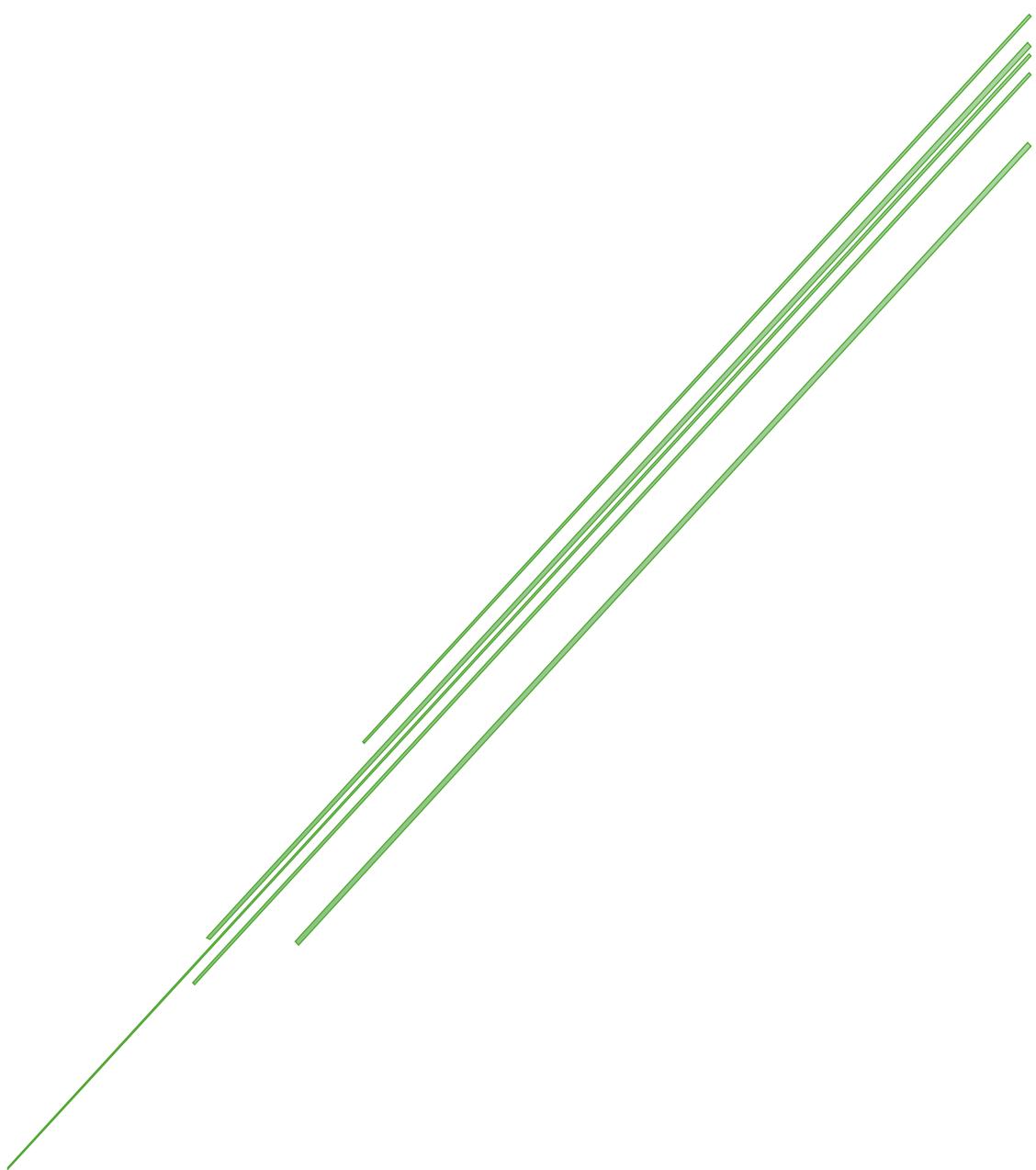


SUPER CRUD COM JDBC

Java



Sumário

CRUD em memória	2
Criação do banco de dados e tabelas	4
Criação do projeto Java	4
JDBC (Java Database Connectivity)	9
O que é um driver JDBC	9
Para que serve um driver JDBC?	9
Como usar um driver JDBC	10
Principais classes.....	11
Implementação do Gerenciar	11
PreparedStatement	12
Salvar	13
Atualizar.....	14
Listar	14
Remover.....	16
FindById.....	16
PesquisarPorNome	17
Fechar.....	17
Código completo da classe GerenciarUsuario:	18
ProgramaPrincipal.....	22
Cadastrar.....	24
Alterar	25
Listar	26
Remover.....	27
Código de “ProgramaPrincipal”	28
Ajustes finais	30
Números mágicos	37
Código final completo da classe “ProgramaPrincipal”	43
GitHub	45

CRUD em memória

CRUD é um acrônimo que representa as quatro operações básicas realizadas em bancos de dados e sistemas de informação:

- **C – Create** (Criar): Inserir novos dados no sistema.
- **R – Read** (Ler): Consultar ou visualizar dados existentes.
- **U – Update** (Atualizar): Modificar dados já existentes.
- **D – Delete** (Deletar): Remover dados do sistema.

Exemplo simples:

Imagina um sistema de cadastro de usuários. O CRUD seria:

- **Create:** Cadastrar um novo usuário.
- **Read:** Listar os usuários cadastrados.
- **Update:** Alterar os dados de um usuário (como o nome ou e-mail).
- **Delete:** Excluir um usuário do sistema.

Importância do CRUD no desenvolvimento de sistemas:

1. **Base para qualquer aplicação:** A maioria dos sistemas precisa gerenciar dados (clientes, produtos, pedidos, etc.), e o CRUD é a forma básica de fazer isso.
2. **Organização do código:** Seguir essa estrutura ajuda a manter o código organizado, com funções bem definidas para cada operação.
3. **Facilidade de manutenção:** Quando o sistema é desenvolvido com base em operações CRUD, fica mais fácil fazer ajustes e melhorias no futuro.
4. **Interface com banco de dados:** O CRUD representa as interações fundamentais com qualquer banco de dados relacional ou não relacional.
5. **Reutilização e padronização:** Como é uma prática comum, os desenvolvedores conseguem entender e trabalhar em sistemas de outras pessoas com mais facilidade.

O CRUD está diretamente **relacionado ao banco de dados**, porque ele define as **quatro operações básicas** que você faz **com os dados armazenados** lá.

CRUD no contexto do banco de dados:

1. CREATE (Criar):

- Operação que insere novos registros no banco.
- Exemplo SQL:

```
INSERT INTO tb_usuarios (nome, email) VALUES ('Maria', 'maria@email.com');
```

2. READ (Ler):

- o Consulta os dados no banco, seja para exibir ou processar.
- o Exemplo SQL:

```
SELECT * FROM tb_usuarios;
```

3. UPDATE (Atualizar):

- o Modifica dados existentes.
- o Exemplo SQL:

```
UPDATE tb_usuarios SET nome = 'Maria Silva' WHERE id = 1;
```

4. DELETE (Deletar):

- o Remove registros do banco de dados.
- o Exemplo SQL:
- o

```
DELETE FROM tb_usuarios WHERE id = 1;
```

Resumindo:

O CRUD é como o "vocabulário básico" de comunicação com um banco de dados. Toda vez que um sistema precisa armazenar, mostrar, alterar ou apagar dados, ele faz isso através de operações CRUD.

Essas operações podem ser feitas manualmente (com SQL, por exemplo) ou através de linguagens de programação (como Python, JavaScript, Java) que se conectam ao banco.

Dada essa introdução, vamos iniciar nosso projeto.

Criação do banco de dados e tabelas

Primeiro crie o banco de dados:

```
create database db_ifpr;
```

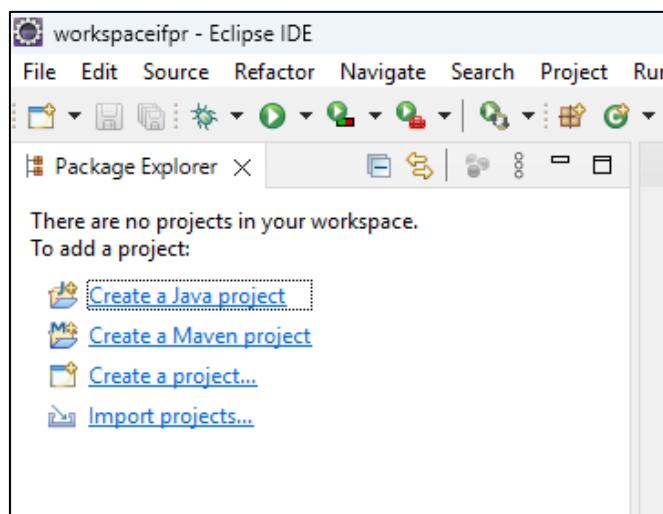
Crie a tabela usuário:

```
use db_ifpr;

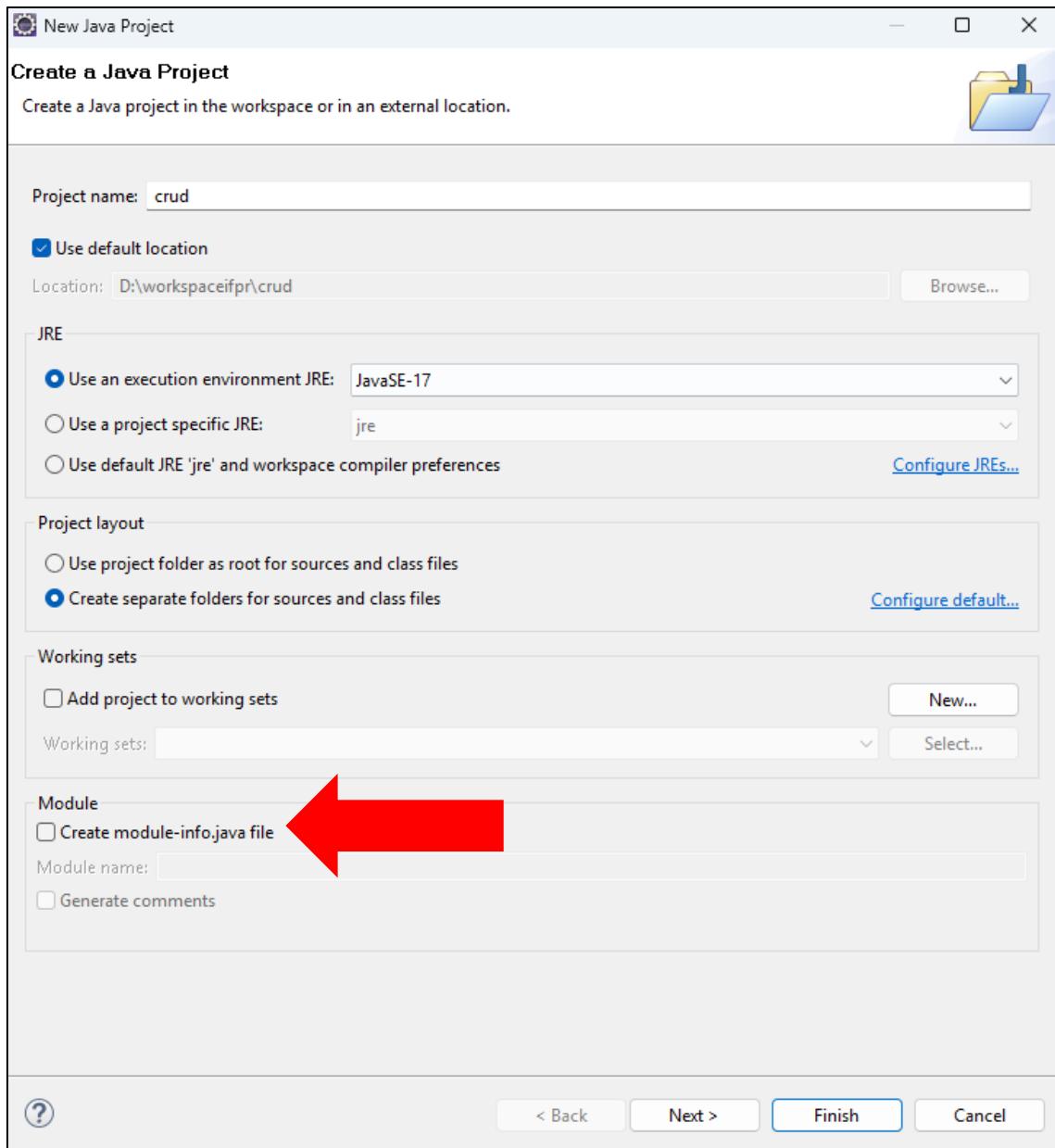
CREATE TABLE tb_usuario (
    id INT AUTO_INCREMENT,
    nome VARCHAR(255),
    cpf VARCHAR(255),
    data_nascimento DATE,
    CONSTRAINT id_pk PRIMARY KEY (id)
);
```

Criação do projeto Java

O primeiro passo é criar um projeto java:



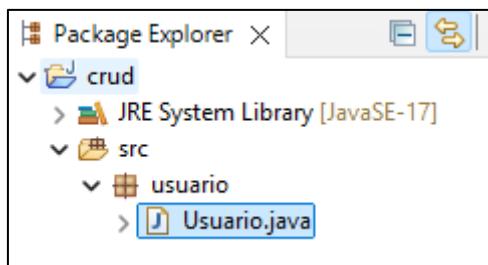
Coloque o nome do projeto como “crud” e desmarque a caixa “Create module-info.java file” e clique em “Finish”:



Vamos fazer o crud de uma classe com 4 atributos:

- Usuario (id, nome, cpf, dataNascimento)

Crie um pacote chamado “usuario” e dentro crie a classe “Usuario”:



```
package usuario;

import java.time.LocalDate;

public class Usuario {

    private int id;
    private String nome;
    private String cpf;
    private LocalDate dataNascimento;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

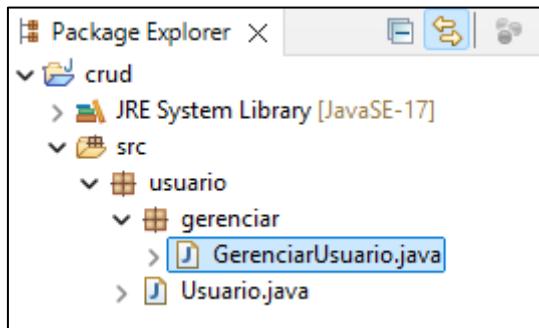
    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public LocalDate getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(LocalDate dataNascimento) {
        this.dataNascimento = dataNascimento;
    }
}
```

Dentro do pacote “usuario” crie outro pacote com o nome “gerenciar”. Dentro do pacote “gerenciar” crie uma classe com o nome “GerenciarUsuario”. Essa classe será responsável por manipular a entidade no banco de dados.



Agora vamos criar os seguintes métodos dentro da classe “GerenciarUsuario”:

- **public void salvar(Usuario usuario)**
- **public void atualizar(Usuario usuário)**
- **public List<Usuario> listar()**
- **public void remover(int id)**

Repare que esses métodos representam o CRUD:

- salvar (C – create)
- listar (R – read)
- atualizar (U – update)
- remover (D – delete)

Além dos métodos do CRUD, vamos colocar mais dois métodos auxiliares:

- **public Usuario findById(int id)**
- **public List<Usuario> pesquisarPorNome(String nome)**
- **fechar()**

Esses métodos irão nos ajudar na implementação da edição de um usuário (atualizar) e na busca de usuários por um nome ou parte do nome.

Não se esqueça de fazer o download do driver JDBC do MySQL, criar a pasta lib, colocar o driver nessa pasta e configurar o BUILD PATH do projeto.

Por enquanto o código da “GerenciarUsuario” deverá estar assim:

```
package usuario.gerenciar;

import java.util.List;

import usuario.Usuario;

public class GerenciarUsuario {

    public void salvar(Usuario usuario) {
    }

    public void atualizar(Usuario usuario) {
    }

    public List<Usuario> listar() {
        return null;
    }

    public void remover(int id) {
    }

    public Usuario findById(int id) {
        return null;
    }

    public List<Usuario> pesquisarPorNome(String nome) {
        return null;
    }

    public void fechar() {
    }
}
```

JDBC (Java Database Connectivity)

JDBC (Java Database Connectivity) é uma **API (Application Programming Interface)** do Java que permite que aplicações Java se conectem e interajam com **bancos de dados relacionais** (como MySQL, PostgreSQL, Oracle, SQL Server, etc.).

Para que serve o JDBC?

Serve para:

1. **Conectar uma aplicação Java a um banco de dados.**
2. **Executar comandos SQL** (como SELECT, INSERT, UPDATE, DELETE).
3. **Recuperar e manipular dados** retornados pelo banco.
4. **Gerenciar conexões**, transações e tratamento de erros.

O que é um driver JDBC

Cada banco de dados tem uma maneira específica de se comunicar com as aplicações, a maneira que usamos para enviar comandos SQL para o MySQL é diferente da maneira que enviamos o mesmo comando para um banco Postgres, MSSQL, Oracle, etc. Cada banco tem seu próprio protocolo de comunicação. Então como podemos fazer nossa aplicação se comunicar com o banco de dados?

A resposta é bem simples. Com um driver.

Funciona da mesma maneira com hardware. Quando compramos uma nova placa de vídeo, como podemos usufruir de todos os recursos e fazer com que o sistema operacional reconheça a nova placa? Através do driver. É o driver que traduz os comandos do sistema operacional para a placa. Quando o Windows precisa que a placa desenhe um polígono na tela, ele dá o comando para o driver que traduz o comando para as instruções específicas da placa de vídeo.

Um **driver JDBC (Java Database Connectivity)** é um componente essencial que permite que uma aplicação Java se conecte a um **banco de dados relacional**. Ele atua como uma **ponte entre o Java e o banco de dados**, traduzindo comandos SQL da aplicação Java em comandos que o banco de dados consegue entender e processar.

Cada banco de dados possui um driver específico e deve ser fornecido pela empresa que desenvolve o banco de dados, assim, no site do banco de dados MySQL podemos fazer o download do driver JDBC, além de o site disponibilizar driver para outras linguagens de programação.

Para que serve um driver JDBC?

O driver JDBC serve para:

- **Estabelecer a conexão** com um banco de dados (como MySQL, PostgreSQL, Oracle, etc.)
- **Enviar comandos SQL** (como SELECT, INSERT, UPDATE)
- **Receber os resultados** dessas consultas
- **Tratar erros** ou exceções que possam ocorrer durante a comunicação com o banco

Como usar um driver JDBC

1. Carregamento do driver

O driver JDBC específico do banco de dados é carregado na aplicação, geralmente com:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

2. Estabelecimento da conexão

Através da classe DriverManager, a aplicação abre uma conexão:

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://localhost:3306/meubanco", "usuario", "senha");
```

3. Criação de um Statement

É criado um objeto Statement ou PreparedStatement para enviar comandos SQL:

```
String sql = "select * from tb_carros";  
PreparedStatement pstm = conn.prepareStatement(sql);
```

4. Processamento dos resultados

O resultado da consulta é processado da seguinte maneira:

```
ResultSet rs = pstm.executeQuery();  
while ( rs.next() ) {  
    String fabricante = rs.getString("fabricante");  
    System.out.println("Fabricante: " + fabricante);  
}
```

O método rs.next() é usado para **percorrer os resultados** de uma consulta SQL executada com ResultSet.

O que ele faz?

rs.next() **move o cursor** para a próxima linha do resultado retornado pelo banco de dados.

- Ele **retorna true** se **existe uma próxima linha** disponível.
 - Retorna **false** se **não há mais linhas** para ler.
-

Explicando:

1. pstm.executeQuery() executa a consulta SQL e retorna um ResultSet com os dados.
2. O rs.next() **verifica se há uma próxima linha** no conjunto de resultados.
3. Se houver, move o cursor para essa linha e permite o acesso aos dados com rs.getString("nome"), rs.getInt("idade"), etc.
4. O while continua até que rs.next() retorne false, ou seja, não há mais linhas.

O cursor do ResultSet **inicia antes da primeira linha**. Por isso, você **precisa chamar rs.next() pelo menos uma vez** antes de acessar os dados.

Ele é geralmente usado em loops para processar todos os registros de uma consulta.

5. Fechamento dos objetos

Após o uso, a conexão e os objetos são fechados:

```
rs.close();
pstm.close();
conn.close();
```

Principais classes

Elemento	Função
Driver JDBC	Faz a comunicação entre Java e o banco de dados
DriverManager	Gerencia os drivers e cria conexões
Connection	Representa a conexão com o banco de dados
PreparedStatement	Executa comandos SQL com parâmetros, prevenindo SQL Injection
ResultSet	Armazena os resultados das consultas SQL

Implementação do Gerenciar

Vamos criar algumas variáveis de classe, uma para a conexão com o banco de dados, uma para executar as consultas e uma para as strings de consultas:

```
private Connection conn;
private PreparedStatement pstm;
private String sql;
```

Seguindo a sequência definida em “Como usar um driver JDBC”, primeiro devemos carregar o driver do banco na memória, faremos isso no construtor:

```
public GerenciarUsuario() {  
    try{  
        Class.forName("com.mysql.cj.jdbc.Driver");  
    } catch (ClassNotFoundException e){  
        e.printStackTrace();  
    }  
  
    try{  
        conn      = DriverManager.getConnection("jdbc:mysql://localhost:3306/db_ifpr",  
        "root", "root");  
    } catch (SQLException e){  
        e.printStackTrace();  
    }  
}
```

Como o carregamento do driver pode dar um erro, devemos colocar o bloco dentro de um try...catch.

Em seguida criamos a conexão com o banco de dados no atributo “conn”.

PreparedStatement

O PreparedStatement é uma interface da linguagem Java (do pacote `java.sql`) usada para executar comandos SQL (como `SELECT`, `INSERT`, `UPDATE`, `DELETE`) de forma **segura**, **eficiente** e **parametrizada** em um banco de dados. Ele é uma evolução do Statement tradicional, com vantagens importantes.

Como funciona o PreparedStatement

1. Criação do comando SQL com parâmetros

Você escreve o SQL com **interrogações (?)** no lugar dos valores que serão inseridos depois.

2. Preparação do comando

O banco de dados analisa e compila o comando **uma única vez**, mesmo que seja executado várias vezes com parâmetros diferentes.

3. Definição dos parâmetros

Usa-se métodos como `setInt()`, `setString()`, etc., para passar valores para as interrogações.

4. Execução

O comando é executado com os parâmetros definidos.

Vantagens

- **Evita SQL Injection:** Os valores são tratados como dados e não como parte do SQL.
- **Melhora de desempenho:** O SQL é pré-compilado e pode ser reutilizado.
- **Código mais organizado:** Separação clara entre comando e dados.

Métodos comuns de PreparedStatement

Método	Descrição
<code>setString(int, String)</code>	Define valor <code>String</code> para o parâmetro
<code>setInt(int, int)</code>	Define valor <code>int</code>
<code>setDouble(int, double)</code>	Define valor <code>double</code>
<code>executeQuery()</code>	Usado para <code>SELECT</code>
<code>executeUpdate()</code>	Usado para <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code>

Salvar

O método salvar recebe como parâmetro um objeto da classe usuário. Para salvar o objeto no banco de dados, vamos usar o comando insert:

“insert into tb_usuario (nome, cpf, data_nascimento) values (?, ?, ?)”.

As interrogações serão substituídas pelos valores reais que o usuário irá digitar na tela.

```
public void salvar(Usuario usuario) {
    try{
        sql = "insert into tb_usuario (nome, cpf, data_nascimento) values (?, ?, ?)";
        pstm = conn.prepareStatement(sql);

        pstm.setString(1, usuario.getNome());
        pstm.setString(2, usuario.getCpf());
        pstm.setDate(3, Date.valueOf(usuario.getDataNascimento()));

        pstm.executeUpdate();
    } catch (SQLException e){
        e.printStackTrace();
    }
}
```

Atualizar

O método atualizar deve fazer um update no registro do usuário existente (id) com o seguinte comando SQL:

“update into tb_usuario set nome = ?, cpf = ?, data_nascimento = ? where id = ?”.

Como não saberemos quais atributos foram alterados, iremos atualizar todos eles no banco de dados. O método atualizar ficará assim:

```
public void atualizar(Usuario usuario) {  
    try {  
        sql = "update into tb_usuario set nome = ?, cpf = ?, data_nascimento = ?  
where id = ?";  
        pstm = conn.prepareStatement(sql);  
  
        pstm.setString(1, usuario.getNome());  
        pstm.setString(2, usuario.getCpf());  
        pstm.setDate(3, Date.valueOf(usuario.getDataNascimento()));  
        pstm.setInt(4, usuario.getId());  
  
        pstm.executeUpdate();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Listar

O método listar deve retornar uma lista com todos os usuários do banco de dados. Listar todos os usuários pode ser um problema, principalmente se a tabela possuir muitos registros. O ideal é que sejam listados os últimos 15 ou 20 registros, caso o usuário precise de um registro específico ou usando algum critério, devemos criar um menu ou fornecer maneiras para que o usuário faça um filtro sobre os resultados.

Vamos listar somente os últimos 20 registros, para isso usaremos o seguinte comando SQL:

“select * from tb_usuario order by id desc limit 20”.

O comando acima irá buscar 20 registros na tabela usuário e ordenar pelo id de forma decrescente.

Cada resultado deve ser convertido em um objeto da classe Usuario e em seguida devemos adicionar o objeto dentro de uma lista que deve ser retornada pelo método listar.

O método ficará da seguinte maneira:

```
public List<Usuario> listar() {
    sql = "select * from tb_usuario order by id desc limit 20";
    return executarSelect();
}

private List<Usuario> executarSelect() {
    List<Usuario> usuarios = new ArrayList<>();
    try{
        pstm = conn.prepareStatement(sql);

        ResultSet rs = pstm.executeQuery();
        while ( rs.next() )
        {
            Usuario usuario = resultToObject(rs);

            usuarios.add(usuario);
        }
    } catch (SQLException e){
        e.printStackTrace();
    }
    return usuarios;
}

private Usuario resultToObject(ResultSet rs) throws SQLException {
    int id = rs.getInt("id");
    String nome = rs.getString("nome");
    String cpf = rs.getString("cpf");
    Date dataNasc = rs.getDate("data_nascimento");

    Usuario usuario = new Usuario();
    usuario.setId(id);
    usuario.setNome(nome);
    usuario.setCpf(cpf);

    usuario.setDataNascimento(dataNasc.toInstant().atZone(ZonedDateTime.systemDefault())
.toLocalDate());
    return usuario;
}
```

Remover

O método remover deve remover o registro escolhido com o seguinte comando SQL:

“delete from tb_usuario where id = ?”.

O método remover ficará da seguinte maneira:

```
public void remover(int id) {  
    try {  
        sql = "delete from tb_usuario where id = ?";  
        pstm = conn.prepareStatement(sql);  
  
        pstm.setInt(1, id);  
  
        pstm.executeUpdate();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

FindByld

O método findByld deve buscar um usuário pelo id especificado com o seguinte comando SQL:

“select * from tb_usuario where id = ?”.

```
public Usuario findByld(int id) {  
    Usuario usuario = null;  
    try {  
        sql = "select * from tb_usuario where id = ?";  
        pstm = conn.prepareStatement(sql);  
  
        pstm.setInt(1, id);  
  
        ResultSet rs = pstm.executeQuery();  
        rs.next();  
        usuario = resultToObject(rs);  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    return usuario;  
}
```

PesquisarPorNome

O método pesquisarPorNome irá fazer a busca de usuários por parte do nome que foi digitado pelo usuário na tela com o seguinte comando SQL:

```
"select * from tb_usuario where nome like '?%' order by id limit 20".
```

O método simplesmente configura o atributo “sql” e chama o método que executa o select que já deixamos pronto quando implementamos o Listar, o código ficará dessa maneira:

```
public List<Usuario> pesquisarPorNome(String nome) {  
    sql = "select * from tb_usuario where nome like '?%' order by id limit 20";  
    return executarSelect();  
}
```

Se precisarmos fazer outro filtro, como por exemplo por CPF, basta mudar o SQL e chamar o método executarSelect, dessa maneira:

```
public List<Usuario> pesquisarPorCpf(String cpf) {  
    sql = "select * from tb_usuario where cpf = ? order by id limit 20";  
    return executarSelect();  
}
```

Dessa maneira ficou bem fácil de fazer novas consultas.

Fechar

Por fim, o método fechar, que irá fechar o “pst” e a conexão com o banco de dados.

```
public void fechar() {  
    try{  
        pst.close();  
        conn.close();  
    } catch (SQLException e){  
        e.printStackTrace();  
    }  
}
```

Código completo da classe GerenciarUsuario:

```
package usuario.gerenciar;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.List;

import usuario.Usuario;

public class GerenciarUsuario {

    private Connection conn;
    private PreparedStatement pstm;
    private String sql;

    public GerenciarUsuario() {
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e){
            e.printStackTrace();
        }
    }

    try{
        conn
    DriverManager.getConnection("jdbc:mysql://localhost:3306/db_ifpr", "root", "root");
    } catch (SQLException e){
        e.printStackTrace();
    }
}

public void salvar(Usuario usuario) {
    try{
        sql = "insert into tb_usuario (nome, cpf, data_nascimento) values (?, ?, ?)";
        pstm = conn.prepareStatement(sql);

        pstm.setString(1, usuario.getNome());
        pstm.setString(2, usuario.getCpf());
        pstm.setDate(3, Date.valueOf(usuario.getDataNascimento()));
    }
}
```

```

        pstm.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void atualizar(Usuario usuario) {
    try{
        sql = "update into tb_usuario set nome = ?, cpf = ?, data_nascimento
= ? where id = ?";
        pstm = conn.prepareStatement(sql);

        pstm.setString(1, usuario.getNome());
        pstm.setString(2, usuario.getCpf());
        pstm.setDate(3, Date.valueOf(usuario.getDataNascimento()));
        pstm.setInt(4, usuario.getId());

        pstm.executeUpdate();
    } catch (SQLException e){
        e.printStackTrace();
    }
}

public List<Usuario> listar() {
    sql = "select * from tb_usuario order by id desc limit 20";
    return executarSelect();
}

private List<Usuario> executarSelect() {
    List<Usuario> usuarios = new ArrayList<>();
    try{
        pstm = conn.prepareStatement(sql);

        ResultSet rs = pstm.executeQuery();
        while ( rs.next() )
        {
            Usuario usuario = resultToObject(rs);

            usuarios.add(usuario);
        }
    } catch (SQLException e){
        e.printStackTrace();
    }
    return usuarios;
}

private Usuario resultToObject(ResultSet rs) throws SQLException {
    int id = rs.getInt("id");
}

```

```

        String nome = rs.getString("nome");
        String cpf = rs.getString("cpf");
        Date dataNasc = rs.getDate("data_nascimento");

        Usuario usuario = new Usuario();
        usuario.setId(id);
        usuario.setNome(nome);
        usuario.setCpf(cpf);
        usuario.setDataNascimento(dataNasc.toLocalDate());
        return usuario;
    }

public void remover(int id) {
    try{
        sql = "delete from tb_usuario where id = ?";
        pstm = conn.prepareStatement(sql);

        pstm.setInt(1, id);

        pstm.executeUpdate();
    } catch (SQLException e){
        e.printStackTrace();
    }
}

public Usuario findById(int id) {
    Usuario usuario = null;
    try{
        sql = "select * from tb_usuario where id = ?";
        pstm = conn.prepareStatement(sql);

        pstm.setInt(1, id);

        ResultSet rs = pstm.executeQuery();
        rs.next();
        usuario = resultToObject(rs);
    } catch (SQLException e){
        e.printStackTrace();
    }
    return usuario;
}

public List<Usuario> pesquisarPorNome(String nome) {
    sql = "select * from tb_usuario where nome like '?%' order by id limit 20";
    return executarSelect();
}

public void fechar() {
}

```

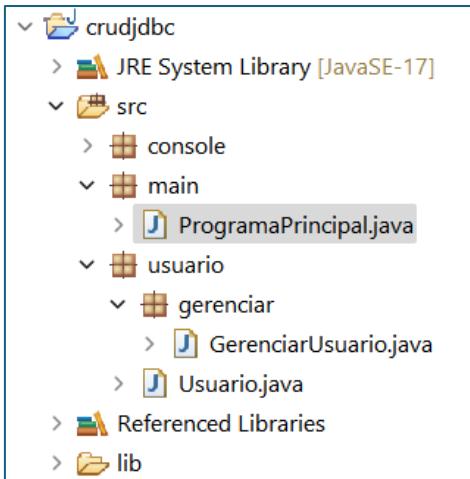
```
    try{
        pstm.close();
        conn.close();
    } catch (SQLException e){
        e.printStackTrace();
    }
}
```

ProgramaPrincipal

Vamos iniciar a implementação do nosso programa, vamos juntar todas as peças e fazê-las funcionar.

Como nosso programa vai funcionar em modo texto (cmd), vamos usar a classe auxiliar “Console” para fazer a leitura do teclado. O código está no fim desse documento.

Crie um pacote chamado “main” e dentro crie a classe “ProgramaPrincipal” com o método main:



O programa deverá mostrar o seguinte menu:

```
--- SUPER CRUD ---
1 – Cadastrar usuário
2 – Alterar usuário
3 – Listar usuários
4 – Remover Usuário
9 – Sair
```

O programa deve permanecer em loop até que o usuário escolha a opção 9. Para isso usaremos um loop **do...while**, pois o menu deve ser mostrado pelo menos uma vez durante a execução do programa.

Vamos iniciar declarando uma variável para a opção escolhida pelo usuário, uma variável para a classe que manipula a entidade no banco de dados (GerenciarUsuario) e uma para ler do teclado usando a classe Console:

```
package main;

import console.Console;

public class ProgramaPrincipal {

    public static void main(String[] args) {
        GerenciarUsuario gerenciarUsuario = new GerenciarUsuario();
```

```
Console console = new Console();
int opcao = 0;

do {

    System.out.println("");
    System.out.println("--- SUPER CRUD ---");
    System.out.println("");
    System.out.println("1 - Cadastrar usuário");
    System.out.println("2 - Alterar usuário");
    System.out.println("3 - Listar usuários");
    System.out.println("4 - Remover usuário");
    System.out.println("9 - Sair");

    opcao = console.readInt("Digite uma opção: ");

    if (opcao == 1)  {

    } else if (opcao == 2) {

    } else if (opcao == 3) {

    } else if (opcao == 4) {

    }

} while (opcao != 9);

System.out.println("Terminando o programa, bye");
}

}
```

Cadastrar

A lógica do cadastro deve ser a seguinte:

- 1 – Criar um novo objeto da classe Usuario.
- 2 - Ler os dados do usuário (nome, cpf e data de nascimento), lembre-se que o ID deve ser gerado pelo banco de dados.
- 3 – Definir os atributos do Usuario com os dados lidos do teclado.
- 4 – Chamar o gerenciarUsuario para salvar o novo objeto no banco de dados.
- 5 – Mostrar mensagem de sucesso para o usuário.

O código ficará dessa maneira:

```
if (opcao == 1) {  
    System.out.println("\nCadastrar novo usuário\n");  
  
    Usuario usuario = new Usuario();  
  
    String nome = console.readLine("Digite o nome: ");  
    long cpf = console.readLong("Digite o CPF: ");  
    LocalDate dataNascimento = console.readLocalDate("Digite a data de nascimento: ");  
  
    usuario.setNome(nome);  
    usuario.setCpf(cpf);  
    usuario.setDataNascimento(dataNascimento);  
  
    gerenciarUsuario.salvar(usuario);  
  
    System.out.println("\nUsuário criado com sucesso\n");  
}
```

Alterar

A lógica da alteração deve ser a seguinte:

- 1 – Perguntar para o usuário qual ID ele quer editar (caso o usuário não saiba, ele deve listar os usuários e descobrir o ID, ou mais para o fim desse documento iremos implementar uma pesquisa pelo nome do usuário).
- 2 – Buscar o objeto referente ao ID escolhido no banco de dados para fazermos as alterações.
- 3 - Ler novamente os dados do usuário (nome, cpf e data de nascimento).
- 4 – Definir os atributos do Usuario com os novos dados lidos do teclado.
- 5 – Chamar o gerenciarUsuario para atualizar objeto no banco de dados.
- 6 – Mostrar mensagem de sucesso para o usuário.

O código ficará dessa maneira:

```
} else if (opcao == 2) {  
    System.out.println("\nAlterar usuário\n");  
  
    int idParaAlterar = console.readInt("Digite o ID para alterar: ");  
  
    Usuario usuario = gerenciarUsuario.findById(idParaAlterar);  
  
    String nome = console.readLine("Digite o nome: ");  
    long cpf = console.readLong("Digite o CPF: ");  
    LocalDate dataNascimento = console.readLocalDate("Digite a data de nascimento: ");  
  
    usuario.setNome(nome);  
    usuario.setCpf(cpf);  
    usuario.setDataNascimento(dataNascimento);  
  
    gerenciarUsuario.atualizar(usuario);  
  
    System.out.println("\nUsuário alterado com sucesso\n");  
}
```

Listar

A lógica para listar os usuários é a seguinte:

- 1 – Criar uma lista de usuário e já atribuir o retorno da chamada do método listar da classe gerenciarUsuario.
- 2 – Fazer um loop na lista de usuários (pode ser um **for** ou um **foreach**).
- 3 – Mostrar na tela os dados de cada elemento da lista.

O código ficará dessa maneira:

```
} else if (opcao == 3) {  
    System.out.println("\nListar usuários\n");  
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");  
    List<Usuario> usuarios = gerenciarUsuario.listar();  
    for (Usuario usuario : usuarios) {  
        System.out.println("ID: " + usuario.getId());  
        System.out.println("Nome: " + usuario.getNome());  
        System.out.println("CPF: " + usuario.getCpf());  
        System.out.println("Data de nascimento: " +  
            dtf.format(usuario.getDataNascimento()));  
    }  
}
```

Remover

A lógica do remover deve ser a seguinte:

- 1 – Perguntar para o usuário qual ID ele quer remover (caso o usuário não saiba, ele deve listar os usuários e descobrir o ID, ou mais para o fim desse documento iremos implementar uma pesquisa pelo nome do usuário).
- 2 – Chamar o gerenciar e passar como parâmetro do método remover o id escolhido.
- 3 – Mostrar uma mensagem de sucesso na tela.

O código ficará dessa maneira:

```
} else if (opcao == 4) {  
    System.out.println("\nRemover usuário\n");  
  
    int idParaExcluir = console.readInt("Digite o ID para excluir: ");  
  
    gerenciarUsuario.remover(idParaExcluir);  
    System.out.println("\nUsuário excluído com sucesso");  
}
```

Código de “ProgramaPrincipal”

```
package main;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;

import console.Console;
import usuario.Usuario;
import usuario.gerenciar.GerenciarUsuario;

public class ProgramaPrincipal {

    public static void main(String[] args) {
        GerenciarUsuario gerenciarUsuario = new GerenciarUsuario();
        Console console = new Console();
        int opcao = 0;

        do {

            System.out.println("");
            System.out.println("--- SUPER CRUD ---");
            System.out.println("");
            System.out.println("1 - Cadastrar usuário");
            System.out.println("2 - Alterar usuário");
            System.out.println("3 - Listar usuários");
            System.out.println("4 - Remover usuário");
            System.out.println("9 - Sair");

            opcao = console.readInt("Digite uma opção: ");

            if (opcao == 1)  {
                System.out.println("\nCadastrar novo usuário\n");

                Usuario usuario = new Usuario();

                String nome = console.readLine("Digite o nome: ");
                long cpf = console.readLong("Digite o CPF: ");
                LocalDate dataNascimento = console.readLocalDate("Digite a
data de nascimento: ");

                usuario.setNome(nome);
                usuario.setCpf(cpf);
                usuario.setDataNascimento(dataNascimento);

                gerenciarUsuario.salvar(usuario);

                System.out.println("\nUsuário criado com sucesso\n");

            } else if (opcao == 2) {
                System.out.println("\nAlterar usuário\n");
            }
        }
    }
}
```

```

        int idParaAlterar = console.readInt("Digite o ID para alterar: ");

        Usuario usuario = gerenciarUsuario.findById(idParaAlterar);

        String nome = console.readLine("Digite o nome: ");
        long cpf = console.readLong("Digite o CPF: ");
        LocalDate dataNascimento = console.readLocalDate("Digite a
data de nascimento: ");

        usuario.setNome(nome);
        usuario.setCpf(cpf);
        usuario.setDataNascimento(dataNascimento);

        gerenciarUsuario.atualizar(usuario);

        System.out.println("\nUsuário alterado com sucesso\n");
    } else if (opcao == 3) {
        System.out.println("\nListar usuários\n");
        DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("dd/MM/yyyy");
        List<Usuario> usuarios = gerenciarUsuario.listar();
        for (Usuario usuario : usuarios) {
            System.out.println("\nID: " + usuario.getId());
            System.out.println("Nome: " + usuario.getNome());
            System.out.println("CPF: " + usuario.getCpf());
            System.out.println("Data de nascimento: " +
dtf.format(usuario.getDataNascimento()));
        }
    } else if (opcao == 4) {
        System.out.println("\nRemover usuário\n");

        int idParaExcluir = console.readInt("Digite o ID para excluir: ");

        gerenciarUsuario.remover(idParaExcluir);
        System.out.println("\nUsuário excluído com sucesso");
    }

} while (opcao != 9);

gerenciarUsuario.fechar();
System.out.println("Terminando o programa, bye");
}
}

```

Ajustes finais

Temos um CRUD completo implementado em “ProgramaPrincipal”, porém, não está orientado a objetos. Fizemos tudo dentro da main e temos alguns problemas, como números mágicos e código duplicado.

Primeiro vamos colocar a console e a variável gerenciarUsuario como atributos da classe:

```
private GerenciarUsuario gerenciarUsuario;  
private Console console;
```

Apenas declare os atributos, a inicialização dos atributos é sempre feita no método construtor, então, crie o método construtor e inicialize os dois atributos:

```
public ProgramaPrincipal() {  
    gerenciarUsuario = new GerenciarUsuario();  
    console = new Console();  
}
```

Mude o nome do método “**main**” para que fique dessa maneira:

```
private void executar() {  
    int opcao = 0;  
    do {  
  
        System.out.println("");  
        System.out.println("--- SUPER CRUD ---");  
        System.out.println("");  
        // restante do código
```

Crie novamente um método “**main**” da seguinte maneira:

```
public static void main(String[] args) {  
    new ProgramaPrincipal().executar();  
}
```

Por enquanto temos o código dessa maneira:

```
package main;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;

import console.Console;
import usuario.Usuario;
import usuario.gerenciar.GerenciarUsuario;

public class ProgramaPrincipal {

    private GerenciarUsuario gerenciarUsuario;
    private Console console;

    public ProgramaPrincipal() {
        gerenciarUsuario = new GerenciarUsuario();
        console = new Console();
    }

    public static void main(String[] args) {
        new ProgramaPrincipal().executar();
    }

    private void executar() {

        int opcao = 0;

        do {
            System.out.println("");
            System.out.println("--- SUPER CRUD ---");
            System.out.println("");
            System.out.println("1 - Cadastrar usuário");
            System.out.println("2 - Alterar usuário");
            System.out.println("3 - Listar usuários");
            System.out.println("4 - Remover usuário");
            System.out.println("9 - Sair");

            opcao = console.readInt("Digite uma opção: ");

            if (opcao == 1) {
                System.out.println("\nCadastrar novo usuário\n");

                Usuario usuario = new Usuario();

                String nome = console.readLine("Digite o nome: ");
                long cpf = console.readLong("Digite o CPF: ");
                LocalDate dataNascimento = console.readLocalDate("Digite a
data de nascimento: ");

                usuario.setNome(nome);
                usuario.setCpf(cpf);
            }
        } while (opcao != 9);
    }
}
```

```

        usuario.setDataNascimento(dataNascimento);

        gerenciarUsuario.salvar(usuario);

        System.out.println("\nUsuário criado com sucesso\n");

    } else if (opcao == 2) {
        System.out.println("\nAlterar usuário\n");

        int idParaAlterar = console.readInt("Digite o ID para alterar: ");

        Usuario usuario = gerenciarUsuario.findById(idParaAlterar);

        String nome = console.readLine("Digite o nome: ");
        long cpf = console.readLong("Digite o CPF: ");
        LocalDate dataNascimento = console.readLocalDate("Digite a
data de nascimento: ");

        usuario.setNome(nome);
        usuario.setCpf(cpf);
        usuario.setDataNascimento(dataNascimento);

        gerenciarUsuario.atualizar(usuario);

        System.out.println("\nUsuário alterado com sucesso\n");
    } else if (opcao == 3) {
        System.out.println("\nListar usuários\n");
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        List<Usuario> usuarios = gerenciarUsuario.listar();
        for (Usuario usuario : usuarios) {
            System.out.println("\nID: " + usuario.getId());
            System.out.println("Nome: " + usuario.getNome());
            System.out.println("CPF: " + usuario.getCpf());
            System.out.println("Data de nascimento: " + +
dtf.format(usuario.getDataNascimento()));
        }
    } else if (opcao == 4) {
        System.out.println("\nRemover usuário\n");

        int idParaExcluir = console.readInt("Digite o ID para excluir: ");

        gerenciarUsuario.remover(idParaExcluir);
        System.out.println("\nUsuário excluído com sucesso");
    }

} while (opcao != 9);

gerenciarUsuario.fechar();
System.out.println("Terminando o programa, bye");
}
}

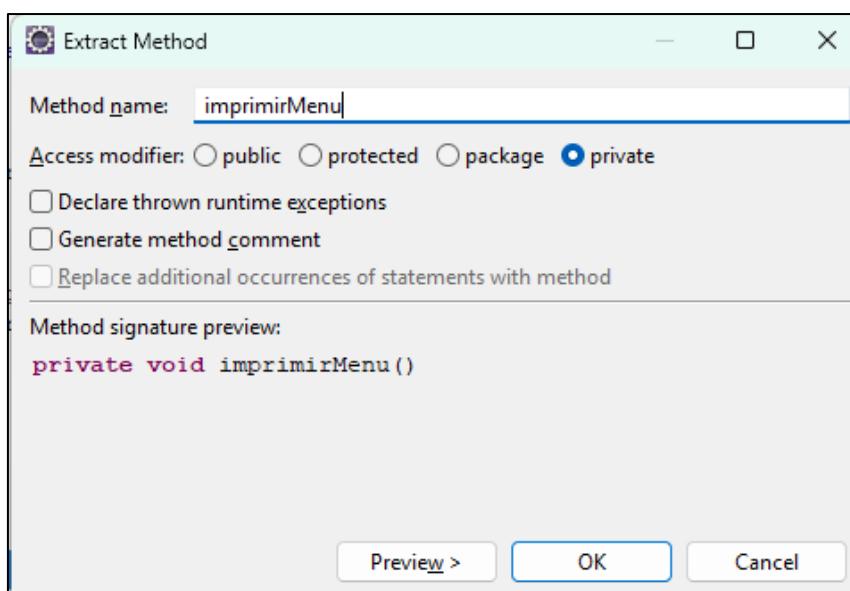
```

Agora vamos refatorar o código em métodos menores com menos responsabilidade (cada método com uma responsabilidade).

Selecione a parte do código que está sendo usada para mostrar o menu na tela:

```
25     private void executar() {  
26  
27         int opcao = 0;  
28  
29         do {  
30             System.out.println("");  
31             System.out.println("---- SUPER CRUD ----");  
32             System.out.println("");  
33             System.out.println("1 - Cadastrar usuário");  
34             System.out.println("2 - Alterar usuário");  
35             System.out.println("3 - Listar usuários");  
36             System.out.println("4 - Remover usuário");  
37             System.out.println("9 - Sair");  
38     }
```

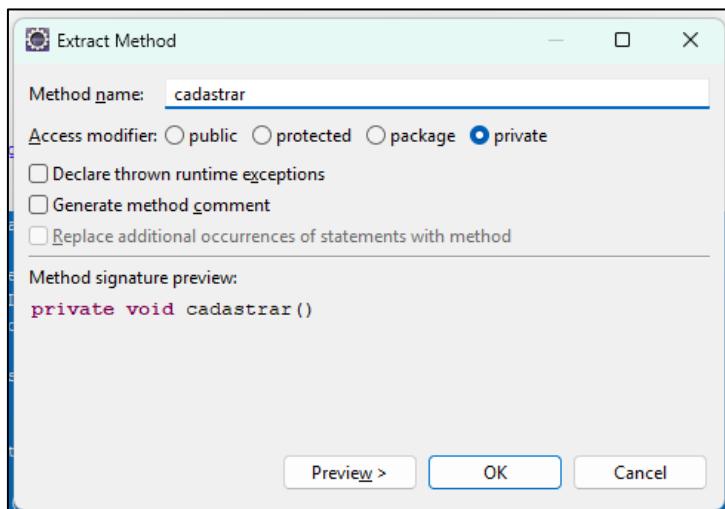
Aperte ALT + SHIFT + M para extrair esse bloco de código para um método com o nome “imprimirMenu”:



O corpo de cada IF será refatorado para métodos referentes às operações que eles realizam, assim, o primeiro IF serve para cadastrar um novo usuário, o segundo para alterar e assim por diante.

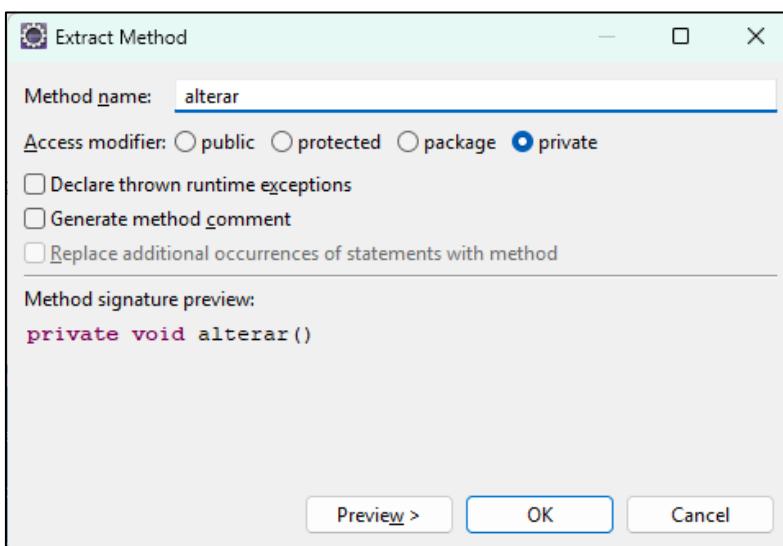
Selecione o código que está dentro do corpo do primeiro IF e aperte ALT + SHIFT + M para extrair o código para um método com o nome “cadastrar”:

```
25@     private void executar() {  
26  
27         int opcao = 0;  
28  
29         do {  
30             imprimirMenu();  
31  
32             opcao = console.readInt("Digite uma opção: ");  
33  
34             if (opcao == 1) {  
35                 System.out.println("\nCadastrar novo usuário\n");  
36  
37                 String nome = console.readLine("Digite o nome: ");  
38                 long cpf = console.readLong("Digite o CPF: ");  
39                 LocalDate dataNascimento = console.readLocalDate("Digite a data de nascimento: ");  
40  
41                 Usuario usuario = new Usuario();  
42                 usuario.setNome(nome);  
43                 usuario.setCpf(cpf);  
44                 usuario.setDataNascimento(dataNascimento);  
45  
46                 gerenciarUsuario.salvar(usuario);  
47  
48                 System.out.println("\nUsuário criado com sucesso\n");  
49  
50             } else if (opcao == 2) {  
51                 System.out.println("\nAlterar usuário\n");  
52         }  
53     }  
54 }
```



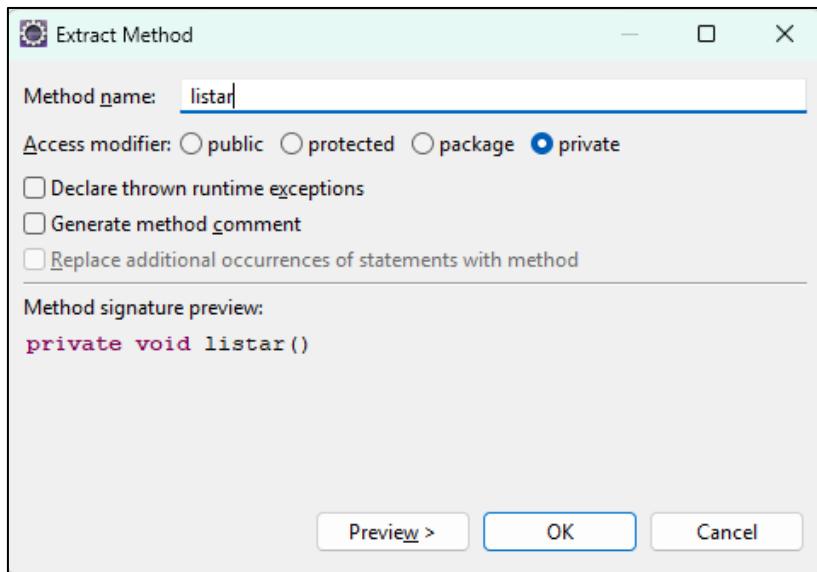
Selecione o código que está dentro do corpo do segundo IF e aperte ALT + SHIFT + M para extrair o código para um método com o nome “alterar”:

```
34         if (opcao == 1) {
35             cadastrar();
36
37         } else if (opcao == 2) {
38             System.out.println("\nAlterar usuário\n");
39
40             int idParaAlterar = console.readInt("Digite o ID para alterar: ");
41
42             Usuario usuario = gerenciarUsuario.findById(idParaAlterar);
43
44             String nome = console.readLine("Digite o nome: ");
45             long cpf = console.readLong("Digite o CPF: ");
46             LocalDate dataNascimento = console.readLocalDate("Digite a data de nascimento: ");
47
48             usuario.setNome(nome);
49             usuario.setCpf(cpf);
50             usuario.setDataNascimento(dataNascimento);
51
52             gerenciarUsuario.atualizar(usuario);
53
54             System.out.println("\nUsuário alterado com sucesso\n");
55         } else if (opcao == 3) {
56             System.out.println("\nListar usuários\n");
57     }
```



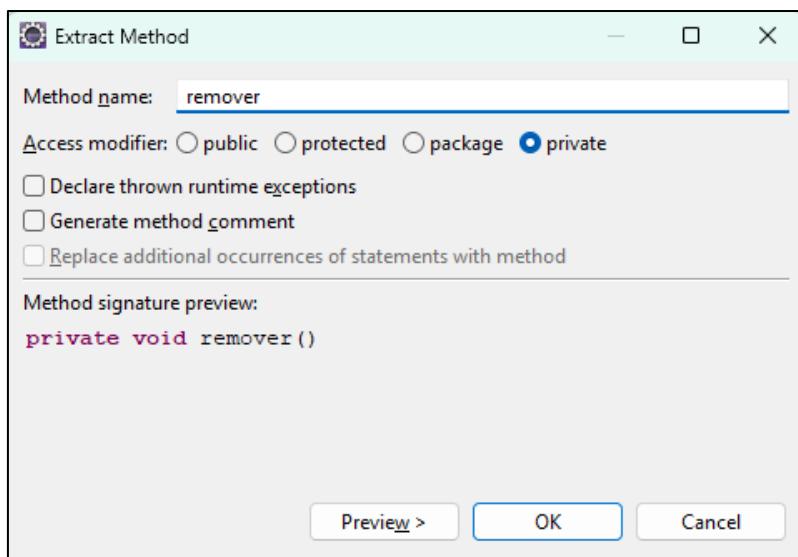
Selecione o código que está dentro do corpo do terceiro IF e aperte ALT + SHIFT + M para extrair o código para um método com o nome “listar”:

```
34         if (opcao == 1) {
35             cadastrar();
36         } else if (opcao == 2) {
37             alterar();
38         } else if (opcao == 3) {
39             System.out.println("\nListar usuários\n");
40             DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");
41             List<Usuario> usuarios = gerenciarUsuario.listar();
42             for (Usuario usuario : usuarios) {
43                 System.out.println("\nID: " + usuario.getId());
44                 System.out.println("Nome: " + usuario.getNome());
45                 System.out.println("CPF: " + usuario.getCpf());
46                 System.out.println("Data de nascimento: " + dtf.format(usuario.getDataNascimento()));
47             }
48         } else if (opcao == 4) {
49             System.out.println("\nRemover usuário\n");
50     }
```



Selecione o código que está dentro do corpo do quarto IF e aperte ALT + SHIFT + M para extrair o código para um método com o nome “remover”:

```
if (opcao == 1) {
    cadastrar();
} else if (opcao == 2) {
    alterar();
} else if (opcao == 3) {
    listar();
} else if (opcao == 4) {
    System.out.println("\nRemover usuário\n");
    int idParaExcluir = console.readInt("Digite o ID para excluir: ");
    gerenciarUsuario.remover(idParaExcluir);
    System.out.println("\nUsuário excluído com sucesso");
}
```



Veja que o método executar já está bem melhor, com menos código e menos responsabilidade:

```
25④  private void executar() {  
26  
27      int opcao = 0;  
28  
29      do {  
30          imprimirMenu();  
31  
32          opcao = console.readInt("Digite uma opção: ");  
33  
34          if (opcao == 1) {  
35              cadastrar();  
36          } else if (opcao == 2) {  
37              alterar();  
38          } else if (opcao == 3) {  
39              listar();  
40          } else if (opcao == 4) {  
41              remover();  
42          }  
43      } while (opcao != 9);  
44  
45      System.out.println("Terminando o programa, bye");  
46  }
```

Ainda dá pra melhorar mais, vamos refatorar esses números mágicos nos IFs e na condição do WHILE, 1, 2, 3, 4 e 9.

Números mágicos

Em programação, **número mágico** (ou *magic number*, em inglês) é um **número literal usado diretamente no código** sem uma explicação clara do que ele representa.

Por que são chamados de "mágicos"?

Porque, ao olhar para o código, o valor simplesmente "aparece" do nada, sem contexto — como se fosse algo mágico. Isso dificulta a compreensão e a manutenção do código.

Exemplo ruim (com número mágico):

```
double salarioFinal = salarioBase * 1.15;
```

Neste caso, o 1.15 é um número mágico. Quem lê o código não sabe de onde ele veio ou por que foi escolhido.

Exemplo bom (sem número mágico):

```
double static final BONUS_ANUAL = 1.15;
salarioFinal = salarioBase * BONUS_ANUAL;
```

Agora, o valor tem um nome explicativo, e fica muito mais fácil entender o que ele representa.

Problemas dos números mágicos:

- **Dificultam a leitura** do código.
- **Complicam a manutenção** (se você precisar mudar o valor, pode esquecer de mudar em todos os lugares).
- **Aumentam a chance de bugs**, especialmente se o número for usado em vários pontos do programa.

Vamos transformar os números mágicos do nosso código em constantes com os nomes significativos, no início da classe crie os seguintes atributos:

```
private static final int CADASTRAR = 1;
private static final int ALTERAR = 2;
private static final int LISTAR = 3;
private static final int REMOVER = 4;
private static final int SAIR = 9;
```

```
11 public class ProgramaPrincipal {
12
13     private GerenciarUsuario gerenciarUsuario;
14     private Console console;
15
16     private static final int CADASTRAR = 1;
17     private static final int ALTERAR = 2;
18     private static final int LISTAR = 3;
19     private static final int REMOVER = 4;
20     private static final int SAIR = 9;
21 }
```

Agora troque os números no código pela respectiva constante:

```
32④    private void executar() {
33
34        int opcao = 0;
35
36        do {
37            imprimirMenu();
38
39            opcao = console.readInt("Digite uma opção: ");
40
41            if (opcao == CADASTRAR) {
42                cadastrar();
43            } else if (opcao == ALTERAR) {
44                alterar();
45            } else if (opcao == LISTAR) {
46                listar();
47            } else if (opcao == REMOVER) {
48                remover();
49            }
50
51        } while (opcao != SAIR);
52
53        System.out.println("Terminando o programa, bye");
54    }
```

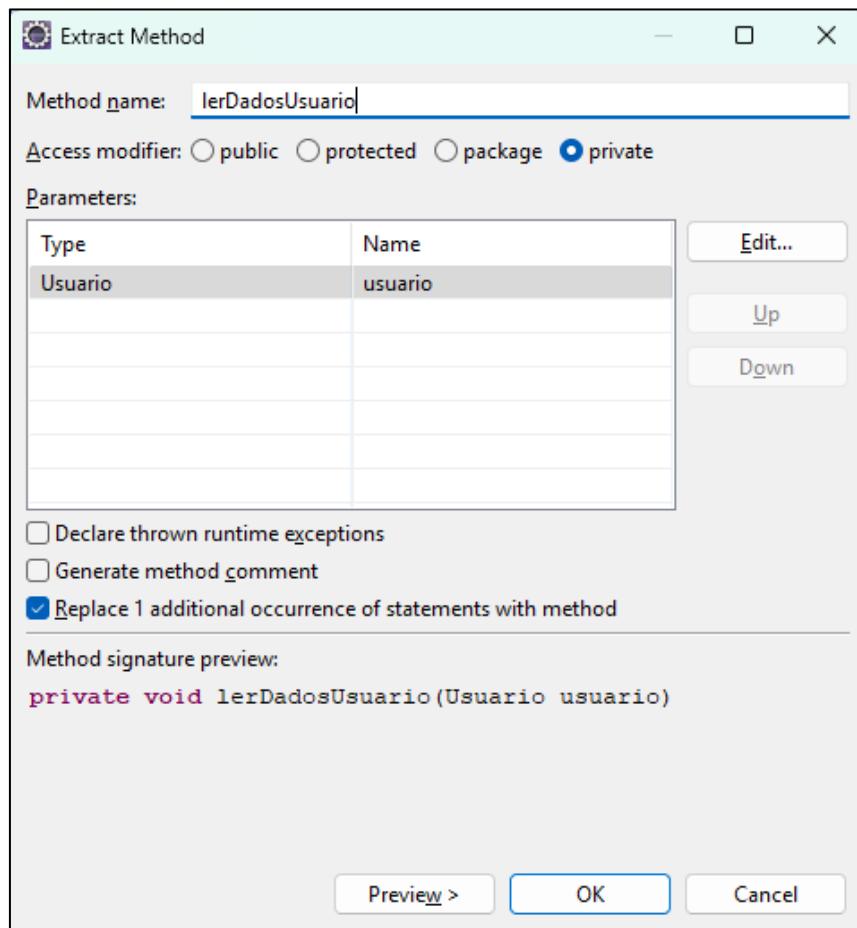
Veja como o código fica mais fácil de entender.

Agora procure o método “cadastrar”, vamos refatorar ele.

```
97④    private void cadastrar() {
98        System.out.println("\nCadastrar novo usuário\n");
99
100        Usuario usuario = new Usuario();
101
102        String nome = console.readLine("Digite o nome: ");
103        long cpf = console.readLong("Digite o CPF: ");
104        LocalDate dataNascimento = console.readLocalDate("Digite a data de nascimento: ");
105
106        usuario.setNome(nome);
107        usuario.setCpf(cpf);
108        usuario.setDataNascimento(dataNascimento);
109
110        gerenciarUsuario.salvar(usuario);
111
112        System.out.println("\nUsuário criado com sucesso\n");
113    }
```

Selecione essa parte do código e aperte ALT + SHIFT + M para extrair esse bloco de código para um método chamado “lerDadosUsuario”:

```
97④    private void cadastrar() {  
98        System.out.println("\nCadastrar novo usuário\n");  
99  
100       Usuario usuario = new Usuario();  
101  
102       String nome = console.readLine("Digite o nome: ");  
103       long cpf = console.readLong("Digite o CPF: ");  
104       LocalDate dataNascimento = console.readLocalDate("Digite a data de nascimento: ");  
105  
106       usuario.setNome(nome);  
107       usuario.setCpf(cpf);  
108       usuario.setDataNascimento(dataNascimento);  
109  
110       gerenciarUsuario.salvar(usuario);  
111  
112       System.out.println("\nUsuário criado com sucesso\n");  
113   }
```



O Eclipse irá procurar todas as ocorrências desse bloco de código e irá substituir o bloco pela chamada de método correspondente, repare que ele refatorou o método “cadastrar” e “alterar”:

```
77④    private void alterar() {
78        System.out.println("\nAlterar usuário\n");
79
80        int idParaAlterar = console.readInt("Digite o ID para alterar: ");
81
82        Usuario usuario = gerenciarUsuario.findById(idParaAlterar);
83
84        lerDadosUsuario(usuario); ←
85
86        gerenciarUsuario.atualizar(usuario);
87
88        System.out.println("\nUsuário alterado com sucesso\n");
89    }
90
91④    private void cadastrar() {
92        System.out.println("\nCadastrar novo usuário\n");
93
94        Usuario usuario = new Usuario();
95
96        lerDadosUsuario(usuario); ←
97
98        gerenciarUsuario.salvar(usuario);
99
100       System.out.println("\nUsuário criado com sucesso\n");
101 }
```

Por fim, vamos refatorar o método “listar”. Primeiro modifique o código para que ele fique igual ao código abaixo (inverta a posição das linhas 68 para a 67):

ANTES:

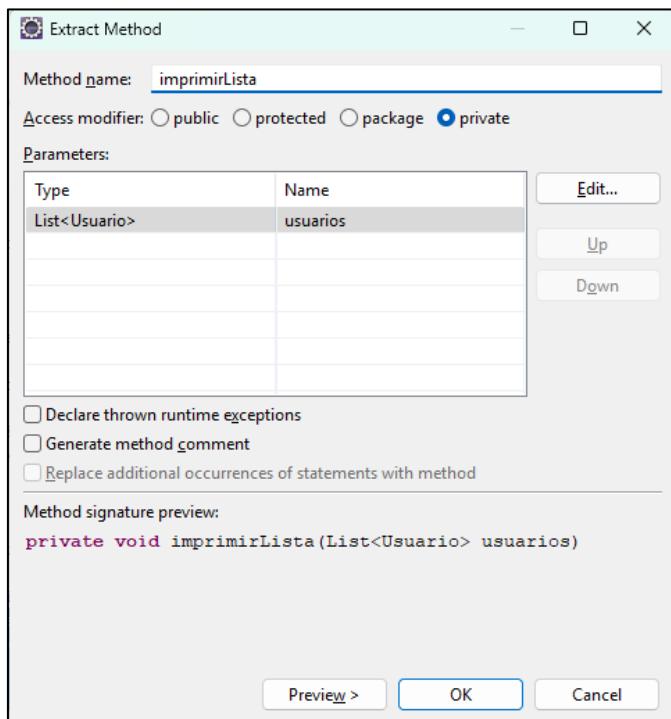
```
65④    private void listar() {
66        System.out.println("\nListar usuários\n");
67        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");
68        List<Usuario> usuarios = gerenciarUsuario.listar();
69        for (Usuario usuario : usuarios) {
70            System.out.println("\nID: " + usuario.getId());
71            System.out.println("Nome: " + usuario.getNome());
72            System.out.println("CPF: " + usuario.getCpf());
73            System.out.println("Data de nascimento: " + dtf.format(usuario.getDataNascimento()));
74        }
75    }
```

DEPOIS:

```
65④    private void listar() {
66        System.out.println("\nListar usuários\n");
67
68        List<Usuario> usuarios = gerenciarUsuario.listar();
69        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");
70        for (Usuario usuario : usuarios) {
71            System.out.println("\nID: " + usuario.getId());
72            System.out.println("Nome: " + usuario.getNome());
73            System.out.println("CPF: " + usuario.getCpf());
74            System.out.println("Data de nascimento: " + dtf.format(usuario.getDataNascimento()));
75        }
76    }
```

Agora, selecione o bloco abaixo e aperte ALT + SHIFT + M para extrair o bloco de código para o método “imprimirLista”:

```
65    private void listar() {
66        System.out.println("\nListar usuários\n");
67
68        List<Usuario> usuarios = gerenciarUsuario.listar();
69        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");
70        for (Usuario usuario : usuarios) {
71            System.out.println("\nID: " + usuario.getId());
72            System.out.println("Nome: " + usuario.getNome());
73            System.out.println("CPF: " + usuario.getCpf());
74            System.out.println("Data de nascimento: " + dtf.format(usuario.getDataNascimento()));
75        }
76    }
```



Pronto, terminamos a refatoração do código, temos o programa usando orientação a objetos, métodos curtos, cada um com sua responsabilidade e o código bem legível.

Código final completo da classe “ProgramaPrincipal”

```
package main;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;

import console.Console;
import usuario.Usuario;
import usuario.gerenciar.GerenciarUsuario;

public class ProgramaPrincipal {

    private GerenciarUsuario gerenciarUsuario;
    private Console console;

    private static final int CADASTRAR = 1;
    private static final int ALTERAR = 2;
    private static final int LISTAR = 3;
    private static final int REMOVER = 4;
    private static final int SAIR = 9;

    public ProgramaPrincipal() {
        gerenciarUsuario = new GerenciarUsuario();
        console = new Console();
    }

    public static void main(String[] args) {
        new ProgramaPrincipal().executar();
    }

    private void executar() {

        int opcao = 0;

        do {
            imprimirMenu();

            opcao = console.readInt("Digite uma opção: ");

            if (opcao == CADASTRAR) {
                cadastrar();
            } else if (opcao == ALTERAR) {
                alterar();
            } else if (opcao == LISTAR) {
                listar();
            } else if (opcao == REMOVER) {
                remover();
            }
        } while (opcao != SAIR);

        gerenciarUsuario.fechar();
        System.out.println("Terminando o programa, bye");
    }

    private void remover() {
        System.out.println("\nRemover usuário\n");

        int idParaExcluir = console.readInt("Digite o ID para excluir: ");
    }
}
```

```

        gerenciarUsuario.remover(idParaExcluir);
        System.out.println("\nUsuário excluído com sucesso");
    }

private void listar() {
    System.out.println("\nListar usuários\n");

    List<Usuario> usuarios = gerenciarUsuario.listar();
    imprimirLista(usuarios);
}

private void imprimirLista(List<Usuario> usuarios) {
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy");
    for (Usuario usuario : usuarios) {
        System.out.println("\nID: " + usuario.getId());
        System.out.println("Nome: " + usuario.getNome());
        System.out.println("CPF: " + usuario.getCpf());
        System.out.println("Data de nascimento: " + +
dtf.format(usuario.getDataNascimento()));
    }
}

private void alterar() {
    System.out.println("\nAlterar usuário\n");

    int idParaAlterar = console.readInt("Digite o ID para alterar: ");

    Usuario usuario = gerenciarUsuario.findById(idParaAlterar);

    lerDadosUsuario(usuario);

    gerenciarUsuario.atualizar(usuario);

    System.out.println("\nUsuário alterado com sucesso\n");
}

private void cadastrar() {
    System.out.println("\nCadastrar novo usuário\n");

    Usuario usuario = new Usuario();

    lerDadosUsuario(usuario);

    gerenciarUsuario.salvar(usuario);

    System.out.println("\nUsuário criado com sucesso\n");
}

private void lerDadosUsuario(Usuario usuario) {
    String nome = console.readLine("Digite o nome: ");
    String cpf = console.readString("Digite o CPF: ");
    LocalDate dataNascimento = console.readLocalDate("Digite a data de
nascimento: ");

    usuario.setNome(nome);
    usuario.setCpf(cpf);
    usuario.setDataNascimento(dataNascimento);
}

private void imprimirMenu() {
    System.out.println("");
    System.out.println("--- SUPER CRUD ---");
    System.out.println("");
}

```

```
        System.out.println("1 - Cadastrar usuário");
        System.out.println("2 - Alterar usuário");
        System.out.println("3 - Listar usuários");
        System.out.println("4 - Remover usuário");
        System.out.println("9 - Sair");
    }
}
```

GitHub

Caso você queira o código fonte desse documento, visite o GitHub.

<https://github.com/pauloryj/crudjdbc>

Você pode clonar o repositório ou fazer o download do código em um arquivo zip.