



## MISSÃO PRÁTICA – NÍVEL 3

**Aluno:** Daniel Kilzer Brasil Dias | **Matrícula:** 202310422026

**Campus:** Polo Bezerra de Menezes – Fortaleza/CE

**Disciplina:** BackEnd Sem Banco Não Tem – **Turma:** 9001 – **Semestre:** 2024.4

### Objetivo da Prática

O objetivo principal desta prática foi desenvolver um sistema de cadastro de pessoas (físicas e jurídicas) em Java, utilizando um banco de dados relacional (SQL Server) como repositório de dados.

O sistema foi projetado para permitir as seguintes funcionalidades básicas:

- **Cadastro:** incluir novas pessoas físicas e jurídicas no sistema.
- **Consulta:** buscar informações sobre pessoas específicas por meio de seu ID.
- **Alteração:** modificar os dados de pessoas já cadastradas.
- **Exclusão:** remover registros de pessoas do sistema.
- **Listagem:** exibir todas as pessoas cadastradas ou filtrar por tipo (física ou jurídica).

As seguintes tecnologias foram utilizadas:

- **Java:** linguagem de programação orientada a objetos.
- **JDBC:** API para conectar as aplicações Java a bancos de dados relacionais.
- **SQL Server:** sistema gerenciador de banco de dados relacional.
- **Orientação a objetos:** para modelar as entidades (Pessoa, PessoaFisica, PessoaJuridica) e suas relações.



- **Padrão DAO:** para encapsular a lógica de acesso ao banco de dados.

## 1º Procedimento | Mapeamento Objeto-Relacional e DAO

### CÓDIGOS

#### Pacote `cadastro.util`

O pacote `cadastro.model.util` armazenará as classes `ConectorBD` e `SequenceManager`. Tais classes permitirão a conexão com o banco de dados.

A classe `ConectorBD` permite: conectar-se ao banco de dados, por meio do método `getConnection()`; realizar uma consulta SQL, por meio do método `getPrepared()`; e armazenar o dado tabular da consulta realizar numa estrutura de dados de objeto, por meio do método `getSelect()`. Além disso, as sobrecargas do método `close()` permitem fechar os recursos abertos para se comunicar com o banco de dados.

As constantes `url`, `usuario` e `senha` contêm as informações necessárias para acessar o banco de dados.

#### Classe `ConectorBD`

```
package cadastrobd.model.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class ConectorBD {
    private static final String url =
"jdbc:sqlserver://localhost:1433;databaseName=Loja;encrypt=
true;trustServerCertificate=true";
    private static final String usuario = "loja";
    private static final String senha = "loja";
```



# Estácio

```
public static Connection getConnection() throws
SQLException {
    return DriverManager.getConnection(url, usuario,
senha);
}

public static PreparedStatement getPrepared(Connection
conexao, String sql) throws SQLException {
    return conexao.prepareStatement(sql);
}

public static ResultSet getSelect(PreparedStatement
declaracao) throws SQLException {
    return declaracao.executeQuery();
}

public static void close(Connection conexao) {
    try {
        if (conexao != null) {
            conexao.close();
        }
    }
    catch (SQLException e) {
        System.err.println("Erro ao fechar a
conexão: " + e.getMessage());
    }
}

public static void close(PreparedStatement declaracao)
{
    try {
        if (declaracao != null) {
            declaracao.close();
        }
    }
    catch (SQLException e) {
        System.err.println("Erro ao fechar o
PreparedStatement: " + e.getMessage());
    }
}

public static void close(ResultSet conjuntoResultado)
{
    try {
        if (conjuntoResultado != null) {
            conjuntoResultado.close();
        }
    }
    catch (SQLException e) {
```



# Estácio

```
        System.err.println("Erro ao fechar o  
ResultSet: " + e.getMessage());  
    }  
}  
  
    public static void close(Connection conexao,  
PreparedStatement declaracao) throws SQLException {  
        try {  
            conexao.close();  
            declaracao.close();  
        }  
        catch (SQLException e) {  
            System.err.println("Erro ao tentar encerrar  
recursos: " + e.getMessage());  
        }  
    }  
  
    public static void close(Connection conexao,  
PreparedStatement declaracao, ResultSet conjuntoResultado)  
throws SQLException {  
        try {  
            conexao.close();  
            declaracao.close();  
            conjuntoResultado.close();  
        }  
        catch (SQLException e) {  
            System.err.println("Erro ao tentar encerrar  
recursos: " + e.getMessage());  
        }  
    }  
}
```

A classe `SequenceManager` lidará com a sequência criada no banco de dados para gerar os identificadores dos registros inseridos, obtendo o próximo valor a ser atribuído a um novo registro por meio de seu método `getValue()`.

## Classe `SequenceManager`

```
package cadastrobd.model.util;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
public class SequenceManager {
```



```
public static int getValue(String nomeSequencia) throws SQLException {
    String sql = "SELECT NEXT VALUE FOR " + nomeSequencia + "
AS proximoValor";

    Connection conexao = null;
    PreparedStatement declaracao = null;
    ResultSet conjuntoResultado = null;

    try {
        conexao = ConectorBD.getConnection();
        declaracao = ConectorBD.getPrepared(conexao, sql);
        conjuntoResultado = declaracao.executeQuery();

        if (conjuntoResultado.next()) {
            return conjuntoResultado.getInt("proximoValor");
        }
        else {
            throw new SQLException("Erro ao obter o próximo valor da
sequência: " + nomeSequencia);
        }
    }
    finally {
        ConectorBD.close(conexao, declaracao,
conjuntoResultado);
    }
}
```

### **Pacote cadastrobd.model**

No pacote cadastrobd.model estarão as classes Pessoa, PessoaFisica, PessoaJuridica, PessoaFisicaDAO e PessoaJuridicaDAO.

A classe Pessoa (superclasse) e as classes PessoaFisica e PessoaJuridica (subclasses) instanciarão objetos a serem armazenados em memória quando da execução da aplicação. As classes PessoaFisicaDAO e PessoaJuridicaDAO realizarão a comunicação com o banco de dados, ora recuperando dados, ora inserindo dados e ora alterando dados.

### **Classe Pessoa**

```
package cadastrobd.model;

public class Pessoa {
```



```
private int id;
private String nome;
private String logradouro;
private String cidade;
private String estado;
private String telefone;
private String email;

public Pessoa() {}

public Pessoa(int id, String nome, String logradouro,
String cidade, String estado, String telefone, String
email) {
    this.id = id;
    this.nome = nome;
    this.logradouro = logradouro;
    this.cidade = cidade;
    this.estado = estado;
    this.telefone = telefone;
    this.email = email;
}

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 getLogradouro() {
    return logradouro;
}

public void setLogradouro(String logradouro) {
    this.logradouro = logradouro;
}

public String getCidade() {
```



```
        return cidade;
    }

    public void setCidade(String cidade) {
        this.cidade = cidade;
    }

    public String getEstado() {
        return estado;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void exibir() {
        System.out.println("-----");
        System.out.println("ID: " + getId());
        System.out.println("Nome: " + getNome());
        System.out.println("Logradouro: " + getLogradouro() + ", " + getCidade() + "/" + getEstado());
        System.out.println("Telefone: " + getTelefone());
        System.out.println("E-mail: " + getEmail());
    }
}
```

### **Classe PessoaFisica**

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;
```



```
public PessoaFisica() {}

    public PessoaFisica(int id, String nome, String
logradouro, String cidade, String estado, String telefone,
String email, String cpf) {
        super(id, nome, logradouro, cidade, estado,
telefone, email);
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

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

    @Override
    public void exhibir() {
        super.exibir();
        System.out.println("CPF: " + getCpf());
    }
}
```

### **Classe PessoaJuridica**

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {}

    public PessoaJuridica(int id, String nome, String
logradouro, String cidade, String estado, String telefone,
String email, String cnpj) {
        super(id, nome, logradouro, cidade, estado,
telefone, email);
        this.cnpj = cnpj;
    }

    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```





```
@Override
public void exhibir() {
    super.exibir();
    System.out.println("CNPJ: " + getCnpj());
}
}
```

### Classe PessoaFisicaDAO

```
package cadastrobd.model;

import cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
    public PessoaFisica getPessoa(int id) throws
    SQLException {
        String sql = "SELECT p.*, pf.cpf FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa
WHERE idPessoa = ?";

        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet conjuntoResultado = null;

        try {
            conexao = ConectorBD.getConnection();
            declaracao = ConectorBD.getPrepared(conexao,
sql);
            declaracao.setInt(1, id);
            conjuntoResultado =
ConectorBD.getSelect(declaracao);

            if (conjuntoResultado.next()) {
                return new PessoaFisica(

conjuntoResultado.getInt("idPessoa"),
conjuntoResultado.getString("nome"),
conjuntoResultado.getString("logradouro"),
```



# Estácio

```
        conjuntoResultado.getString("cidade"),
        conjuntoResultado.getString("estado"),
        conjuntoResultado.getString("telefone"),
        conjuntoResultado.getString("email"),
        conjuntoResultado.getString("cpf")
    );
    }
    catch (SQLException e) {
        System.err.println("Erro ao recuperar pessoa
física: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao,          declaracao,
conjuntoResultado);
    }

    return null;
}

public List<PessoaFisica> getPessoas() throws
SQLException {
    List<PessoaFisica> pessoas = new ArrayList<>();

    Connection conexao = null;
    PreparedStatement declaracao = null;
    ResultSet conjuntoResultado = null;

    String sql = "SELECT p.*, pf.cpf FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa";

    try {
        conexao = ConectorBD.getConnection();
        declaracao = ConectorBD.getPrepared(conexao,
sql);
        conjuntoResultado =
ConectorBD.getSelect(declaracao);

        while (conjuntoResultado.next()) {
            PessoaFisica pessoa = new PessoaFisica(
                conjuntoResultado.getInt("idPessoa"),
```



# Estácio

```
        conjuntoResultado.getString("nome"),
        conjuntoResultado.getString("logradouro"),
        conjuntoResultado.getString("cidade"),
        conjuntoResultado.getString("estado"),
        conjuntoResultado.getString("telefone"),
        conjuntoResultado.getString("email"),
        conjuntoResultado.getString("cpf")
    );

    pessoas.add(pessoa);
}

System.out.println("\n=====");
System.out.println("PESSOAS FÍSICAS CADASTRADAS:");
for (PessoaFisica pessoa : pessoas) {
    pessoa.exibir();
}

System.out.println("=====\\n");
}
catch (SQLException e) {
    System.err.println("Erro ao recuperar pessoas físicas: " + e.getMessage());
}
finally {
    ConectorBD.close(conexao, declaracao, conjuntoResultado);
}

return pessoas;
}

public void incluir(PessoaFisica pessoa) throws SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
```



## Estácio

```
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        int id =
SequenceManager.getValue("SeqIdPessoa");
        pessoa.setId(id);

        String sqlPessoa = "INSERT INTO
Pessoa(idPessoa, nome, logradouro, cidade, estado,
telefone, email) VALUES(?, ?, ?, ?, ?, ?, ?)";
        declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.setString(2,
pessoa.getNome());
        declaracaoPessoa.setString(3,
pessoa.getLogradouro());
        declaracaoPessoa.setString(4,
pessoa.getCidade());
        declaracaoPessoa.setString(5,
pessoa.getEstado());
        declaracaoPessoa.setString(6,
pessoa.getTelefone());
        declaracaoPessoa.setString(7,
pessoa.getEmail());
        declaracaoPessoa.executeUpdate();

        String sqlPessoaFisica = "INSERT INTO
PessoaFisica(idPessoa, cpf) VALUES(?, ?)";
        declaracaoPessoaFisica =
ConectorBD.getPrepared(conexao, sqlPessoaFisica);
        declaracaoPessoaFisica.setInt(1, id);
        declaracaoPessoaFisica.setString(2,
pessoa.getCpf());
        declaracaoPessoaFisica.executeUpdate();

        conexao.commit();

        System.out.println("Pessoa física incluída
com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
            try {
                conexao.rollback();
            }
            catch (SQLException ex) {
                System.err.println("Erro ao tentar
encerrar conexão: " + ex.getMessage());
            }
        }
    }
}
```



# Estácio

```
        }
    }

    System.err.println("Erro ao tentar incluir
pessoa física: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaFisica);
    }
}

public void alterar(PessoaFisica pessoa) throws
SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        String sqlPessoa = "UPDATE Pessoa SET nome =
?, logradouro = ?, cidade = ?, estado = ?, telefone = ?,
email = ? WHERE idPessoa = ?";
        declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setString(1,
pessoa.getNome());
        declaracaoPessoa.setString(2,
pessoa.getLogradouro());
        declaracaoPessoa.setString(3,
pessoa.getCidade());
        declaracaoPessoa.setString(4,
pessoa.getEstado());
        declaracaoPessoa.setString(5,
pessoa.getTelefone());
        declaracaoPessoa.setString(6,
pessoa.getEmail());
        declaracaoPessoa.setInt(7, pessoa.getId());
        declaracaoPessoa.executeUpdate();

        String sqlPessoaFisica = "UPDATE
PessoaFisica SET cpf = ? WHERE idPessoa = ?";
        declaracaoPessoaFisica =
ConectorBD.getPrepared(conexao, sqlPessoaFisica);
        declaracaoPessoaFisica.setString(1,
pessoa.getCpf());
```



## Estácio

```
        declaracaoPessoaFisica.setInt(2,
        pessoa.getId());
        declaracaoPessoaFisica.executeUpdate();

        conexao.commit();

        System.out.println("Pessoa física alterada
        com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
            try {
                conexao.rollback();
            }
            catch (SQLException ex) {
                System.err.println("Erro ao tentar
                encerrar conexão: " + ex.getMessage());
            }
        }

        System.err.println("Erro ao tentar alterar a
        pessoa física: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaFisica);
    }
}

public void excluir(int id) throws SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        String sqlPessoaFisica = "DELETE FROM
        PessoaFisica WHERE idPessoa = ?";
        declaracaoPessoaFisica =
        ConectorBD.getPrepared(conexao, sqlPessoaFisica);
        declaracaoPessoaFisica.setInt(1, id);
        declaracaoPessoaFisica.executeUpdate();

        String sqlPessoa = "DELETE FROM Pessoa WHERE
        idPessoa = ?";
```



# Estácio

```
        declaracaoPessoa
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.executeUpdate();

        conexao.commit();

        System.out.println("Pessoa física excluída
com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
            try {
                conexao.rollback();
            }
            catch (SQLException ex) {
                System.err.println("Erro ao tentar
desfazer a transação: " + ex.getMessage());
            }
        }

        System.err.println("Erro ao tentar excluir
pessoa física: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaFisica);
    }
}
}
```

## Classe PessoaJuridicaDAO

```
package cadastrobd.model;

import cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaDAO {
    public PessoaJuridica getPessoa(int id) throws
SQLException {
```



## Estácio

```
String sql = "SELECT p.*, pj.cnpj FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa
WHERE idPessoa = ?";

Connection conexao = null;
PreparedStatement declaracao = null;
ResultSet conjuntoResultado = null;

try {
    conexao = ConectorBD.getConnection();
    declaracao = ConectorBD.getPrepared(conexao,
sql);
    declaracao.setInt(1, id);
    conjuntoResultado =
ConectorBD.select(declaracao);

    if (conjuntoResultado.next()) {
        return new PessoaJuridica(

conjuntoResultado.getInt("idPessoa"),
conjuntoResultado.getString("nome"),
conjuntoResultado.getString("logradouro"),
conjuntoResultado.getString("cidade"),
conjuntoResultado.getString("estado"),
conjuntoResultado.getString("telefone"),
conjuntoResultado.getString("email"),
conjuntoResultado.getString("cnpj")
        );
    }
} catch (SQLException e) {
    System.err.println("Erro ao tentar recuperar
pessoa jurídica: " + e.getMessage());
}
finally {
    ConectorBD.close(conexao, declaracao,
conjuntoResultado);
}

return null;
}
```





# Estácio

```
public List<PessoaJuridica> getPessoas() throws
SQLException {
    List<PessoaJuridica> pessoas = new ArrayList<>();

    Connection conexao = null;
    PreparedStatement declaracao = null;
    ResultSet conjuntoResultado = null;

    String sql = "SELECT p.*, pj.cnpj FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa";

    try {
        conexao = ConectorBD.getConnection();
        declaracao = ConectorBD.getPrepared(conexao,
sql);
        conjuntoResultado =
ConectorBD.getSelect(declaracao);

        while (conjuntoResultado.next()) {
            PessoaJuridica pessoa = new
PessoaJuridica(
                conjuntoResultado.getInt("idPessoa"),
                conjuntoResultado.getString("nome"),
                conjuntoResultado.getString("logradouro"),
                conjuntoResultado.getString("cidade"),
                conjuntoResultado.getString("estado"),
                conjuntoResultado.getString("telefone"),
                conjuntoResultado.getString("email"),
                conjuntoResultado.getString("cnpj")
            );

            pessoas.add(pessoa);
        }

        for (PessoaJuridica pessoa : pessoas) {
            pessoa.exibir();
        }
    }
    catch (SQLException e) {
```



## Estácio

```
        System.err.println("Erro ao recuperar
pessoas jurídicas: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracao,
conjuntoResultado);
    }

    return pessoas;
}

public void incluir(PessoaJuridica pessoa) throws
SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica =
null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        int id =
SequenceManager.getValue("SeqIdPessoa");
        pessoa.setId(id); // O ID gerado pelo banco
de dados é atribuído ao objeto carregado em memória.

        String sqlPessoa = "INSERT INTO
Pessoa(idPessoa, nome, logradouro, cidade, estado,
telefone, email) VALUES(?, ?, ?, ?, ?, ?, ?)";
        declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.setString(2,
pessoa.getNome());
        declaracaoPessoa.setString(3,
pessoa.getLogradouro());
        declaracaoPessoa.setString(4,
pessoa.getCidade());
        declaracaoPessoa.setString(5,
pessoa.getEstado());
        declaracaoPessoa.setString(6,
pessoa.getTelefone());
        declaracaoPessoa.setString(7,
pessoa.getEmail());
        declaracaoPessoa.executeUpdate();
    }
```



## Estácio

```
String sqlPessoaJuridica = "INSERT INTO
PessoaJuridica(idPessoa, cnpj) VALUES(?, ?)";
declaracaoPessoaJuridica =
ConectorBD.getPrepared(conexao, sqlPessoaJuridica);
declaracaoPessoaJuridica.setInt(1, id);
declaracaoPessoaJuridica.setString(2,
pessoa.getCnpj());
declaracaoPessoaJuridica.executeUpdate();

conexao.commit();

System.out.println("Pessoa jurídica incluída
com sucesso!");
}
catch (SQLException e) {
    if (conexao != null) {
        try {
            conexao.rollback();
        }
        catch (SQLException ex) {
            System.err.println("Erro ao tentar
encerrar a conexão: " + ex.getMessage());
        }
    }

    System.err.println("Erro ao tentar incluir
pessoa jurídica: " + e.getMessage());
}
finally {
    ConectorBD.close(conexao, declaracaoPessoa);
    ConectorBD.close(declaracaoPessoaJuridica);
}

}

public void alterar(PessoaJuridica pessoa) throws
SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica =
null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        String sqlPessoa = "UPDATE Pessoa SET nome =
?, logradouro = ?, cidade = ?, estado = ?, telefone = ?,
email = ? WHERE idPessoa = ?";
```



## Estácio

```
        declaracaoPessoa
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setString(1,
        pessoa.getNome());
        declaracaoPessoa.setString(2,
        pessoa.getLogradouro());
        declaracaoPessoa.setString(3,
        pessoa.getCidade());
        declaracaoPessoa.setString(4,
        pessoa.getEstado());
        declaracaoPessoa.setString(5,
        pessoa.getTelefone());
        declaracaoPessoa.setString(6,
        pessoa.getEmail());
        declaracaoPessoa.setInt(7, pessoa.getId());

        String      sqlPessoaJuridica      =      "UPDATE
PessoaJurirca SET cnpj = ? WHERE idPessoa = ?";
        declaracaoPessoaJuridica
ConectorBD.getPrepared(conexao, sqlPessoaJuridica);
        declaracaoPessoaJuridica.setString(1,
        pessoa.getCnpj());
        declaracaoPessoaJuridica.setInt(2,
        pessoa.getId());

        conexao.commit();

        System.out.println("Pessoa jurídica alterada
com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
            try {
                conexao.rollback();
            }
            catch (SQLException ex) {
                System.err.println("Erro ao tentar
desfazer a transação: " + ex.getMessage());
            }
        }

        System.err.println("Erro ao tentar alterar a
        pessoa jurídica: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaJuridica);
    }
}
```



# Estácio

```
}

    public void excluir(int id) throws SQLException {
        Connection conexao = null;
        PreparedStatement declaracaoPessoa = null;
        PreparedStatement declaracaoPessoaJuridica = null;

        try {
            conexao = ConectorBD.getConnection();
            conexao.setAutoCommit(false);

            String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE idPessoa = ?";
            declaracaoPessoaJuridica = ConectorBD.getPrepared(conexao, sqlPessoaJuridica);
            declaracaoPessoaJuridica.setInt(1, id);
            declaracaoPessoaJuridica.executeUpdate();

            String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
            declaracaoPessoa = ConectorBD.getPrepared(conexao, sqlPessoa);
            declaracaoPessoa.setInt(1, id);
            declaracaoPessoa.executeUpdate();

            conexao.commit();

            System.out.println("Pessoa jurídica excluída com sucesso!");
        } catch (SQLException e) {
            if (conexao != null) {
                try {
                    conexao.rollback();
                } catch (SQLException ex) {
                    System.err.println("Erro ao tentar desfazer a transação: " + ex.getMessage());
                }
            }

            System.err.println("Erro ao tentar excluir pessoa jurídica: " + e.getMessage());
        } finally {
            ConectorBD.close(conexao, declaracaoPessoa);
            ConectorBD.close(declaracaoPessoaJuridica);
        }
    }
}
```



```
    }  
}  
}
```

### **Pacote cadastrobd**

O pacote cadastrobd armazena a classe CadastroBDTeste, responsável pelo método main, porta de entrada para a aplicação Java.

Aqui serão realizados testes de execução do código escrito até o momento, que corresponde ao código do 1º Procedimento. Os testes envolvem instanciar objetos de pessoas físicas e jurídicas, incluir, alterar e excluir suas informações no banco de dados.

### **Classe CadastroBDTeste**

```
package cadastrobd;  
  
import cadastrobd.model.PessoaFisica;  
import cadastrobd.model.PessoaFisicaDAO;  
import cadastrobd.model.PessoaJuridica;  
import cadastrobd.model.PessoaJuridicaDAO;  
import java.sql.SQLException;  
  
public class CadastroBDTeste {  
    public static void main(String[] args) throws  
SQLException {  
        PessoaFisica pessoaFisica = new PessoaFisica(  
            0,  
            "João Silva",  
            "Rua das Flores, nº 123",  
            "São Paulo",  
            "SP",  
            "(11) 98765-4321",  
            "joao@email.com",  
            "12345678900"  
        );  
  
        PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();  
  
        pfDAO.incluir(pessoaFisica);  
  
        pessoaFisica.setLogradouro("Rua dos Cravos, nº  
456");
```



## Estácio

```
        pfDAO.alterar(pessoaFisica);

        pfDAO.getPessoas();

        pfDAO.excluir(pessoaFisica.getId());

        PessoaJuridica      pessoaJuridica      =      new
PessoaJuridica(
            0,
            "Tech Solutions Ltda.",
            "Rua das Empresas, nº 789",
            "Rio de Janeiro",
            "RJ",
            "(21) 98123-4567",
            "contato@techsolutions.com",
            "12345678000199"
        );

        PessoaJuridicaDAO      pjDAO      =      new
PessoaJuridicaDAO();

        pjDAO.incluir(pessoaJuridica);

        pessoaJuridica.setNome("Tech Solutions S.A.");
        pessoaJuridica.setLogradouro("Avenida      das
Empresas, nº 321");

        pjDAO.alterar(pessoaJuridica);

        pjDAO.getPessoas();

        pjDAO.excluir(pessoaJuridica.getId());
    }
}
```



## RESULTADOS

Após a execução do código escrito até aqui, temos a criação de uma pessoa física e uma pessoa jurídica, bem como suas respectivas inclusão, alteração e exclusão do banco de dados, além da exibição de todas as pessoas físicas e jurídicas presentes no banco de dados.

Na figura a seguir, podemos ver o resultado da execução do código na saída do console do NetBeans 24.

*Figura 1: Resultado do 1º procedimento*

```
run:
Pessoa fisica incluida com sucesso!
Pessoa fisica alterada com sucesso!

=====
PESSOAS FÍSICAS CADASTRADAS:
=====
ID: 1
Nome: João Silva
Logradouro: Rua dos Cravos, nº 456, São Paulo/SP
Telefone: (11) 98765-4321
E-mail: joao@email.com
CPF: 12345678900
=====

Pessoa fisica excluida com sucesso!
Pessoa juridica incluida com sucesso!
Pessoa juridica alterada com sucesso!

=====
PESSOAS JURÍDICAS CADASTRADAS:
=====
ID: 2
Nome: Tech Solutions Ltda.
Logradouro: Rua das Empresas, nº 789, Rio de Janeiro/RJ
Telefone: (21) 98123-4567
E-mail: contato@techsolutions.com
CNPJ: 12345678000199
=====

Pessoa juridica excluida com sucesso!
BUILD SUCCESSFUL (total time: 0 seconds)
```





## ANÁLISE

### a) Qual a importância dos componentes de *middleware*, como o JDBC?

Os componentes de *middleware*, como o JDBC (*Java Database Connectivity*), são fundamentais para permitir a comunicação entre uma aplicação Java e um banco de dados relacional. Alguns pontos que tornam tornam interessante o uso de uma API como essa são:

- **Abstração do banco de dados:** oferece recursos importantes como gerenciamento de conexões, transações e tratamento de erros, que seriam muito complexos de implementar manualmente. Isso permite que o código Java se comunique com o banco de dados sem precisar conhecer detalhes específicos do protocolo de comunicação
- **Facilidade de uso e manutenção:** simplifica operações como conexão, execução de consultas SQL e manipulação de resultados. No nosso projeto, isso é evidente no ConectorBD, que encapsula toda complexidade de conexão em métodos simples como `getConnection()`.
- **Portabilidade:** como o JDBC é parte do Java SE (*Java Standard Edition*), ele funciona em qualquer plataforma que suporte Java. Isso permite trocar o banco de dados (SQL Server, MySQL, Oracle, etc.) com mínimas alterações no código.
- **Segurança:** oferece suporte a transações, *prepared statements* e outros mecanismos que ajudam a evitar vulnerabilidades como injeção de SQL.

### b) Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?



O *Statement* é um objeto usado para a execução estática (sem parâmetros) de uma consulta SQL e retorna o resultado produzido. Já o *PreparedStatement* representa uma consulta SQL pré-compilada.

Na prática, se uma consulta SQL precisa ser executada várias vezes, pode ser mais interessante utilizar um *PreparedStatement*, que costuma reduzir o tempo de execução, pois a consulta já é pré-compilada, não sendo necessário que seja compilada pelo banco de dados a cada execução.

Em termos de segurança, o *PreparedStatement* mostra-se como uma melhor opção ao prevenir ataques de injeção de SQL. Isso porque no *Statement* os dados fornecidos pelo usuário são concatenados na *String* a ser utilizada na consulta, enquanto o *PreparedStatement* utiliza marcadores de posição (*placeholders*) para consultas parametrizadas e os parâmetros são tratados de forma segura.

### c) Como o padrão DAO melhora a manutenibilidade do software?

O DAO (*Data Access Object*) é um padrão de projeto que separa a lógica de acesso a dados (como consultas SQL) da lógica de negócios da aplicação.

A manutenibilidade do software é melhorada por:

- **Separação de responsabilidades:** o código de acesso ao banco de dados fica isolado em classes específicas (DAOs), facilitando a organização e o entendimento do código. Com isso, alterações na estrutura do banco ou na forma como os dados são manipulados podem ser feitas nas classes DAO sem impactar outras partes do sistema.
- **Reutilização de código:** operações comuns de CRUD (*Create, Read, Update, Delete*) podem ser reutilizadas em diferentes partes da aplicação.
- **Flexibilidade:** se o banco de dados for alterado, apenas as classes DAO precisam ser modificadas, sem afetar o restante da aplicação.



## Estácio

- **Testabilidade:** as classes DAO podem ser testadas unitariamente de forma mais fácil, pois não dependem da lógica de negócios.

### d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

Em bancos de dados relacionais, a herança (um conceito de orientação a objetos) não é diretamente suportada. No entanto, existem estratégias para mapear hierarquias de classes em tabelas:

- **Tabela única (*single table*):** Todas as classes da hierarquia são mapeadas para uma única tabela.
  - Uma coluna discriminadora indica a classe específica de cada registro.
  - Em nosso projeto, isso seria implementado utilizando apenas a tabela Pessoa para todos os tipos de pessoa, utilizando uma coluna tipo para indicar se se trata de pessoa física ou jurídica.
  - Pode gerar colunas vazias para atributos que não se aplicam a todas as classes. No nosso caso, um registro de pessoa física ficaria sempre com o atributo do CNPJ vazio e um registro de pessoa jurídica ficaria sempre com o atributo do CPF vazio.
- **Tabela por classe (*table per class*):** cada classe concreta tem sua própria tabela.
  - As tabelas filhas contêm colunas para os atributos da classe pai.
  - Repetição de atributos comuns.
  - Em nosso projeto, isso seria implementado utilizando classes para as tabelas PessoaFisica e PessoaJuridica, sem uma tabela para a classe Pessoa.



## Estácio

Isso faria com que campos comuns fossem repetidos em ambas tabelas, como os campos nome, logradouro, cidade etc.

- **Tabela por subclasse (*joined table*):** cada classe na hierarquia é mapeada para uma tabela, mas os atributos da classe pai são armazenados em uma tabela separada.
  - O relacionamento entre as tabelas se dá através de chaves estrangeiras (*foreign keys*).
  - Esta foi a abordagem escolhida para nosso projeto e apresenta vantagens e desvantagens.
    - **Vantagens**
      - Não há duplicação de campos comuns.
      - Integridade referencial garantida pelo banco de dados.
      - Facilidade para consultas específicas de cada tipo.
    - **Desvantagens**
      - Necessidade de *joins* para recuperar dados completos.
      - Operações de inclusão e exclusão precisam tratar múltiplas tabelas.
      - Transações são necessárias para manter consistência.



# Estácio

## 2º Procedimento | Alimentando a Base

### CÓDIGOS

#### Pacote cadastrobd.view

O pacote cadastro.view armazenará a classe CadastroConsole, responsável pelas funcionalidades de interação com o usuário, possuindo os métodos com os menus adequados ao caminho percorrido pelo usuário no sistema.

#### Classe CadastroConsole

```
package cadastrobd.view;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;

public class CadastroConsole {
    private static final Scanner sc = new
Scanner(System.in);
    private static final PessoaFisicaDAO pfDAO = new
PessoaFisicaDAO();
    private static final PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO();

    public CadastroConsole() {}

    public void iniciar() {
        int opcao;

        do {
            menu();
            opcao = lerOpcao();

            try {
                processarOpcao(opcao);
            }
            catch (SQLException e) {
                System.err.println("Erro: " +
e.getMessage());
            }
        }
    }
}
```



## Estácio

```
        } while (opcao != 0);
    }

    private static void menu() {
        System.out.println("\n===== SISTEMA DE CADASTRO
DE PESSOAS =====");
        System.out.println("0 que deseja fazer?");
        System.out.println("1 - Incluir");
        System.out.println("2 - Alterar");
        System.out.println("3 - Excluir");
        System.out.println("4 - Exibir por ID");
        System.out.println("5 - Exibir todos");
        System.out.println("0 - Sair");
        System.out.print("\nOpção: ");
    }

    private static int lerOpcao() {
        try {
            return Integer.parseInt(sc.nextLine());
        }
        catch (NumberFormatException e) {
            System.err.println("Erro ao ler a opção: a
entrada deve ser um NÚMERO.");
            return -1;
        }
    }

    private static void processarOpcao(int opcao) throws
SQLException {
        switch (opcao) {
            case 1 -> incluirPessoa();
            case 2 -> alterarPessoa();
            case 3 -> excluirPessoa();
            case 4 -> exibirPessoa();
            case 5 -> exibirTodos();
            case 0 -> {
                System.out.println("\nEncerrando o
sistema...");
                System.out.println("Tchau! Até a
próxima :)\n");

                break;
            }
            default -> System.err.println("\nOPÇÃO
INVÁLIDA. Escolha um número entre 0(zero) e 5 (cinco).\n");
        }
    }

    private static char escolherTipoPessoa() {
```



## Estácio

```
        System.out.print("\nPessoa física (F) ou jurídica
(J)? ");
        return sc.nextLine().toUpperCase().charAt(0);
    }

    private static void incluirPessoa() throws
SQLException {
        char tipoPessoa = escolherTipoPessoa();

        System.out.print("Nome: ");
        String nome = sc.nextLine();
        System.out.print("Logradouro: ");
        String logradouro = sc.nextLine();
        System.out.print("Cidade: ");
        String cidade = sc.nextLine();
        System.out.print("Estado: ");
        String estado = sc.nextLine();
        System.out.print("Telefone: ");
        String telefone = sc.nextLine();
        System.out.print("E-mail: ");
        String email = sc.nextLine();

        switch (tipoPessoa) {
            case 'F' -> {
                System.out.print("CPF (apenas números):
");
                String cpf = sc.nextLine();
                PessoaFisica pf = new PessoaFisica(0, nome,
logradouro, cidade, estado, telefone, email, cpf);
                pfDAO.incluir(pf);
            }
            case 'J' -> {
                System.out.print("CNPJ (apenas
números): ");
                String cnpj = sc.nextLine();
                PessoaJuridica pj = new
PessoaJuridica(0, nome, logradouro, cidade, estado,
telefone, email, cnpj);
                pjDAO.incluir(pj);

                System.out.println("\nPessoa jurídica
incluída com sucesso! ID: " + pj.getId());
            }
            default -> System.err.println("\nTipo de
pessoa inválido.\n");
        }
    }
}
```



## Estácio

```
private static void alterarPessoa() throws
SQLException {
    char tipoPessoa = escolherTipoPessoa();

    System.out.print("Informe o ID: ");
    int id = Integer.parseInt(sc.nextLine());

    switch (tipoPessoa) {
        case 'F' -> {
            PessoaFisica pf = pfDAO.getPessoa(id);

            if (pf != null) {
                System.out.println("\nDados atuais:");
                pf.exibir();

                System.out.println("\nInforme os novos
dados:");

                System.out.print("Nome: ");
                pf.setNome(sc.nextLine());
                System.out.print("Logradouro: ");
                pf.setLogradouro(sc.nextLine());
                System.out.print("Cidade: ");
                pf.setCidade(sc.nextLine());
                System.out.print("Estado: ");
                pf.setEstado(sc.nextLine());
                System.out.print("Telefone: ");
                pf.setTelefone(sc.nextLine());
                System.out.print("E-mail: ");
                pf.setEmail(sc.nextLine());
                System.out.print("CPF: ");
                pf.setCpf(sc.nextLine());

                pfDAO.alterar(pf);
                System.out.println("\nDados da pessoa
física alterados com sucesso!\n");
            }
            else {
                System.err.println("\nPessoa
física não encontrada.\n");
            }
        }
        case 'J' -> {
            PessoaJuridica pj =
pjDAO.getPessoa(id);

            if (pj != null) {
                System.out.println("Informe os
novos dados:");
```





## Estácio

```
        System.out.print("Nome: ");
        pj.setNome(sc.nextLine());
        System.out.print("Logradouro: ");
        pj.setLogradouro(sc.nextLine());
        System.out.print("Cidade: ");
        pj.setCidade(sc.nextLine());
        System.out.print("Estado: ");
        pj.setEstado(sc.nextLine());
        System.out.print("Telefone: ");
        pj.setTelefone(sc.nextLine());
        System.out.print("E-mail: ");
        pj.setEmail(sc.nextLine());
        System.out.print("CPF: ");
        pj.setCnpj(sc.nextLine());

        pjDAO.alterar(pj);
        System.out.println("\nDados      da
pessoa jurídica alterados com sucesso!\n");
    }
    else {
        System.err.println("\nPessoa
jurídica não encontrada.\n");
    }
    }
    default -> System.err.println("\nTipo  de
pessoa inválido.\n");
}

private static void excluirPessoa() throws
SQLException {
    char tipoPessoa = escolherTipoPessoa();

    System.out.print("Informe o ID: ");
    int id = Integer.parseInt(sc.nextLine());

    switch(tipoPessoa) {
        case 'F' -> {
            PessoaFisica pf = pfDAO.getPessoa(id);

            if (pf != null) {
                System.out.println("\nDados      da
pessoa a ser excluída:");

                System.out.println("-----");
                pf.exibir();

                System.out.println("-----");
            }
        }
    }
}
```



```

        System.out.print("\nATENÇÃO! Tem
certeza que deseja excluir (S/N)? ");
        switch
(sc.nextLine().toUpperCase().charAt(0)) {
            case 'S' -> {
                pfDAO.excluir(id);

                System.out.println("Pessoa física excluída com
sucesso!\n");
            }
            case 'N' ->
System.out.println("Pessoa física não excluída. Registro
mantido.\n");
            default ->
System.err.println("Opção desconhecida. Registro mantido.\n
n");
        }
    }
    else {
        System.err.println("Pessoa física
não encontrada.\n");
    }
}
case 'J' -> {
    PessoaJuridica pj =
pjDAO.getPessoa(id);

    if (pj != null) {
        System.out.println("\nDados da
pessoa a ser excluída:");
        pj.exibir();

        System.out.print("ATENÇÃO! Tem
certeza que deseja excluir (S/N)? ");
        switch
(sc.nextLine().toUpperCase().charAt(0)) {
            case 'S' -> {
                pjDAO.excluir(id);

                System.out.println("Pessoa jurídica excluída com
sucesso!\n");
            }
            case 'N' ->
System.out.println("Pessoa física não excluída. Registro
mantido.\n");
            default ->
System.err.println("Opção desconhecida. Registro mantido.\n
n");
        }
    }
}
}

```



## Estácio

```
        }
    }
    else {
        System.err.println("Pessoa jurídica não
encontrada.\n");
    }
}
default -> System.err.println("\nTipo de
pessoa inválido.\n");
}
}

private static void exibirPessoa() throws SQLException
{
    char tipoPessoa = escolherTipoPessoa();

    System.out.print("Informe o ID: ");
    int id = Integer.parseInt(sc.nextLine());

    switch(tipoPessoa) {
        case 'F' -> {
            PessoaFisica pf = pfDAO.getPessoa(id);

            if (pf != null) {
                System.out.println("\n
-----");
                pf.exibir();

                System.out.println("-----\n");
            }
            else {
                System.err.println("\nPessoa
física não encontrada.\n");
            }
        }
        case 'J' -> {
            PessoaJuridica pj =
pjDAO.getPessoa(id);

            if (pj != null) {
                pj.exibir();
            }
            else {
                System.err.println("\nPessoa
jurídica não encontrada.\n");
            }
        }
    }
}
```



## Estácio

```
        default -> System.err.println("\nTipo de
        pessoa inválido.\n");
    }
}

private static void exibirTodos() throws SQLException
{
    char tipoPessoa = escolherTipoPessoa();

    switch (tipoPessoa) {
        case 'F' -> {
            List<PessoaFisica> pessoas =
pfDAO.getPessoas();

            if (pessoas.isEmpty()) {
                System.out.println("\nNenhuma
                pessoa física cadastrada.\n");
            }
            else {
                System.out.println("\nPessoas
                físicas cadastradas:");
                for (PessoaFisica pessoa :
                pessoas) {
                    System.out.println("\
                n-----");
                    pessoa.exibir();
                }
            }
        }
        case 'J' -> {
            List<PessoaJuridica> pessoas =
pjDAO.getPessoas();

            if (pessoas.isEmpty()) {
                System.out.println("\nNenhuma
                pessoa jurídica cadastrada.\n");
            }
            else {
                System.out.println("\nPessoas
                juridicas cadastradas:");
                for (PessoaJuridica pessoa :
                pessoas) {
                    System.out.println("\
                n-----");
                    pessoa.exibir();
                }
            }
        }
    }
}
```



```
                default -> System.err.println("\nTipo de  
pessoa inválido.\n");  
            }  
        }  
    }  
}
```

### **Pacote cadastrobd**

O cadastrobd armazena apenas a classe Main, responsável pelo método main() (porta de entrada da aplicação).

Com as alterações de interface realizadas, o método main(), por sua vez, ficou significativamente mais conciso, em comparação com o procedimento anterior (1º Procedimento), contendo apenas a criação do objeto console, instância da classe CadastroConsole, e a chamada do método iniciar().

### **Classe Main**

```
package cadastrobd;  
  
import cadastrobd.view.CadastroConsole;  
  
public class Main {  
    public static void main(String[] args) {  
        CadastroConsole console = new CadastroConsole();  
        console.iniciar();  
    }  
}
```

## **RESULTADOS**

Após a execução do código escrito até aqui, a utilização de um menu de console agindo como uma interface de texto para a interação do usuário com a aplicação.

Nas figuras que seguem, podemos ver diversas saídas em uma simulação da atuação do usuário na realização das diferentes operações oferecidas pelo sistema.



Figura 2: Exemplo de procedimento de inclusão de pessoa

```
Output - CadastroBD (run) x
run:
===== SISTEMA DE CADASTRO DE PESSOAS =====
0 que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 1

Pessoa física (F) ou jurídica (J)? F
Nome: Fulano
Logradouro: Rua dos Cravos, nº 123
```

Figura 3: Exemplo de procedimento de alteração de pessoa

```
Output - CadastroBD (run) x
run:
===== SISTEMA DE CADASTRO DE PESSOAS =====
0 que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 2

Pessoa física (F) ou jurídica (J)? F
Informe o ID: 1

Dados atuais:
ID: 1
Nome: Fulano
Logradouro: Rua dos Cravos, nº 123, Fortaleza/CE
Telefone: (85) 99999-9999
E-mail: fulano@email.com
CPF: 11111111111

Informe os novos dados:
Nome: Fulano de Tal
Logradouro: Rua dos Cravos, nº 123
Cidade: Fortaleza
Estado: CE
Telefone: (85) 99999-9999
E-mail: fulano@email.com
CPF: 11111111111

Dados da pessoa física alterados com sucesso!
```



Figura 4: Exemplo de procedimento de exclusão de pessoa

```
Output - CadastroBD (run) x
run:

===== SISTEMA DE CADASTRO DE PESSOAS =====
0 que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 3

Pessoa física (F) ou jurídica (J)? J
Informe o ID: 6

Dados da pessoa a ser excluída:
-----
ID: 6
Nome: Empresa XPTO S.A.
Logradouro: Avenida das Companhias, nº 345, Manaus/AM
Telefone: (92) 4555-5555
E-mail: contato@xpto.com.br
CNPJ: 2222222000100
-----

ATENÇÃO! Tem certeza que deseja excluir (S/N)? S
Pessoa jurídica excluída com sucesso!
```



Figura 5: Exemplo de procedimento de exibição de dados de uma pessoa identificada por seu ID

```
Output - CadastroBD (run) x
run:
===== SISTEMA DE CADASTRO DE PESSOAS =====
O que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 4

Pessoa física (F) ou jurídica (J)? F
Informe o ID: 2

-----
ID: 2
Nome: Sicrano
Logradouro: Alameda das Rosas, nº 456, Rio de Janeiro/RJ
Telefone: (21) 98888-8888
E-mail: sicrano@email.com
CPF: 22222222222
-----
```



Figura 6: Exemplo de procedimento de exibição dos dados de todas as pessoas físicas cadastradas

```
Output - CadastroBD (run) x
run:

===== SISTEMA DE CADASTRO DE PESSOAS =====
O que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 5

Pessoa física (F) ou jurídica (J)? F

Pessoas físicas cadastradas:

-----
ID: 1
Nome: Fulano de Tal
Logradouro: Rua dos Cravos, nº 123, Fortaleza/CE
Telefone: (85) 99999-9999
E-mail: fulano@email.com
CPF: 11111111111

-----
ID: 2
Nome: Sicrano
Logradouro: Alameda das Rosas, nº 456, Rio de Janeiro/RJ
Telefone: (21) 98888-8888
E-mail: sicrano@email.com
CPF: 22222222222

-----
ID: 3
Nome: Beltrano
Logradouro: Avenida dos Cravos, nº 789, São Paulo/SP
Telefone: (11) 97777-7777
E-mail: beltrano@email.com
CPF: 33333333333
```

Figura 7: Exemplo de procedimento de exibição dos dados de todas as pessoas jurídicas cadastradas e saída do sistema

```
Output - CadastroBD (run) x
run:

===== SISTEMA DE CADASTRO DE PESSOAS =====
0 que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 5

Pessoa física (F) ou jurídica (J)? J

Pessoas juridicas cadastradas:

-----
ID: 4
Nome: Empresa XYZ Ltda.
Logradouro: Rua das Empresas, nº 567, Belo Horizonte/MG
Telefone: (31) 3444-4444
E-mail: contato@xyz.com.br
CNPJ: 11111111000100

===== SISTEMA DE CADASTRO DE PESSOAS =====
0 que deseja fazer?
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir por ID
5 - Exibir todos
0 - Sair

Opção: 0

Encerrando o sistema...
Tchau! Até a próxima :)
```



## ANÁLISE

### a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

A persistência de dados pode ser feita tanto em arquivos quanto em bancos de dados, e cada abordagem tem suas características e usos específicos. Aqui estão as principais diferenças:

CARACTERÍSTICA	PERSISTÊNCIA EM ARQUIVO	PERSISTÊNCIA EM BANCO DE DADOS
<b>Estrutura</b>	Dados armazenados em arquivos de texto, JSON, XML, CSV, binários, etc.	Dados organizados em tabelas relacionais com chaves e índices.
<b>Consulta e Recuperação</b>	Requer leitura e processamento manual dos arquivos.	Usa SQL para buscar e manipular dados de forma eficiente.
<b>Performance</b>	Pode ser lenta para grandes volumes de dados, pois exige leitura sequencial.	Muito mais rápida, com otimizações como indexação e cache.
<b>Concorrência</b>	Difícil de gerenciar; risco de corromper arquivos quando múltiplos usuários escrevem simultaneamente.	Suporte a múltiplos acessos simultâneos com controle de transações.
<b>Escalabilidade</b>	Limitada; conforme o volume de dados cresce, a busca e escrita ficam mais lentas.	Projetado para suportar grandes volumes de dados e acessos concorrentes.
<b>Integridade dos Dados</b>	Nenhum controle automático de integridade.	Suporte a chaves primárias, estrangeiras e transações ACID.

Em resumo, o uso de arquivos é mais adequado para cenários simples e de pequena escala, já o uso de bancos de dados é melhor para aplicações mais complexas e escaláveis.



**b) Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?**

O operador lambda (->) foi introduzido no Java 8 e trouxe uma forma mais concisa e funcional de trabalhar com coleções e *streams*, o que simplificou a impressão de valores contidos em entidades.

Em nosso código podemos ver a utilização de operadores lambda em diversas situações de uso de estruturas *rule switch*, tornando o código mais conciso e legível. Se não, vejamos.

A seguir um trecho de código da classe `CadastroConsole`, do pacote `cadastrobd.view`:

```
private static void processarOpcao(int opcao) throws
SQLException {
    switch (opcao) {
        case 1 -> incluirPessoa();
        case 2 -> alterarPessoa();
        case 3 -> excluirPessoa();
        case 4 -> exibirPessoa();
        case 5 -> exibirTodos();
        case 0 -> {
            System.out.println("\nEncerrando o
sistema...");
            System.out.println("Tchau! Até a próxima :) \n");
            break;
        }
        default -> System.err.println("\nOPÇÃO INVÁLIDA.
Escolha um número entre 0(zero) e 5 (cinco).\n");
    }
}
```

De outra forma escrito, utilizando a estrutura *switch* clássica, teríamos:



```
private static void processarOpcao(int opcao) throws
SQLException {
    switch (opcao) {
        case 1:
            incluirPessoa();
            break;
        case 2:
            alterarPessoa();
            break;
        case 3:
            excluirPessoa();
            break;
        case 4:
            exibirPessoa();
            break;
        case 5:
            exibirTodos();
            break;
        case 0:
            System.out.println("\nEncerrando o
sistema...");
            System.out.println("Tchau! Até a próxima :)\\
n");
            break;
        default:
            System.err.println("\nOPÇÃO INVÁLIDA.
Escolha um número entre 0(zero) e 5 (cinco).\\n");
            break;
    }
}
```

**c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static?**

Métodos estáticos (*static*) pertencem a uma classe, mas não necessitam da instanciação de sua classe respectiva (criação de objeto) para serem usados, podem ser acessados diretamente pelo nome da classe ou diretamente no contexto estático; métodos não estáticos necessitam da instanciação de sua classe para serem utilizados. Esse comportamento de métodos estáticos é particularmente interessante para o método main.

Como o método main é o ponto de entrada de uma aplicação Java e é chamado pela JVM antes da instanciação de qualquer objeto.



No geral, métodos estáticos são úteis para funções utilitárias que não precisam de um contexto específico, como métodos de impressão, leitura de dados, etc.

Em nosso projeto, na classe `CadastroConsole` todos os métodos, exceto o `iniciar()`, são estáticos podendo ser acessados diretamente pelo nome da classe. Optou-se pelo uso do método `iniciar()`, que não é estático, para chamá-los por uma questão de melhor organização do código.

## Conclusão

A prática foi bem-sucedida em demonstrar a integração de uma aplicação Java com um banco de dados SQL Server, reforçando o uso do JDBC para comunicação com o banco.

Durante o desenvolvimento, foi possível identificar desafios relacionados à conexão, manipulação de dados e organização do código.

Entre os principais aprendizados e pontos positivos, destacam-se:

- ✓ Uso eficiente de banco de dados para armazenar informações estruturadas.
- ✓ Aplicação de expressões lambda para simplificar a leitura e exibição de dados.
- ✓ Modularização do código, tornando a aplicação mais organizada e reutilizável.
- ✓ Compreensão do papel dos métodos estáticos, especialmente no contexto do `main()`.

Em resumo, a prática permitiu consolidar conhecimentos essenciais de Java, banco de dados e boas práticas de programação, preparando o caminho para o desenvolvimento de aplicações mais avançadas e escaláveis no futuro.