



**Estácio**

UNIVERSIDADE ESTÁCIO DE SÁ

CAMPUS POLO CENTRO - MARICÁ - RJ

**ALUNO:** LUIZ CARLOS MARINHO JUNIOR

**MATRÍCULA:** 2023.11.17557-1

**TURMA:** 9001

**SEMESTRE LETIVO:** 2024.4

**DISCIPLINA:** NÍVEL 3: BACK-END SEM BANCO NÃO TEM

**TÍTULO: MISSÃO PRÁTICA - NÍVEL 3**

RIO DE JANEIRO

2024

LUIZ CARLOS MARINHO JUNIOR

**MISSÃO PRÁTICA - NÍVEL 3**

Trabalho prático para aprovação na disciplina de Nível 3: Back-end sem banco não tem.

Tutora: Prof. Maria B.

RIO DE JANEIRO

2024

## SUMÁRIO

<b>1. OBJETIVOS DA PRÁTICA.....</b>	<b>4</b>
<b>2. INFORMAÇÕES COMPLEMENTARES .....</b>	<b>4</b>
<b>2.1. APRESENTAÇÃO.....</b>	<b>4</b>
<b>2.2. ORGANIZAÇÃO DO REPOSITÓRIO REMOTO.....</b>	<b>4</b>
<b>3. 1º PROCEDIMENTO   MAPEAMENTO OBJETO-RELACIONAL E DAO .....</b>	<b>5</b>
<b>3.1. CÓDIGOS SOLICITADOS.....</b>	<b>5</b>
<b>3.2. PRINCIPAIS INSTRUÇÕES DO 1ª PROCEDIMENTO.....</b>	<b>5</b>
<b>3.3. RESULTADO DOS CÓDIGOS.....</b>	<b>20</b>
<b>3.4. ANÁLISE E CONCLUSÃO.....</b>	<b>20</b>
<b>3.4.1. a) QUAL A IMPORTÂNCIA DOS COMPONENTES DE MIDDLEWARE, COMO O JDBC?.....</b>	<b>20</b>
<b>3.4.2. b) QUAL A DIFERENÇA NO USO DE STATEMENT OU PREPAREDSTATEMENT PARA A MANIPULAÇÃO DE DADOS?.....</b>	<b>20</b>
<b>3.4.3. c) COMO O PADRÃO DAO MELHORA A MANUTENIBILIDADE DO SOFTWARE?.....</b>	<b>21</b>
<b>3.4.4. d) COMO A HERANÇA É REFLETIDA NO BANCO DE DADOS, QUANDO LIDAMOS COM UM MODELO ESTRITAMENTE RELACIONAL?.....</b>	<b>21</b>
<b>4. 2º PROCEDIMENTO   ALIMENTANDO A BASE.....</b>	<b>22</b>
<b>4.1. CÓDIGOS SOLICITADOS.....</b>	<b>22</b>
<b>4.2. PRINCIPAIS INSTRUÇÕES DO 2º PROCEDIMENTO.....</b>	<b>22</b>
<b>4.3. RESULTADO DOS CÓDIGOS.....</b>	<b>34</b>
<b>4.4. ANÁLISE E CONCLUSÃO.....</b>	<b>34</b>
<b>4.4.1. a) QUAIS AS DIFERENÇAS ENTRE A PERSISTÊNCIA EM ARQUIVO E A PERSISTÊNCIA EM BANCO DE DADOS? .....</b>	<b>34</b>
<b>4.4.2. b) COMO O USO DE OPERADOR LAMBDA SIMPLIFICOU A IMPRESSÃO DOS VALORES CONTIDOS NAS ENTIDADES, NAS VERSÕES MAIS RECENTES DO JAVA?.....</b>	<b>35</b>
<b>4.4.3. c) POR QUE MÉTODOS ACIONADOS DIRETAMENTE PELO MÉTODO MAIN, SEM O USO DE UM OBJETO, PRECISAM SER MARCADOS COMO STATIC?.....</b>	<b>35</b>

## **1. OBJETIVOS DA PRÁTICA**

1. Implementar persistência com base no middleware JDBC.
2. Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
3. Implementar o mapeamento objeto-relacional em sistemas Java.
4. Criar sistemas cadastrais com persistência em banco relacional.
5. No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL
6. Server na persistência de dados.

## **2. INFORMAÇÕES COMPLEMENTARES**

### **2.1. APRESENTAÇÃO**

Esta missão prática está dividida em 2 procedimentos, o “1º Procedimento | Mapeamento Objeto-Relacional e DAO” e o “2º Procedimento | Alimentando a Base”. Todos os códigos que serão aqui apresentados, estão disponíveis de forma completa no repositório do Github.

O link para o repositório do Github se encontra logo abaixo:  
<https://github.com/luizmarinhojr/cadastro-BD>

### **2.2. ORGANIZAÇÃO DO REPOSITÓRIO REMOTO**

O repositório remoto do Github, apresentado no link a seguir, está divido em 3 branchs, a branch “main”, que se refere ao desenvolvimento completo da missão, a branch “PrimeiroProcedimento”, que se refere ao desenvolvimento completo do 1º Procedimento, e a branch “SegundoProcedimento”, que se refere ao desenvolvimento completo do 2º Procedimento.

As seguintes branchs podem ser acessadas diretamente a partir dos seguintes links:

Branch main: <https://github.com/luizmarinhojr/cadastro-BD>

Branch PrimeiroProcedimento: <https://github.com/luizmarinhojr/cadastro-BD/tree/PrimeiroProcedimentoento>

Branch SegundoProcedimento: <https://github.com/luizmarinhojr/cadastro-BD/tree/SegundoProcedimento>

### **3. 1º PROCEDIMENTO | MAPEAMENTO OBJETO-RELACIONAL E DAO**

#### **3.1. CÓDIGOS SOLICITADOS**

Os códigos serão demonstrados neste relatório de acordo com as instruções apresentadas no enunciado da missão prática. Somente serão demonstrados nesse relatório as principais instruções que são relevantes ao desenvolvimento do código.

#### **3.2. PRINCIPAIS INSTRUÇÕES DO 1<sup>a</sup> PROCEDIMENTO**

3. Voltando ao projeto, criar o pacote cadastrobd.model, e nele criar as classes apresentadas a seguir:
  - a. Classe Pessoa, com os campos id, nome, logradouro, cidade, estado, telefone e email, construtor padrão e completo, além de método exibir, para impressão dos dados no console.

Código na próxima página...

```

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 String exibir() {
        return String.format(
            """
            ID: %d
            Nome: %s
            Logradouro: %s
            Cidade: %s
            Estado: %s
            Telefone: %s
            Email: %s
            """,
            getId(), getNome(), getLogradouro(), getCidade(), getEstado(),
            getTelefone(), getEmail()
        );
    }
}

```

- b. Classe PessoaFisica, herdando de Pessoa, com acréscimo do campo cpf, além da reescrita dos construtores e uso de polimorfismo em exibir.

Código na próxima página...

```

package cadastrobd.model;

public class PessoaFisica extends Pessoa {

    private String cpf;

    public PessoaFisica() {}

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

    public PessoaFisica(Pessoa pessoa, String cpf) {
        super(pessoa.getId(), pessoa.getNome(), pessoa.getLogradouro(), pessoa.getCidade(),
              pessoa.getEstado(), pessoa.getTelefone(), pessoa.getEmail());
        this.cpf = cpf;
    }

    @Override
    public String exibir() {
        return String.format(
            """
            ID: %d
            Nome: %s
            Logradouro: %s
            Cidade: %s
            Estado: %s
            Telefone: %s
            Email: %s
            CPF: %s
            """,
            getId(), getNome(), getLogradouro(), getCidade(), getEstado(),
            getTelefone(), getEmail(), getCpf()
        );
    }

    public String getCpf() {
        return cpf;
    }
}

```

- c. Classe PessoaJuridica, herdando de Pessoa, com acréscimo do campo cnpj, além da reescrita dos construtores e uso de polimorfismo em exibir.

Código na próxima página...

```

package cadastrobd.model;

public class PessoaJuridica extends Pessoa {

    private String cnpj;

    public PessoaJuridica() {}

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

    public PessoaJuridica(Pessoa pessoa, String cnpj) {
        super(pessoa.getId(), pessoa.getNome(), pessoa.getLogradouro(), pessoa.getCidade(),
            pessoa.getEstado(), pessoa.getTelefone(), pessoa.getEmail());
        this.cnpj = cnpj;
    }

    @Override
    public String exibir() {
        return String.format(
            """
                ID: %d
                Nome: %s
                Logradouro: %s
                Cidade: %s
                Estado: %s
                Telefone: %s
                Email: %s
                CNPJ: %s
                """,
                getId(), getNome(), getLogradouro(), getCidade(), getEstado(),
                getTelefone(), getEmail(), getCnpj()
        );
    }

    public String getCnpj() {
        return cnpj;
    }
}

```

4. Criar o pacotes `cadastro.model.util`, para inclusão das classes utilitárias que são apresentadas a seguir:
  - a. Classe `ConectorBD`, com os métodos `getConnection`, para retornar uma conexão com o banco de dados, `getPrepared`, para retornar um objeto do tipo `PreparedStatement` a partir de um SQL fornecido com parâmetro, e `getSelect`, para retornar o `ResultSet` relacionado a uma consulta.

Código na próxima página...

```
package cadastrobd.model.util;

import java.sql.*;

public class ConectorBD {

    private static final String URL =
        "jdbc:sqlserver://172.17.0.2:1433;databaseName=Loja;encrypt=true;trustServerCertificate=true;";
    private static final String USER = "loja";
    private static final String PASSWORD = "loja";

    public Connection getConnection() throws SQLException {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }

        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

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

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

- b. Ainda na classe ConectorBD, adicionar métodos close sobre carregados para Statement, ResultSet e Connection, visando garantir o fechamento, ou encerramento, de todos os objetos de acesso ao banco gerados.

Código na próxima página...

```

public void close(PreparedStatement declaracao) {
    if (declaracao != null) {
        try {
            declaracao.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

public void close	ResultSet resultado) {
    if (resultado != null) {
        try {
            resultado.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

public void close(Connection conexao) {
    if (conexao != null) {
        try {
            conexao.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

public void close(Connection conexao, PreparedStatement declaracao, ResultSet resultado) {
    close(resultado);
    close(declaracao);
    close(conexao);
}
}

```

- c. Classe SequenceManager, que terá o método getValue, recebendo o nome da sequência como parâmetro e retornando o próximo valor.

Código na próxima página...

```

package cadastrobd.model.util;

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

public class SequenceManager {

    private static final ConecorBD conector = new ConecorBD();

    // Pega o próximo valor da chave primária
    public static int getValue(String nomeSequencia) {
        int valor = -1;
        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet resultado = null;

        try {
            conexao = conector.getConnection();
            declaracao = conector.getPrepared(conexao, String.format(
                "SELECT NEXT VALUE FOR %s AS proximo_id;", nomeSequencia));
            resultado = conector.getSelect(declaracao);
            if (resultado.next()) {
                valor = resultado.getInt("proximo_id");
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } finally {
            conector.close(conexao, declaracao, resultado);
        }

        return valor;
    }
}

```

5. Codificar as classes no padrão DAO, no pacote cadastro.model.

- Classe PessoaFisicaDAO, com os métodos getPessoa, retornando uma pessoa física a partir do seu id, getPessoas, para retorno de todas as pessoas físicas do banco de dados, incluir, para inclusão de uma pessoa física, fornecida como parâmetro, nas tabelas Pessoa e PessoaFisica, alterar, para alteração dos dados de uma pessoa física, e excluir, para remoção da pessoa do banco em ambas as tabelas.

```

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 {

    private static final ConectorBD conector = new ConectorBD();

    public PessoaFisica getPessoa(int id) {
        String sqlGetPessoa =
            "SELECT * FROM Pessoas p JOIN PessoasFisicas pf ON p.pessoaID = pf.pessoaID WHERE p.pessoaID = ?";
        PessoaFisica pessoaFisica = null;
        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet resultado = null;

        try {
            conexao = conector.getConnection();
            declaracao = conector.getPreparedStatement(conexao, sqlGetPessoa);
            declaracao.setInt(1, id);
            resultado = conector.getSelect(declaracao);
            if (resultado.next()) {
                pessoaFisica = new PessoaFisica(
                    resultado.getString("cpf"),
                    resultado.getInt("pessoaID"),
                    resultado.getString("nome"),
                    resultado.getString("logradouro"),
                    resultado.getString("cidade"),
                    resultado.getString("estado"),
                    resultado.getString("telefone"),
                    resultado.getString("email")
                );
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } finally {
            conector.close(conexao, declaracao, resultado);
        }

        return pessoaFisica;
    }

    public List<PessoaFisica> getPessoas() {
        String sql = "SELECT * FROM Pessoas p JOIN PessoasFisicas pf ON p.pessoaID = pf.pessoaID";
        List<PessoaFisica> pessoasFisicas = new ArrayList<>();
        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet resultado = null;

        try {
            conexao = conector.getConnection();
            declaracao = conector.getPreparedStatement(conexao, sql);
            resultado = conector.getSelect(declaracao);
            while (resultado.next()) {
                pessoasFisicas.add(new PessoaFisica(
                    resultado.getString("cpf"),
                    resultado.getInt("pessoaID"),
                    resultado.getString("nome"),
                    resultado.getString("logradouro"),
                    resultado.getString("cidade"),
                    resultado.getString("estado"),
                    resultado.getString("telefone"),
                    resultado.getString("email")
                ));
            }
        } catch (SQLException e) {
            System.out.println("ERRO" + e.getMessage());
        } finally {
            conector.close(conexao, declaracao, resultado);
        }

        return pessoasFisicas;
    }
}

```

```

public void incluir(PessoaFisica pessoaFisica) {
    String sqlPessoa =
        "INSERT INTO Pessoas (pessoaID, nome, logradouro, cidade, estado, telefone, email) " +
        "VALUES (?, ?, ?, ?, ?, ?, ?)";
    String sqlPessoaFisica = "INSERT INTO PessoasFisicas (pessoaID, cpf) " +
        "VALUES (?, ?)";

    int id = SequenceManager.getValue("Seq_PessoaID");
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
        conexao = conector.getConnection();
        declaracaoPessoa = conector.getPreparedStatement(conexao, sqlPessoa);
        declaracaoPessoaFisica = conector.getPreparedStatement(conexao, sqlPessoaFisica);

        // INSERIR NA TABELA DE PESSOA
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.setString(2, pessoaFisica.getNome());
        declaracaoPessoa.setString(3, pessoaFisica.getLogradouro());
        declaracaoPessoa.setString(4, pessoaFisica.getCidade());
        declaracaoPessoa.setString(5, pessoaFisica.getEstado());
        declaracaoPessoa.setString(6, pessoaFisica.getTelefone());
        declaracaoPessoa.setString(7, pessoaFisica.getEmail());
        declaracaoPessoa.executeUpdate();

        // INSERIR NA TABELA DE PESSOA FISICA
        declaracaoPessoaFisica.setInt(1, id);
        declaracaoPessoaFisica.setString(2, pessoaFisica.getCpf());
        declaracaoPessoaFisica.executeUpdate();

        System.out.println("\nPessoa física adicionada com sucesso!");
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } finally {
        conector.close(conexao);
        conector.close(declaracaoPessoaFisica);
        conector.close(declaracaoPessoa);
    }
}

public void alterar(PessoaFisica pessoaFisica) {
    String sqlPessoa =
        "UPDATE Pessoas SET nome = ?, logradouro = ?, cidade = ?, estado = ?, telefone = ?, email = ? " +
        "WHERE pessoaID = ?";
    String sqlPessoaFisica = "UPDATE PessoasFisicas SET cpf = ? WHERE pessoaID = ?";

    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
        conexao = conector.getConnection();
        declaracaoPessoa = conector.getPreparedStatement(conexao, sqlPessoa);
        declaracaoPessoaFisica = conector.getPreparedStatement(conexao, sqlPessoaFisica);

        // ATUALIZAR A TABELA DE PESSOA
        declaracaoPessoa.setString(1, pessoaFisica.getNome());
        declaracaoPessoa.setString(2, pessoaFisica.getLogradouro());
        declaracaoPessoa.setString(3, pessoaFisica.getCidade());
        declaracaoPessoa.setString(4, pessoaFisica.getEstado());
        declaracaoPessoa.setString(5, pessoaFisica.getTelefone());
        declaracaoPessoa.setString(6, pessoaFisica.getEmail());
        declaracaoPessoa.setInt(7, pessoaFisica.getId());
        declaracaoPessoa.executeUpdate();

        // ATUALIZAR A TABELA DE PESSOA FISICA
        declaracaoPessoaFisica.setString(1, pessoaFisica.getCpf());
        declaracaoPessoaFisica.setInt(2, pessoaFisica.getId());
        declaracaoPessoaFisica.executeUpdate();

        System.out.println("\nPessoa física alterada com sucesso!");

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } finally {
        conector.close(conexao);
        conector.close(declaracaoPessoaFisica);
        conector.close(declaracaoPessoa);
    }
}

```

```

public void excluir(PessoaFisica pessoaFisica) {
    String sqlPessoaFisica = "DELETE FROM PessoasFisicas WHERE pessoaID = ?";
    String sqlPessoa = "DELETE FROM Pessoas WHERE pessoaID = ?";

    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
        conexao = conector.getConnection();
        declaracaoPessoaFisica = conector.getPreparedStatement(conexao, sqlPessoaFisica);
        declaracaoPessoa = conector.getPreparedStatement(conexao, sqlPessoa);

        // EXCLUIR UMA PESSOA FÍSICA DA TABELA
        declaracaoPessoaFisica.setInt(1, pessoaFisica.getId());
        declaracaoPessoaFisica.executeUpdate();

        // EXCLUIR UMA PESSOA DA TABELA
        declaracaoPessoa.setInt(1, pessoaFisica.getId());
        declaracaoPessoa.executeUpdate();

        System.out.println("\nPessoa física excluída com sucesso!");

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } finally {
        conector.close(conexao);
        conector.close(declaracaoPessoaFisica);
        conector.close(declaracaoPessoa);
    }
}
}

```

- b. Classe PessoaJuridicaDAO, com os métodos getPessoa, retornando uma pessoa jurídica a partir do seu id, getPessoas, para retorno de todas as pessoas jurídicas do banco de dados, incluir, para inclusão de uma pessoa jurídica, fornecida como parâmetro, nas tabelas Pessoa e PessoaJuridica, alterar, para alteração dos dados de uma pessoa jurídica, e excluir, para remoção da pessoa do banco em ambas as tabelas.

Código na próxima página...

```

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 {

    private static final ConectorBD conector = new ConectorBD();

    public PessoaJuridica getPessoa(int id) {
        String sqlGetPessoa =
            "SELECT * FROM Pessoas p JOIN PessoasJuridicas pf ON p.pessoaID = pf.pessoaID WHERE p.pessoaID = ?";

        PessoaJuridica pessoaJuridica = null;
        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet resultado = null;

        try {
            conexao = conector.getConnection();
            declaracao = conector.getPreparedStatement(conexao, sqlGetPessoa);
            declaracao.setInt(1, id);
            resultado = conector.getSelect(declaracao);
            if (resultado.next()) {
                pessoaJuridica = new PessoaJuridica(
                    resultado.getString("cnpj"),
                    resultado.getInt("pessoaID"),
                    resultado.getString("nome"),
                    resultado.getString("logradouro"),
                    resultado.getString("cidade"),
                    resultado.getString("estado"),
                    resultado.getString("telefone"),
                    resultado.getString("email")
                );
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } finally {
            conector.close(conexao, declaracao, resultado);
        }

        return pessoaJuridica;
    }

    public List<PessoaJuridica> getPessoas() {
        String sql = "SELECT * FROM Pessoas p JOIN PessoasJuridicas pf ON p.pessoaID = pf.pessoaID";
        List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();
        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet resultado = null;

        try {
            conexao = conector.getConnection();
            declaracao = conector.getPreparedStatement(conexao, sql);
            resultado = conector.getSelect(declaracao);
            while (resultado.next()) {
                pessoasJuridicas.add(new PessoaJuridica(
                    resultado.getString("cnpj"),
                    resultado.getInt("pessoaID"),
                    resultado.getString("nome"),
                    resultado.getString("logradouro"),
                    resultado.getString("cidade"),
                    resultado.getString("estado"),
                    resultado.getString("telefone"),
                    resultado.getString("email")
                ));
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } finally {
            conector.close(conexao, declaracao, resultado);
        }

        return pessoasJuridicas;
    }
}

```

```

public void incluir(PessoaJuridica pessoaJuridica) {
    String sqlPessoa = "INSERT INTO Pessoas (pessoaID, nome, logradouro, cidade, estado, telefone, email) " +
        "VALUES (?, ?, ?, ?, ?, ?, ?)";
    String sqlPessoaJuridica = "INSERT INTO PessoasJuridicas (pessoaID, cnpj) " +
        "VALUES (?, ?)";

    int id = SequenceManager.getValue("Seq_PessoaID");
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica = null;

    try {
        conexao = conector.getConnection();
        declaracaoPessoa = conector.getPreparedStatement(conexao, sqlPessoa);
        declaracaoPessoaJuridica = conector.getPreparedStatement(conexao, sqlPessoaJuridica);

        // INSERIR NA TABELA DE PESSOA
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.setString(2, pessoaJuridica.getNome());
        declaracaoPessoa.setString(3, pessoaJuridica.getLogradouro());
        declaracaoPessoa.setString(4, pessoaJuridica.getCidade());
        declaracaoPessoa.setString(5, pessoaJuridica.getEstado());
        declaracaoPessoa.setString(6, pessoaJuridica.getTelefone());
        declaracaoPessoa.setString(7, pessoaJuridica.getEmail());
        declaracaoPessoa.executeUpdate();

        // INSERIR NA TABELA DE PESSOA FISICA
        declaracaoPessoaJuridica.setInt(1, id);
        declaracaoPessoaJuridica.setString(2, pessoaJuridica.getCnpj());
        declaracaoPessoaJuridica.executeUpdate();

        System.out.println("\nPessoa jurídica adicionada com sucesso!");

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } finally {
        conector.close(conexao);
        conector.close(declaracaoPessoaJuridica);
        conector.close(declaracaoPessoa);
    }
}

public void alterar(PessoaJuridica pessoaJuridica) {
    String sqlPessoa =
        "UPDATE Pessoas SET nome = ?, logradouro = ?, cidade = ?, estado = ?, telefone = ?, email = ? " +
        "WHERE pessoaID = ?";
    String sqlPessoaJuridica = "UPDATE PessoasJuridicas SET cnpj = ? WHERE pessoaID = ?";

    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica = null;

    try {
        conexao = conector.getConnection();
        declaracaoPessoa = conector.getPreparedStatement(conexao, sqlPessoa);
        declaracaoPessoaJuridica = conector.getPreparedStatement(conexao, sqlPessoaJuridica);

        // ATUALIZAR A TABELA DE PESSOA
        declaracaoPessoa.setString(1, pessoaJuridica.getNome());
        declaracaoPessoa.setString(2, pessoaJuridica.getLogradouro());
        declaracaoPessoa.setString(3, pessoaJuridica.getCidade());
        declaracaoPessoa.setString(4, pessoaJuridica.getEstado());
        declaracaoPessoa.setString(5, pessoaJuridica.getTelefone());
        declaracaoPessoa.setString(6, pessoaJuridica.getEmail());
        declaracaoPessoa.setInt(7, pessoaJuridica.getId());
        declaracaoPessoa.executeUpdate();

        // ATUALIZAR A TABELA DE PESSOA FISICA
        declaracaoPessoaJuridica.setString(1, pessoaJuridica.getCnpj());
        declaracaoPessoaJuridica.setInt(2, pessoaJuridica.getId());
        declaracaoPessoaJuridica.executeUpdate();

        System.out.println("\nPessoa jurídica alterada com sucesso!");

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } finally {
        conector.close(conexao);
        conector.close(declaracaoPessoaJuridica);
        conector.close(declaracaoPessoa);
    }
}

```

```

public void excluir(PessoaJuridica pessoaJuridica) {
    String sqlPessoaJuridica = "DELETE FROM PessoasJuridicas WHERE pessoaID = ?";
    String sqlPessoa = "DELETE FROM Pessoas WHERE pessoaID = ?";

    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica = null;

    try {
        conexao = conector.getConnection();
        declaracaoPessoaJuridica = conector.getPrepared(conexao, sqlPessoaJuridica);
        declaracaoPessoa = conector.getPrepared(conexao, sqlPessoa);

        // EXCLUIR UMA PESSOA FISICA DA TABELA
        declaracaoPessoaJuridica.setInt(1, pessoaJuridica.getId());
        declaracaoPessoaJuridica.executeUpdate();

        // EXCLUIR UMA PESSOA DA TABELA
        declaracaoPessoa.setInt(1, pessoaJuridica.getId());
        declaracaoPessoa.executeUpdate();

        System.out.println("\nPessoa juridica excluida com sucesso!");

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } finally {
        conector.close(conexao);
        conector.close(declaracaoPessoaJuridica);
        conector.close(declaracaoPessoa);
    }
}
}

```

c. Utilizar nas classes objetos dos tipos ConectorBD e SequenceManager.

Ambas as classes estão utilizando objetos do tipo ConectorBD e SequenceManager, como demonstra nas instruções anteriores.

6. Criar uma classe principal de testes com o nome CadastroBDTeste, efetuando as operações seguintes no método main:
  - a. Instanciar uma pessoa física e persistir no banco de dados.

Código na próxima página...

```
public class CadastroBDTeste {  
  
    public static void main(String[] args) {  
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();  
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();  
  
        // a. Instanciar uma pessoa física e persistir no banco de dados  
        PessoaFisica pessoaFisica = new PessoaFisica("23133322255", 0, "Marcelo",  
            "Rua Fernandes, 8, Malaio", "São João", "PE",  
            "88999999998", "emailtest@gmail.com");  
        pessoaFisicaDAO.incluir(pessoaFisica);
```

b. Alterar os dados da pessoa física no banco.

```
// b. Alterar os dados da pessoa física no banco.  
PessoaFisica pessoaFisica2 = new PessoaFisica("23133322255", 47, "Fernanda",  
    "Rua Fernandes, 8, Malaio", "São João", "PE",  
    "88999999998", "emailtest@gmail.com");  
pessoaFisicaDAO.alterar(pessoaFisica2);
```

c. Consultar todas as pessoas físicas do banco de dados e listar no console.

```
// c. Consultar todas as pessoas físicas do banco de dados e listar no console.  
pessoaFisicaDAO.getPessoas().forEach(pessoa -> System.out.println(pessoa.exibir()));
```

Resultado do código:

```
run:  
ID: 7  
Nome: João Marcelo  
Logradouro: Rua 12, casa 3, Quitanda  
Cidade: Riacho do Norte  
Estado: PA  
Telefone: 1111-1111  
Email: joao@riacho.com  
CPF: 11111111111  
  
ID: 47  
Nome: Fernanda  
Logradouro: Rua Fernandes, 8, Malaio  
Cidade: São João  
Estado: PE  
Telefone: 88999999998  
Email: emailtest@gmail.com  
CPF: 23133322255  
  
BUILD SUCCESSFUL (total time: 0 seconds)
```

d. Excluir a pessoa física criada anteriormente no banco.

```
// d. Excluir a pessoa física criada anteriormente no banco.  
pessoaFisicaDAO.excluir(pessoaFisica);
```

e. Instanciar uma pessoa jurídica e persistir no banco de dados.

```
// e. Instanciar uma pessoa jurídica e persistir no banco de dados.  
PessoaJuridica pessoaJuridica = new PessoaJuridica("44444444444444", 0, "Golden Value",  
    "Avenida Lopes, 33, Apto 359", "São Paulo", "SP",  
    "11977777765", " contato@goldenvalue.com.br");  
pessoaJuridicaDAO.incluir(pessoaJuridica);
```

f. Alterar os dados da pessoa jurídica no banco.

```
// f. Alterar os dados da pessoa jurídica no banco.  
PessoaJuridica pessoaJuridica2 = new PessoaJuridica("444444444444", 55, "Golden Value",  
    "Avenida Lopes, 33, Apto 359", "São Paulo", "SP",  
    "11666666666", " contato@goldenvalue.com.br");  
pessoaJuridicaDAO.alterar(pessoaJuridica2);
```

g. Consultar todas as pessoas jurídicas do banco e listar no console.

```
// g. Consultar todas as pessoas jurídicas do banco e listar no console.  
pessoaJuridicaDAO.getPessoas().forEach(pessoa -> System.out.println(pessoa.exibir()));
```

Resultado do código:

```
run:  
ID: 15  
Nome: JJC  
Logradouro: Rua 11, Centro  
Cidade: Riacho do Norte  
Estado: PA  
Telefone: 1212-1212  
Email: jjc@riacho.com  
CNPJ: 22222222222222  
  
BUILD SUCCESSFUL (total time: 0 seconds)
```

h. Excluir a pessoa jurídica criada anteriormente no banco.

```
// h. Excluir a pessoa jurídica criada anteriormente no banco.  
pessoaJuridicaDAO.excluir(pessoaJuridica2);
```

### **3.3. RESULTADO DOS CÓDIGOS**

A maioria dos códigos apresentados não possuem resultados visualizáveis dos códigos, entretanto, alguns códigos possuem resultados visualizáveis e estão listados em suas respectivas instruções.

### **3.4. ANÁLISE E CONCLUSÃO**

#### **3.4.1. a) QUAL A IMPORTÂNCIA DOS COMPONENTES DE MIDDLEWARE, COMO O JDBC?**

A importância dos componentes de middleware, como o JDBC, está em sua capacidade de facilitar a comunicação entre aplicações Java e bancos de dados relacionais. O JDBC atua como uma camada intermediária que abstrai as complexidades inerentes à interação com diferentes sistemas de gerenciamento de banco de dados, proporcionando uma interface padronizada para executar operações como consultas, inserções, atualizações e exclusões de dados. Essa padronização permite que desenvolvedores escrevam código independente do banco de dados específico, promovendo a portabilidade da aplicação. Além disso, o JDBC oferece mecanismos para gerenciar conexões, transações e resultados de consultas, garantindo segurança e eficiência. Ao simplificar o acesso a dados, o JDBC reduz a carga de trabalho do desenvolvedor e permite que ele se concentre na lógica de negócios, em vez de detalhes de baixo nível relacionados ao banco de dados.

#### **3.4.2. b) QUAL A DIFERENÇA NO USO DE STATEMENT OU PREPAREDSTATEMENT PARA A MANIPULAÇÃO DE DADOS?**

A diferença entre o uso de Statement e PreparedStatement na manipulação de dados está principalmente na forma como as consultas SQL são tratadas. O Statement é uma interface simples que permite a execução de consultas SQL estáticas, onde os valores são concatenados diretamente na string da consulta. Essa abordagem pode levar a vulnerabilidades de segurança, como injeção de SQL, e a um desempenho inferior, já que o banco de dados precisa recompilar a consulta a cada execução. Por outro lado, o PreparedStatement oferece uma solução mais robusta e segura, permitindo a execução de consultas parametrizadas. Com o PreparedStatement, a

consulta é pré-compilada no banco de dados, o que melhora o desempenho em execuções repetidas. Além disso, os parâmetros são tratados de forma segura, evitando ataques de injeção de SQL. Essa abordagem também torna o código mais legível e organizado, especialmente em consultas complexas com múltiplos parâmetros.

### **3.4.3. c) COMO O PADRÃO DAO MELHORA A MANUTENIBILIDADE DO SOFTWARE?**

O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao promover a separação de responsabilidades entre a lógica de acesso a dados e a lógica de negócios. Ao isolar as operações de banco de dados em classes específicas, o DAO permite que alterações na estrutura do banco de dados ou na forma como os dados são acessados sejam realizadas sem impactar o restante do sistema. Isso facilita a manutenção e a evolução do código, já que as mudanças são localizadas e não afetam outras partes da aplicação. Além disso, o DAO promove a reutilização de código, uma vez que operações comuns de CRUD (Create, Read, Update, Delete) são centralizadas em métodos específicos. Essa organização também melhora a testabilidade, pois as operações de acesso a dados podem ser testadas de forma isolada, utilizando mocks ou bancos de dados em memória. Em resumo, o padrão DAO contribui para um código mais modular, organizado e fácil de manter.

### **3.4.4. d) COMO A HERANÇA É REFLETIDA NO BANCO DE DADOS, QUANDO LIDAMOS COM UM MODELO ESTRITAMENTE RELACIONAL?**

A herança, um conceito fundamental da orientação a objetos, não é diretamente suportada em bancos de dados relacionais, que seguem um modelo estritamente tabular. No entanto, é possível refletir a herança em bancos de dados relacionais utilizando estratégias como tabela única, tabela por classe ou tabela por subclasse. Na abordagem de tabela única, todas as classes da hierarquia são mapeadas para uma única tabela, com colunas adicionais para armazenar atributos específicos de cada subclasse e uma coluna discriminadora para indicar o tipo de registro. Essa abordagem é simples e eficiente em termos de consultas, mas pode resultar em tabelas esparsas, com muitas colunas nulas. Na abordagem de tabela por classe, cada classe é mapeada para uma tabela separada, com a tabela da subclasse contendo uma chave estrangeira para a tabela da superclasse. Essa estratégia evita colunas nulas, mas pode tornar as consultas mais complexas. Por fim, na abordagem de tabela por subclasse, cada classe é mapeada para uma tabela separada, com a tabela da subclasse contendo apenas os atributos específicos e uma chave primária que também é uma chave estrangeira para a tabela da superclasse. Essa abordagem

promove a normalização, mas pode impactar o desempenho devido à necessidade de joins em consultas. Cada estratégia tem suas vantagens e desvantagens, e a escolha depende das necessidades específicas da aplicação e do banco de dados.

## **4. 2º PROCEDIMENTO | ALIMENTANDO A BASE**

### **4.1. CÓDIGOS SOLICITADOS**

Os códigos serão demonstrados neste relatório de acordo com as instruções apresentadas no enunciado da missão prática. Somente serão demonstrados nesse relatório as principais instruções que são relevantes ao desenvolvimento do código.

### **4.2. PRINCIPAIS INSTRUÇÕES DO 2º PROCEDIMENTO**

1. Alterar o método main da classe principal do projeto, para implementação do cadastro em modo texto:
  - a. Apresentar as opções do programa para o usuário, sendo 1 para incluir, 2 para alterar, 3 para excluir, 4 para exibir pelo id, 5 para exibir todos e 0 para finalizar a execução.

```

import cadastrobd.model.*;
import java.util.Scanner;

public class CadastroBD {

    static final Scanner LEITOR = new Scanner(System.in);

    static final PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
    static final PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

    public static void main(String[] args) {

        int escolhaMenu = -1;

        String opcoes = """
                        MENU PRINCIPAL
                        =====
                        1 - Incluir Pessoa
                        2 - Alterar Pessoa
                        3 - Excluir pessoa
                        4 - Buscar pelo Id
                        5 - Exibir Todos
                        0 - Finalizar Programa
                        =====""";

        do {
            boolean escolhaValida = false;
            System.out.println(opcoes);
            System.out.print("Digite o número da opção -> ");
            do {
                try {
                    escolhaMenu = Integer.parseInt(LEITOR.nextLine());
                    escolhaValida = true;
                } catch(NumberFormatException ex) {
                    System.out.println("\nEntrada inválida, tente novamente");
                }
            } while (!escolhaValida);
            switch (escolhaMenu) {
                case 1 -> incluirPessoa();

                case 2 -> alterarPessoa();

                case 3 -> excluirPessoa();

                case 4 -> buscarPeloId();

                case 5 -> exibirTodos();

                case 0 -> System.out.println("\nFinalizando o programa...");

                default -> System.out.println("\nOpção inválida! Tente novamente.");
            }
        } while (escolhaMenu != 0);
    }
}

```

Resultado do código:

```
run:

        MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> |
```

- b. Selecionada a opção incluir, escolher o tipo (Física ou Jurídica), receber os dados a partir do teclado e adicionar no banco de dados através da classe DAO correta.

```
private static void incluirPessoa() {
    switch(selecionarTipoPessoa().toUpperCase()) {
        case "F" -> incluirPessoaFisica();
        case "J" -> incluirPessoaJuridica();
    }
    digiteParaVoltarAoMenu();
}

private static void incluirPessoaFisica() {
    Pessoa pessoa = digitarPessoa();
    System.out.print("\nCPF (somente números) -> ");
    String cpf = LEITOR.nextLine();
    pessoaFisicaDAO.incluir(new PessoaFisica(pessoa, cpf));
}

private static void incluirPessoaJuridica() {
    Pessoa pessoa = digitarPessoa();
    System.out.print("\nCNPJ (Somente números) -> ");
    String cnpj = LEITOR.nextLine();
    pessoaJuridicaDAO.incluir(new PessoaJuridica(pessoa, cnpj));
}
```

```

private static String selecionarTipoPessoa() {
    String escolhaMenu = "";
    do {
        System.out.println("\nF - Pessoa física | J - Pessoa Jurídica");
        System.out.print("Digite a letra da opção desejada -> ");
        escolhaMenu = LEITOR.nextLine();
    } while(!(escolhaMenu.equalsIgnoreCase("F") || escolhaMenu.equalsIgnoreCase("J")));

    return escolhaMenu;
}

private static Pessoa digitarPessoa() {
    int id = 0;
    System.out.print("\nDigite o nome -> ");
    String nome = LEITOR.nextLine();
    System.out.print("\nDigite o logradouro -> ");
    String logradouro = LEITOR.nextLine();
    System.out.print("\nDigite a cidade -> ");
    String cidade = LEITOR.nextLine();
    System.out.print("\nDigite o estado -> ");
    String estado = LEITOR.nextLine();
    System.out.print("\nDigite o telefone -> ");
    String telefone = LEITOR.nextLine();
    System.out.print("\nDigite o email -> ");
    String email = LEITOR.nextLine();

    return new Pessoa(id, nome, logradouro, cidade, estado, telefone, email);
}

```

### Resultado do código:

```

run:

        MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> 1

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada -> j

Digite o nome -> Ferrari

Digite o logradouro -> Avenida Beiramar, 25, Lopes

Digite a cidade -> Manaus

Digite o estado -> AM

Digite o telefone -> 89988332211

Digite o email -> support@ferrari.com

CNPJ (Somente números) -> 99008765432198

Pessoa jurídica adicionada com sucesso!

Digite 1 para voltar ao Menu Principal -> 1

```

- c. Selecionada a opção alterar, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado, apresentar os dados atuais, solicitar os novos dados e alterar no banco de dados através do DAO.

```

private static void alterarPessoa() {
    switch(selecionarTipoPessoa().toUpperCase()) {
        case "F" -> alterarPessoaFisica();
        case "J" -> alterarPessoaJuridica();
    }
    digiteParaVoltarAoMenu();
}

private static void alterarPessoaFisica() {
    String escolhaMenu = "";
    PessoaFisica pessoaFisica = buscarPessoaFisicaPeloId();

    if (pessoaFisica != null) {
        do {
            System.out.println("Deseja alterar essa pessoa física? (S) Sim | (N) Não ");
            System.out.print("\nDigite a letra da opção desejada -> ");
            escolhaMenu = LEITOR.nextLine();
        } while(!(escolhaMenu.equalsIgnoreCase("S") || escolhaMenu.equalsIgnoreCase("N")));

        switch (escolhaMenu.toUpperCase()) {
            case "S" -> {
                System.out.println("\nInsira os novos dados...");
                Pessoa pessoa = digitarPessoa();
                System.out.print("\nCPF (somente números) -> ");
                String cpf = LEITOR.nextLine();
                pessoa.setId(pessoaFisica.getId());
                PessoaFisica pessoaFisicaAtualizada = new PessoaFisica(pessoa, cpf);
                pessoaFisicaDAO.alterar(pessoaFisicaAtualizada);
            }
            case "N" -> System.out.println("\nPessoa física não alterada. Voltando para o menu...");
        }
    }
}

private static void alterarPessoaJuridica() {
    String escolhaMenu = "";
    PessoaJuridica pessoaJuridica = buscarPessoaJuridicaPeloId();

    if (pessoaJuridica != null) {
        do {
            System.out.println("Deseja alterar essa pessoa jurídica? (S) Sim | (N) Não ");
            System.out.print("Digite a letra da opção desejada -> ");
            escolhaMenu = LEITOR.nextLine();
        } while(!(escolhaMenu.equalsIgnoreCase("S") || escolhaMenu.equalsIgnoreCase("N")));

        switch (escolhaMenu.toUpperCase()) {
            case "S" -> {
                System.out.println("\nInsira os novos dados...");
                Pessoa pessoa = digitarPessoa();
                System.out.print("\nCNPJ (somente números) -> ");
                String cnpj = LEITOR.nextLine();
                pessoa.setId(pessoaJuridica.getId());
                PessoaJuridica pessoaJuridicaAtualizada = new PessoaJuridica(pessoa, cnpj);
                pessoaJuridicaDAO.alterar(pessoaJuridicaAtualizada);
            }
            case "N" -> System.out.println("\nPessoa física não alterada. Voltando para o menu...");
        }
    }
}

```

Resultado do código:

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> 2

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada -> j

Digite o ID da pessoa jurídica -> 71

**** Pessoa jurídica ****
ID: 71
Nome: Ferrari
Logradouro: Avenida Beiramar, 25, Lopes
Cidade: Manaus
Estado: AM
Telefone: 89988332211
Email: support@ferrari.com
CNPJ: 99008765432198

Deseja alterar essa pessoa jurídica? (S) Sim | (N) Não
Digite a letra da opção desejada -> s

Insira os novos dados...

Digite o nome -> LS Plásticos

Digite o logradouro -> Avenida Beiramar, 25, Lopes

Digite a cidade -> Manaus

Digite o estado -> AM

Digite o telefone -> 899999999993

Digite o email -> contato@lsplasticos.com.br

CNPJ (somente números) -> 00091113897332

Pessoa jurídica alterada com sucesso!
```

- d. Selecionada a opção excluir, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado e remover do banco de dados através do DAO.

```

private static void excluirPessoa() {
    switch(selecionarTipoPessoa().toUpperCase()) {
        case "F" -> excluirPessoaFisica();

        case "J" -> excluirPessoaJuridica();
    }
    digiteParaVoltarAoMenu();
}

private static void excluirPessoaFisica() {
    String escolhaMenu = "";
    PessoaFisica pessoaFisica = buscarPessoaFisicaPeloId();
    if (pessoaFisica != null) {
        do {
            System.out.println("Deseja excluir " + pessoaFisica.getNome() + " do registro? (S) Sim | (N) Não ");
            System.out.print("Digite a letra da opção desejada -> ");
            escolhaMenu = LEITOR.nextLine();
        } while(!(escolhaMenu.equalsIgnoreCase("S") || escolhaMenu.equalsIgnoreCase("N")));

        switch (escolhaMenu.toUpperCase()) {
            case "S" -> pessoaFisicaDAO.excluir(pessoaFisica);

            case "N" -> System.out.println("\nPessoa física não excluída. Voltando para o menu...");
        }
    } else {
        System.out.println("\nNão existe pessoa física cadastrada com esse ID");
    }
}

private static void excluirPessoaJuridica() {
    String escolhaMenu = "";
    PessoaJuridica pessoaJuridica = buscarPessoaJuridicaPeloId();
    if (pessoaJuridica != null) {
        do {
            System.out.println("Deseja excluir " + pessoaJuridica.getNome() + " do registro? (S) Sim | (N) Não ");
            System.out.print("Digite a letra da opção desejada -> ");
            escolhaMenu = LEITOR.nextLine();
        } while(!(escolhaMenu.equalsIgnoreCase("S") || escolhaMenu.equalsIgnoreCase("N")));

        switch (escolhaMenu.toUpperCase()) {
            case "S" -> pessoaJuridicaDAO.excluir(pessoaJuridica);

            case "N" -> System.out.println("\nPessoa jurídica não excluída. Voltando para o menu...");
        }
    } else {
        System.out.println("\nNão existe pessoa jurídica cadastrada com esse ID");
    }
}

private static PessoaFisica buscarPessoaFisicaPeloId() {
    Integer idPessoa = digitarIdPessoa("física");
    PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoa);
    if (pessoaFisica != null) {
        System.out.println("\n**** Pessoa física ****\n" + pessoaFisica.exibir());
        return pessoaFisica;
    }
    System.out.println("\nNão existe pessoa física cadastrada com esse ID");
    return null;
}

private static PessoaJuridica buscarPessoaJuridicaPeloId() {
    Integer idPessoa = digitarIdPessoa("jurídica");
    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoa);
    if (pessoaJuridica != null) {
        System.out.println("\n**** Pessoa jurídica ****\n" + pessoaJuridica.exibir());
        return pessoaJuridica;
    }
    System.out.println("\nNão existe pessoa jurídica cadastrada com esse ID");
    return null;
}

```

Resultado do código:

```
run:

        MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> 3

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada -> f

Digite o ID da pessoa fisica -> 47

**** Pessoa fisica ****
ID: 47
Nome: Fernanda
Logradouro: Rua Fernandes, 8, Malaio
Cidade: São João
Estado: PE
Telefone: 88999999998
Email: emailtest@gmail.com
CPF: 23133322255

Deseja excluir Fernanda do registro? (S) Sim | (N) Não
Digite a letra da opção desejada -> s

Pessoa fisica excluída com sucesso!

Digite 1 para voltar ao Menu Principal -> |
```

- e. Selecionada a opção obter, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado e apresentar os dados atuais, recuperados do banco através do DAO.

```

private static void buscarPeloId() {
    switch(selecionarTipoPessoa().toUpperCase()) {
        case "F" -> buscarPessoaFisicaPeloId();
        case "J" -> buscarPessoaJuridicaPeloId();
    }
    digiteParaVoltarAoMenu();
}

private static PessoaFisica buscarPessoaFisicaPeloId() {
    Integer idPessoa = digitarIdPessoa("física");
    PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoa);
    if (pessoaFisica != null) {
        System.out.println("\n**** Pessoa física ****\n" + pessoaFisica.exibir());
        return pessoaFisica;
    }
    System.out.println("\nNão existe pessoa física cadastrada com esse ID");
    return null;
}

private static PessoaJuridica buscarPessoaJuridicaPeloId() {
    Integer idPessoa = digitarIdPessoa("jurídica");
    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoa);
    if (pessoaJuridica != null) {
        System.out.println("\n**** Pessoa jurídica ****\n" + pessoaJuridica.exibir());
        return pessoaJuridica;
    }
    System.out.println("\nNão existe pessoa física cadastrada com esse ID");
    return null;
}

```

### Resultado do código:

```

run:

        MENU PRINCIPAL
-----
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
-----
Digite o número da opção -> 4

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada -> j

Digite o ID da pessoa jurídica -> 71

**** Pessoa jurídica ****
ID: 71
Nome: LS Plásticos
Logradouro: Avenida Beiramar, 25, Lopes
Cidade: Manaus
Estado: AM
Telefone: 899999999993
Email: contato@lsp plasticos.com.br
CNPJ: 00091113897332

Digite 1 para voltar ao Menu Principal ->

```

- f. Selecionada a opção obterTodos, escolher o tipo (Física ou Jurídica) e apresentar os dados de todas as entidades presentes no banco de dados por intermédio do DAO.

```

private static void exibirTodos() {
    switch(selecionarTipoPessoa().toUpperCase()) {
        case "F" -> {
            System.out.println("\n**** Todas as pessoas físicas ****\n");
            pessoaFisicaDAO.getPessoas().forEach(pessoa -> System.out.println(pessoa.exibir()));
        }

        case "J" -> {
            System.out.println("\n**** Todas as pessoas jurídicas ****\n");
            pessoaJuridicaDAO.getPessoas().forEach(pessoa -> System.out.println(pessoa.exibir()));
        }
    }
    digiteParaVoltarAoMenu();
}

```

Resultado do código:

```

run:

                MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> 5

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada -> j

**** Todas as pessoas jurídicas ****

ID: 71
Nome: LS Plásticos
Logradouro: Avenida Beiramar, 25, Lopes
Cidade: Manaus
Estado: AM
Telefone: 89999999999
Email: contato@lsplasticos.com.br
CNPJ: 00091113897332

ID: 15
Nome: JJC
Logradouro: Rua 11, Centro
Cidade: Riacho do Norte
Estado: PA
Telefone: 1212-1212
Email: jjc@riacho.com
CNPJ: 22222222222222

Digite 1 para voltar ao Menu Principal ->

```

- g. Qualquer exceção que possa ocorrer durante a execução do sistema deverá ser tratada.

Todas exceções que possam ocorrer serão devidamente tratadas.

Alguns resultados de exceções implementadas:

```
run:

        MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> h

Entrada inválida, tente novamente

Digite o número da opção ->
```

```
run:

        MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> h

Entrada inválida, tente novamente

Digite o número da opção -> 5

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada -> 8

F - Pessoa física | J - Pessoa Jurídica
Digite a letra da opção desejada ->
```

h. Selecionada a opção sair, finalizar a execução do sistema.

Assim que o usuário selecionar a opção sair, através da opção de menu número “0”, o loop se encerra, e junto a execução do programa.

```
do {
    boolean escolhaValida = false;
    System.out.println(opcoes);
    System.out.print("Digite o número da opção -> ");
    do {
        try {
            escolhaMenu = Integer.parseInt(LEITOR.nextLine());
            escolhaValida = true;
        } catch(NumberFormatException ex) {
            System.out.println("\nEntrada inválida, tente novamente");
        }
    } while (!escolhaValida);

    switch (escolhaMenu) {
        case 1 -> incluirPessoa();

        case 2 -> alterarPessoa();

        case 3 -> excluirPessoa();

        case 4 -> buscarPeloId();

        case 5 -> exibirTodos();

        case 0 -> System.out.println("\nFinalizando o programa...");

        default -> System.out.println("\nOpção inválida! Tente novamente.");
    }
} while (escolhaMenu != 0);
```

Resultado do código:

```
run:

                MENU PRINCIPAL
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
Digite o número da opção -> 0

Finalizando o programa...
BUILD SUCCESSFUL (total time: 4 seconds)
```

### **4.3. RESULTADO DOS CÓDIGOS**

A maioria dos códigos apresentados não possuem resultados visualizáveis dos códigos, entretanto, alguns códigos possuem resultados visualizáveis e estão listados em suas respectivas instruções.

### **4.4. ANÁLISE E CONCLUSÃO**

#### **4.4.1. a) QUAIS AS DIFERENÇAS ENTRE A PERSISTÊNCIA EM ARQUIVO E A PERSISTÊNCIA EM BANCO DE DADOS?**

A persistência em arquivo e a persistência em banco de dados são duas abordagens distintas para armazenar e gerenciar dados, cada uma com suas características e aplicações específicas. A persistência em arquivo envolve o armazenamento de dados em arquivos no sistema de arquivos do computador, como arquivos de texto, binários ou formatos estruturados como JSON ou XML. Essa abordagem é simples e direta, sendo adequada para cenários onde a quantidade de dados é pequena e a complexidade das operações é baixa. No entanto, a persistência em arquivo pode se tornar ineficiente em aplicações que exigem consultas complexas, gerenciamento de transações ou acesso concorrente aos dados, já que o sistema de arquivos não foi projetado para lidar com essas demandas de forma otimizada. Além disso, a falta de mecanismos de segurança e integridade de dados pode tornar essa abordagem vulnerável a corrupção de arquivos ou acessos não autorizados.

Por outro lado, a persistência em banco de dados oferece uma solução mais robusta e escalável para o armazenamento de dados. Bancos de dados relacionais, como MySQL, PostgreSQL ou SQL Server, são projetados para gerenciar grandes volumes de dados de forma eficiente, com suporte a consultas complexas, transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e controle de acesso. Essa abordagem permite a normalização dos dados, evitando redundâncias e garantindo a integridade referencial. Além disso, bancos de dados, com auxílio de outras tecnologias, como Sistemas Gerenciadores de Banco de Dados (SGBD), fornecem ferramentas para backup, recuperação de desastres e otimização de desempenho, o que os torna ideais para aplicações críticas e de grande escala. No entanto, a persistência em banco de dados requer uma infraestrutura mais complexa, incluindo servidores de banco de dados e conhecimento de linguagens SQL, o que pode aumentar o custo e a complexidade do desenvolvimento.

#### **4.4.2. b) COMO O USO DE OPERADOR LAMBDA SIMPLIFICOU A IMPRESSÃO DOS VALORES CONTIDOS NAS ENTIDADES, NAS VERSÕES MAIS RECENTES DO JAVA?**

O uso de operadores lambda nas versões mais recentes do Java trouxe uma simplificação significativa para a impressão de valores contidos nas entidades, especialmente quando se trabalha com coleções ou fluxos de dados. Antes da introdução das expressões lambda, a iteração sobre coleções e a execução de operações como impressão de valores exigiam o uso de loops explícitos ou classes anônimas, o que resultava em código verboso e de difícil leitura. Com o advento das expressões lambda, é possível escrever código mais conciso e expressivo, utilizando métodos como `forEach` em conjunto com lambdas para realizar operações em cada elemento de uma coleção. Por exemplo, em vez de escrever um loop `for` para iterar sobre uma lista e imprimir seus valores, pode-se simplesmente usar `lista.forEach(valor -> System.out.println(valor))`. Essa abordagem não apenas reduz a quantidade de código, mas também melhora a legibilidade e a manutenção, alinhando-se com os princípios de programação funcional.

#### **4.4.3. c) POR QUE MÉTODOS ACIONADOS DIRETAMENTE PELO MÉTODO MAIN, SEM O USO DE UM OBJETO, PRECISAM SER MARCADOS COMO STATIC?**

Métodos acionados diretamente pelo método `main`, sem o uso de um objeto, precisam ser marcados como `static` porque o método `main` é o ponto de entrada de uma aplicação Java e é executado em um contexto onde nenhuma instância da classe foi criada. Em Java, métodos de instância dependem de um objeto específico para serem chamados, já que podem acessar atributos e comportamentos associados a essa instância. No entanto, o método `main` é chamado pela JVM (Java Virtual Machine) antes de qualquer objeto ser instanciado, o que significa que ele não pode depender de uma instância da classe para funcionar. Ao marcar um método como `static`, ele se torna independente de instâncias, pertencendo à classe em si e podendo ser chamado diretamente sem a necessidade de um objeto. Isso garante que o método `main` e quaisquer outros métodos auxiliares chamados por ele possam ser executados corretamente no início da aplicação, antes que qualquer instância seja criada. Essa característica é fundamental para o funcionamento correto do ponto de entrada de programas Java.

## **REFERÊNCIAS**

CTC Tech, “Middleware: o que é e para que serve?”. Disponível em:  
<https://ctctech.com.br/blog/middleware/>. Acesso em 31 de janeiro de 2025.

Baeldung, “Difference Between Statement and PreparedStatement”. Disponível em:  
<https://www.baeldung.com/java-statement-preparedstatement>. Acesso em 31 de janeiro de 2025.

DevMedia, “DAO Pattern: Persistência de Dados utilizando o padrão DAO”. Disponível em: <https://www.devmedia.com.br/dao-pattern-persistencia-de-dados-utilizando-o-padrao-dao/30999>. Acesso em 31 de janeiro de 2025.

StackOverFlow, “Herança em banco de dados relacionais”. Disponível em:  
<https://pt.stackoverflow.com/questions/35054/heran%C3%A7a-em-banco-de-dados-relacionais>. Acesso em 31 de janeiro de 2025.

Alura, “Orientação a Objetos, Herança e Banco de dados”. Disponível em  
<https://cursos.alura.com.br/forum/topic/orientacao-a-objetos-heranca-e-banco-de-dados-43969>. Acesso em 31 de janeiro de 2025.

DevMedia, “Artigo SQL Magazine 6 - Métodos de persistência em Java”. Disponível em <https://www.devmedia.com.br/artigo-sql-magazine-6-metodos-de-persistencia-em-java/7075>. Acesso em 31 de janeiro de 2025.

Blip Blog, “Tudo que você precisa saber sobre persistência de dados: conceito, tipos e técnicas”. Disponível em <<https://www.blip.ai/blog/tecnologia/persistencia-de-dados/>>. Acesso em 31 de janeiro de 2025.

Treina Web, “Dica de Código: Expressões Lambda em Java”. Disponível em <<https://www.treinaweb.com.br/blog/dica-de-codigo-expressoes-lambda-em-java>>. Acesso em 31 de janeiro de 2025.

DevMedia, “Programando com Java Lambda Expressions”. Disponível em <<https://www.devmedia.com.br/programando-com-javascript-lambda-expressions/30072>>. Acesso em 31 de janeiro de 2025.

Geeks for Geeks, “Static Keyword in Java”. Disponível em <<https://www.geeksforgeeks.org/static-keyword-java/>>. Acesso em 31 de janeiro de 2025.