concluir a implementação da interface gráfica devido a dificuldades técnicas e limitações de conhecimento.

No entanto, é importante ressaltar que, mesmo sem a parte gráfica, a estrutura do projeto e as funcionalidades principais da rede social foram implementadas com sucesso. Através da classe Main e das demais classes envolvidas, é possível cadastrar usuários, realizar login, adicionar amigos, consultar a lista de amigos e enviar mensagens.

Reconhecemos que a interface gráfica desempenha um papel fundamental na melhoria da usabilidade e da experiência do usuário. Por isso, continuaremos estudando e com o tempo, adquirindo conhecimento adicional sobre o JavaFX e adicionando a camada visual para tornar a rede social mais atraente e intuitiva.

Também é importante mencionar que optamos por não utilizar uma classe abstrata no projeto. Embora a classe abstrata seja uma ferramenta poderosa para compartilhar comportamentos comuns entre classes relacionadas, nossa abordagem foi focada na criação de interfaces claras e contratos definidos por meio da interface SocialNetwork.

Embora não tenhamos alcançado todos os objetivos estabelecidos inicialmente, consideramos esse projeto uma oportunidade valiosa de aprendizado e desenvolvimento de habilidades. Através dos desafios enfrentados, adquirimos conhecimento sobre a estruturação de um projeto de software e a importância da colaboração em equipe.

Em suma, mesmo com as dificuldades encontradas, o projeto proporcionou uma base sólida para a construção de uma rede social, e a falta de implementação da parte gráfica e a abstração não invalida os esforços e conhecimentos adquiridos ao longo do desenvolvimento.