



Inserir aqui o Título da Prática

MAIARA ACCACIO MACHADO - 202204268183

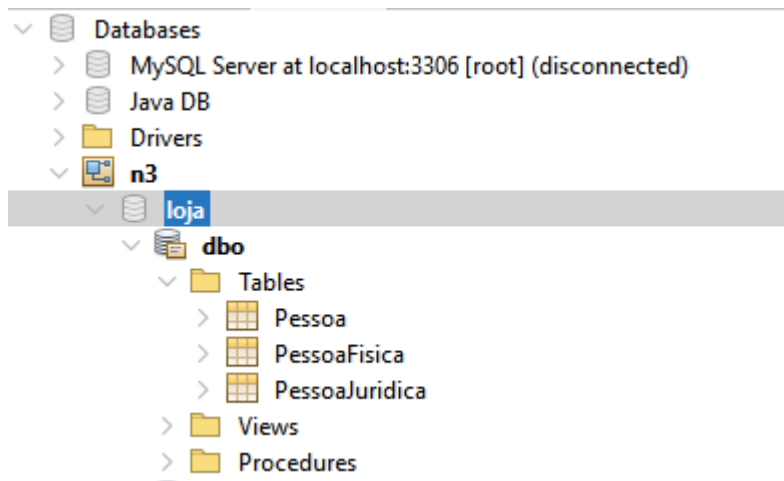
Polo Downtown – Barra da Tijuca – RJ

Nível 3: Back-end Sem Banco Não Tem – 9001 – Mundo 3

Objetivo da Prática

- Implementar persistência com base no middleware JDBC.
- Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
- Implementar o mapeamento objeto-relacional em sistemas Java.
- Criar sistemas cadastrais com persistência em banco relacional.
- Criar um aplicativo cadastral com uso do SQL Server na persistência de dados.

1º Procedimento | Mapeamento Objeto-Relacional e DAO



Script do banco de dados (Obs.: feito apenas com as tabelas necessárias nessa prática)

```
create database loja;

use loja;

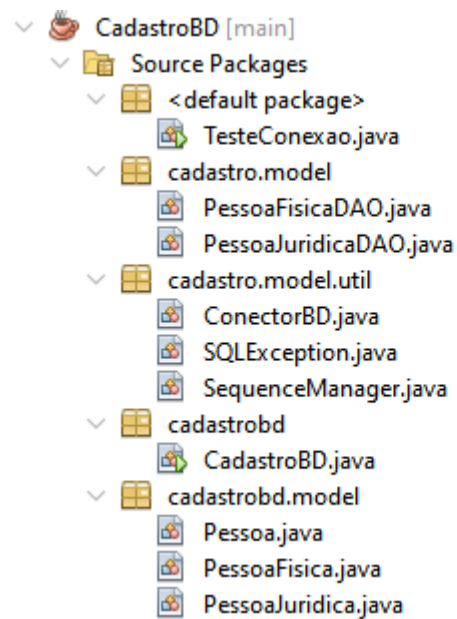
CREATE SEQUENCE PessoaSequence START WITH 1 INCREMENT BY 1;

CREATE TABLE Pessoa(
    id_Pessoa INTEGER PRIMARY KEY NOT NULL DEFAULT (NEXT VALUE FOR
PessoaSequence),
    nome VARCHAR(100) NOT NULL,
    logradouro VARCHAR(100) NOT NULL,
    cidade VARCHAR(30) NOT NULL,
    estado VARCHAR(2) NOT NULL,
    email VARCHAR(50) NOT NULL,
    telefone VARCHAR(15) NOT NULL
);

CREATE TABLE Pessoa_Juridica(
    id_Pessoa INTEGER PRIMARY KEY,
    cnpj VARCHAR(30) NOT NULL,
    FOREIGN KEY(id_Pessoa) REFERENCES Pessoa(id_Pessoa)
);

CREATE TABLE Pessoa_Fisica(
    id_Pessoa INTEGER PRIMARY KEY,
    cpf VARCHAR(15) NOT NULL,
    FOREIGN KEY(id_Pessoa) REFERENCES Pessoa(id_Pessoa)
);
```

Estrutura das pastas



CadastroBD

```
package cadastrobd;

import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridicaDAO;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaJuridica;
import java.util.Scanner;
import java.util.InputMismatchException;
import java.util.List;

/**
 *
 *
 * @author Maiara
 *
 */

public class CadastroBD {
    public static void main(String[] args) {
```

```
try (Scanner scanner = new Scanner(System.in)) {  
    PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();  
    PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();  
  
    boolean continuar = true;  
    while (continuar) {  
        exibirMenu();  
  
        String opcao = lerOpcao(scanner);  
  
        switch (opcao) {  
            case "1":  
                executarIncluir  
                    (scanner, pessoaFisicaDAO, pessoaJuridicaDAO);  
                break;  
  
            case "2":  
                executarAlterar  
                    (scanner, pessoaFisicaDAO, pessoaJuridicaDAO);  
                break;  
  
            case "3":  
                executarExcluir  
                    (scanner, pessoaFisicaDAO, pessoaJuridicaDAO);  
                break;  
  
            case "4":  
                executarObterID  
                    (scanner, pessoaFisicaDAO, pessoaJuridicaDAO);  
                break;  
  
            case "5":  
                executarListar  
                    (scanner, pessoaFisicaDAO, pessoaJuridicaDAO);  
                break;  
  
            case "0":  
                continuar = false;  
                break;
```

```

        default: System.out.println("Opção inválida. Tente novamente.");
        break;
    }
}
}

public static void exibirMenu() {
    String menu = "===== " +
        "\n1 - Incluir Pessoa" +
        "\n2 - Alterar Pessoa" +
        "\n3 - Excluir Pessoa" +
        "\n4 - Buscar pelo ID" +
        "\n5 - Exibir todos" +
        "\n0 - Finalizar programa" +
        "\n===== \n";

    System.out.println(menu);
}

public static String lerOpcao(Scanner scanner) {
    String opcao = null;
    boolean opcaoValida = false;
    while (!opcaoValida) {
        try {
            System.out.print("\nEscolha uma opcao: ");
            opcao = scanner.nextLine();
            opcaoValida = true;
        } catch (InputMismatchException e) {
            System.out.println("Opcao inválida. Insira uma opção válida do MENU.");
        }
    }
    return opcao;
}

public static String selecionarTipo(Scanner scanner) {
    String tipo = "";

```

```

boolean tipoValido = false;
while (!tipoValido) {
    System.out.print("\nF - Pessoa Fisica | J - Pessoa Juridica ");
    tipo = scanner.next();
    if (tipo.equalsIgnoreCase("F") || tipo.equalsIgnoreCase("J")) {
        tipoValido = true;
    } else {
        System.out.println("Tipo invalido. Digite F ou J.");
    }
}
return tipo;
}

public static void executarIncluir(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipoInclusao = selecionarTipo(scanner);
    if (tipoInclusao.equalsIgnoreCase("F")) {

        try{
            PessoaFisica pessoaFisica;
            pessoaFisica = incluirPessoaFisica(scanner);

            pessoaFisicaDAO.incluirPessoaFisica(pessoaFisica);
        } catch (Exception e) {
            System.out.println("Erro. Não foi possível concluir a solicitação "+e);
        }

    } else if (tipoInclusao.equalsIgnoreCase("J")) {
        incluirPessoaJuridica(scanner, pessoaJuridicaDAO);
    }
}

public static void executarAlterar(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipoAlteracao = selecionarTipo(scanner);

    if (tipoAlteracao.equalsIgnoreCase("F")) {
        alterarPessoaFisica(scanner, pessoaFisicaDAO);
    }
}

```

```

    } else if (tipoAlteracao.equalsIgnoreCase("J")) {
        alterarPessoaJuridica(scanner, pessoaJuridicaDAO);
    }
}

public static void executarExcluir(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipoExclusao = selecionarTipo(scanner);

    if (tipoExclusao.equalsIgnoreCase("F")) {
        System.out.print("Digite o ID da pessoa física para exclusão: ");
        int id_Pessoa = scanner.nextInt();
        pessoaFisicaDAO.excluirPessoaFisica(id_Pessoa);
        System.out.println("Pessoa física excluída com sucesso.");

    } else if (tipoExclusao.equalsIgnoreCase("J")) {
        System.out.print("Digite o ID da pessoa jurídica para exclusão: ");
        int id_Pessoa = scanner.nextInt();
        pessoaJuridicaDAO.excluirPessoaJuridica(id_Pessoa);
        System.out.println("Pessoa jurídica excluída com sucesso.");

    }
    scanner.nextLine();
}

public static void executarObterID(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipoObtencao = selecionarTipo(scanner);
    if (tipoObtencao.equalsIgnoreCase("F")) {
        exibirPFId(scanner, pessoaFisicaDAO);
    } else if (tipoObtencao.equalsIgnoreCase("J")) {
        exibirPJID(scanner, pessoaJuridicaDAO);
    }
}

```

```

    }
}

public static void executarListar(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO,
PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipoListagem = selecionarTipo(scanner);
    scanner.nextLine();
    if (tipoListagem.equalsIgnoreCase("F")) {
        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.listarTodasPessoasFisicas();
        System.out.println("\nLista de Pessoas Físicas");
        System.out.println("=====");

        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            pessoaFisica.exibir();
            System.out.println("=====\\n");
        }
    } else if (tipoListagem.equalsIgnoreCase("J")) {
        List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.listarTodasPessoasJuridicas();
        System.out.println("\nLista de Pessoas Jurídicas");
        System.out.println("=====");
        for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {
            pessoaJuridica.exibir();
            System.out.println("=====\\n");
        }
    }
}

public static PessoaFisica incluirPessoaFisica(Scanner scanner) {
    PessoaFisica pessoaFisica = new PessoaFisica();
    scanner.nextLine();

    System.out.print("Digite o nome da pessoa fisica: ");
    String nome = scanner.nextLine();
    pessoaFisica.setNome(nome);

    System.out.print("Digite o CPF da pessoa fisica: ");
    String cpf = scanner.nextLine();

```



```
        pessoaFisica.setCpf(cpf);

        System.out.print("Digite o Logradouro da pessoa fisica: ");
        String logradouro = scanner.nextLine();
        pessoaFisica.setLogradouro(logradouro);

        System.out.print("Digite o cidade da pessoa fisica: ");
        String cidade = scanner.nextLine();
        pessoaFisica.setCidade(cidade);

        System.out.print("Digite o estado da pessoa fisica: ");
        String estado = scanner.nextLine();
        pessoaFisica.setEstado(estado);

        System.out.print("Digite o telefone da pessoa fisica: ");
        String telefone = scanner.nextLine();
        pessoaFisica.setTelefone(telefone);

        System.out.print("Digite o email da pessoa fisica: ");
        String email = scanner.nextLine();
        pessoaFisica.setEmail(email);

        System.out.println("Pessoa fisica criada.");
        return pessoaFisica;
    }

    public static void incluirPessoaJuridica(Scanner scanner, PessoaJuridicaDAO pessoaJuridicaDAO) {
        PessoaJuridica pessoaJuridica = new PessoaJuridica();
        scanner.nextLine();

        System.out.print("Digite o nome da pessoa jurídica: ");
        String nome = scanner.nextLine();
        pessoaJuridica.setNome(nome);

        System.out.print("Digite o CNPJ da pessoa jurídica: ");
        String cnpj = scanner.nextLine();
        pessoaJuridica.setCnpj(cnpj);
```

```

System.out.print("Digite o Logradouro da pessoa jurídica: ");
String logradouro = scanner.nextLine();
pessoaJuridica.setLogradouro(logradouro);

System.out.print("Digite a cidade da pessoa jurídica: ");
String cidade = scanner.nextLine();
pessoaJuridica.setCidade(cidade);

System.out.print("Digite o estado da pessoa jurídica: ");
String estado = scanner.nextLine();
pessoaJuridica.setEstado(estado);

System.out.print("Digite o telefone da pessoa jurídica: ");
String telefone = scanner.nextLine();
pessoaJuridica.setTelefone(telefone);

System.out.print("Digite o email da pessoa jurídica: ");
String email = scanner.nextLine();
pessoaJuridica.setEmail(email);

pessoaJuridicaDAO.incluirPessoaJuridica(pessoaJuridica);
System.out.println("Pessoa jurídica incluída com sucesso.");
}

public static void alterarPessoaFisica(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO) {
    System.out.print("Digite o ID da pessoa física a ser alterada: ");
    int id = scanner.nextInt();

    PessoaFisica pessoaFisica = new PessoaFisica();
    PessoaFisica pessoaFisicaAntiga = pessoaFisicaDAO.getPessoaFisica(id);
    if (pessoaFisicaAntiga != null) {
        System.out.println("Pessoa física encontrada:");
        pessoaFisicaAntiga.exibir();

        scanner.nextLine();

        System.out.print("Digite o nome da pessoa física: ");
        String nome = scanner.nextLine();

```

```

        pessoaFisica.setNome(nome);

        System.out.print("Digite o CPF da pessoa física: ");
        String cpf = scanner.nextLine();
        pessoaFisica.setCpf(cpf);

        System.out.print("Digite o Logradouro da pessoa física: ");
        String logradouro = scanner.nextLine();
        pessoaFisica.setLogradouro(logradouro);

        System.out.print("Digite o cidade da pessoa física: ");
        String cidade = scanner.nextLine();
        pessoaFisica.setCidade(cidade);

        System.out.print("Digite o estado da pessoa física: ");
        String estado = scanner.nextLine();
        pessoaFisica.setEstado(estado);

        System.out.print("Digite o telefone da pessoa física: ");
        String telefone = scanner.nextLine();
        pessoaFisica.setTelefone(telefone);

        System.out.print("Digite o email da pessoa física: ");
        String email = scanner.nextLine();
        pessoaFisica.setEmail(email);

        pessoaFisicaDAO.alterarPessoaFisica(pessoaFisicaAntiga, pessoaFisica);
        System.out.println("Pessoa física alterada com sucesso.");
    } else {
        System.out.println("Pessoa física não encontrada com o ID informado.");
    }
}

public static void alterarPessoaJuridica(Scanner scanner, PessoaJuridicaDAO pessoaJuridicaDAO) {
    System.out.print("Digite o ID da pessoa jurídica a ser alterada: ");
    int id = scanner.nextInt();

```

```
PessoaJuridica pessoaJuridica = new PessoaJuridica();
PessoaJuridica pessoaJuridicaAntiga = pessoaJuridicaDAO.getPessoaJuridica(id);
if (pessoaJuridicaAntiga != null) {
    System.out.println("Pessoa jurídica encontrada:");
    pessoaJuridicaAntiga.exibir();

    scanner.nextLine();

    System.out.print("Digite o nome da pessoa física: ");
    String nome = scanner.nextLine();
    pessoaJuridica.setNome(nome);

    System.out.print("Digite o CNPJ da pessoa física: ");
    String cnpj = scanner.nextLine();
    pessoaJuridica.setCnpj(cnpj);

    System.out.print("Digite o Logradouro da pessoa física: ");
    String logradouro = scanner.nextLine();
    pessoaJuridica.setLogradouro(logradouro);

    System.out.print("Digite o cidade da pessoa física: ");
    String cidade = scanner.nextLine();
    pessoaJuridica.setCidade(cidade);

    System.out.print("Digite o estado da pessoa física: ");
    String estado = scanner.nextLine();
    pessoaJuridica.setEstado(estado);

    System.out.print("Digite o telefone da pessoa física: ");
    String telefone = scanner.nextLine();
    pessoaJuridica.setTelefone(telefone);

    System.out.print("Digite o email da pessoa física: ");
    String email = scanner.nextLine();
    pessoaJuridica.setEmail(email);

    pessoaJuridicaDAO.alterarPessoaJuridica(pessoaJuridicaAntiga, pessoaJuridica);
    System.out.println("Pessoa física alterada com sucesso.");
```

```

    } else {
        System.out.println("Pessoa física não encontrada com o ID informado.");
    }
}

public static void listarPessoasJuridicas(PessoaJuridicaDAO pessoaJuridicaDAO) {
    List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.listarTodasPessoasJuridicas();
    System.out.println("Dados de Pessoa Jurídica");
    for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {
        System.out.println(pessoaJuridica.toString());
    }
}

public static void listarPessoasFisicas(PessoaFisicaDAO pessoaFisicaDAO) {
    List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.listarTodasPessoasFisicas();
    if (pessoasFisicas.isEmpty()) {
        System.out.println("Não há pessoas físicas cadastradas.");
    } else {
        System.out.println("Dados de Pessoa Física");
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            System.out.println(pessoaFisica.toString());
        }
    }
}

public static void exibirPFId(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO) {
    System.out.println("Digite o ID da pessoa Física para exibição:");
    int id_Pessoa = scanner.nextInt();
    scanner.nextLine();

    PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoaFisicaById(id_Pessoa);
    if (pessoaFisica != null) {
        System.out.println("=====\\n");
        pessoaFisica.exibir();
    } else {
        System.out.println("ID não encontrado para Pessoa Física");
    }
}

```

```

    }
}

public static void exibirPJID(Scanner scanner, PessoaJuridicaDAO pessoaJuridicaDAO) {
    System.out.println("Digite o ID da pessoa Jurídica para exibição:");
    int id_Pessoa = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer do scanner

    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoaJuridicaById(id_Pessoa);
    if (pessoaJuridica != null) {
        System.out.println("=====\n");
        pessoaJuridica.exibir();
    } else {
        System.out.println("ID não encontrado para Pessoa Jurídica.");
    }
}
}
}

```

Pessoa

```

package cadastrobd.model;

/**
 *
 * @author Maiara
 */
import java.io.Serializable;

public class Pessoa implements Serializable {

    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.nome = nome;

    this.id = id;

    this.logradouro = logradouro;

    this.cidade = cidade;

    this.estado = estado;

    this.telefone = telefone;

    this.email = email;

}


public String getNome() {

    return nome;

}


public String getLogradouro() {

    return logradouro;

}


public String getCidade() {

    return cidade;
```

```
}
```

```
public String getEstado() {  
    return estado;  
}
```

```
public String getTelefone() {  
    return telefone;  
}
```

```
public String getEmail() {  
    return email;  
}
```

```
public int getId() {  
    return id;  
}
```

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

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

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



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

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

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

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

```
public void exibir(){  
    System.out.println("Id: "+this.getId());  
    System.out.println("Nome: "+this.getNome());  
    System.out.println("Logradouro: "+this.getLogradouro());  
    System.out.println("Cidade: "+this.getCidade());  
    System.out.println("Estado: "+this.getEstado());  
    System.out.println("Telefone: "+this.getTelefone());  
    System.out.println("Email: "+this.getEmail());  
}
```

```
}
```

PessoaFisica

```
package cadastrobd.model;

/**
 *
 * @author Maiara
 */
import cadastro.model.PessoaFisicaDAO;
import java.io.Serializable;
import java.util.Scanner;

public class PessoaFisica extends Pessoa implements Serializable {
    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 exibir(){
```

```
        super.exibir();  
        System.out.println("CPF: "+this.getCpf());  
    }  
}
```

PessoaJuridica

```
package cadastrbd.model;  
  
/**  
 *  
 * @author Maiara  
 */  
  
import cadastro.model.PessoaJuridicaDAO;  
import java.io.Serializable;  
import java.util.Scanner;  
  
public class PessoaJuridica extends Pessoa implements Serializable {  
    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 exibir(){

    super.exibir();

    System.out.println("CNPJ: "+this.getCnpj());

}

}

```

ConectorBD

```

package cadastro.model.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/*
    Autor: Maiara
*/

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";

    public Connection getConnection() throws SQLException {

        return DriverManager.getConnection(URL, USUARIO, SENHA);

    }

}

```

```

public PreparedStatement getPreparedStatement(String sql) throws SQLException {
    Connection connection = getConnection();
    return connection.prepareStatement(sql);
}

public ResultSet executeQuery(String sql) throws SQLException {
    Statement statement = null;
    ResultSet resultSet = null;

    try {
        Connection connection = getConnection();
        statement = connection.createStatement();
        resultSet = statement.executeQuery(sql);
        return resultSet;
    } finally {
        close(statement);
    }
}

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

public void close(Statement statement) {
    try {
        if (statement != null) {
            statement.close();
        }
    } catch (SQLException e) {
        System.out.println(e);
    }
}

```

```

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

```

SequenceManager

```

package cadastro.model.util;

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

/**
 *
 * @author Maiara
 */
public class SequenceManager {
    private final ConectorBD conectorBD;

    public SequenceManager() {
        this.conectorBD = new ConectorBD();
    }

    public int getNextValue(String sequenceName) throws SQLException {
        int proximoValor = 0;
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {

```

```

        connection = conectorBD.getConnection();
        String sql = "SELECT NEXT VALUE FOR " + sequenceName + " AS NextVal";
        preparedStatement = connection.prepareStatement(sql);
        resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            proximoValor = resultSet.getInt("NextVal");
        }
    } finally {
        conectorBD.close(resultSet);
        conectorBD.close(preparedStatement);
        conectorBD.close(connection);
    }

    return proximoValor;
}
}

```

PessoaFisicaDAO

```

package cadastro.model;

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

/**
 *
 * @author Maiara
 */
public class PessoaFisicaDAO {

```

```
private final ConectorBD conectorBD;

public PessoaFisicaDAO() {
    this.conectorBD = new ConectorBD();
}

public PessoaFisica getPessoaFisica(int id) {
    PessoaFisica pessoaFisica = null;
    Connection connection = null;
    PreparedStatement statementPessoa = null;
    PreparedStatement statementPessoaFisica = null;
    ResultSet resultSetPessoa = null;
    ResultSet resultSetPessoaFisica = null;

    try {
        connection = conectorBD.getConnection();

        String sqlPessoa = "SELECT * FROM Pessoa WHERE idPessoa = ?";

        statementPessoa = connection.prepareStatement(sqlPessoa);
        statementPessoa.setInt(1, id);
        resultSetPessoa = statementPessoa.executeQuery();

        if (resultSetPessoa.next()) {
            pessoaFisica = new PessoaFisica();
            pessoaFisica.setId(resultSetPessoa.getInt("idPessoa"));
            pessoaFisica.setNome(resultSetPessoa.getString("nome"));
            pessoaFisica.setLogradouro(resultSetPessoa.getString("logradouro"));
            pessoaFisica.setCidade(resultSetPessoa.getString("cidade"));
            pessoaFisica.setEstado(resultSetPessoa.getString("estado"));
            pessoaFisica.setEmail(resultSetPessoa.getString("email"));
            pessoaFisica.setTelefone(resultSetPessoa.getString("telefone"));
        }

        String sqlPessoaFisica = "SELECT * FROM PessoaFisica WHERE idPessoa = ?";
        statementPessoaFisica = connection.prepareStatement(sqlPessoaFisica);
        statementPessoaFisica.setInt(1, id);
        resultSetPessoaFisica = statementPessoaFisica.executeQuery();
    }
}
```



```

        if (resultSetPessoaFisica.next()) {
            pessoaFisica.setCpf(resultSetPessoaFisica.getString("CPF"));
        }
        statementPessoaFisica.execute();
    } catch (SQLException e) {
        System.out.println("Erro. Não foi possível concluir a solicitação " + e);
    } finally {
        conectorBD.close(resultSetPessoaFisica);
        conectorBD.close(statementPessoaFisica);
        conectorBD.close(resultSetPessoa);
        conectorBD.close(statementPessoa);
        conectorBD.close(connection);
    }

    return pessoaFisica;
}

public List<PessoaFisica> listarTodasPessoasFisicas() {
    List<PessoaFisica> pessoasFisicas = new ArrayList<>();
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;

    try {
        connection = conectorBD.getConnection();

        String sql = "SELECT * FROM Pessoa RIGHT JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa";
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();

        while (resultSet.next()) {
            PessoaFisica pessoaFisica = new PessoaFisica();
            pessoaFisica.setId(resultSet.getInt("idPessoa"));
            pessoaFisica.setNome(resultSet.getString("nome"));
            pessoaFisica.setLogradouro(resultSet.getString("logradouro"));
            pessoaFisica.setCidade(resultSet.getString("cidade"));
            pessoaFisica.setEstado(resultSet.getString("estado"));
        }
    }
}

```

```

        pessoaFisica.setEmail(resultSet.getString("email"));
        pessoaFisica.setTelefone(resultSet.getString("telefone"));
        pessoaFisica.setCpf(resultSet.getString("CPF"));

        pessoasFisicas.add(pessoaFisica);
    }
} catch (SQLException e) {
    System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
} finally {
    conectorBD.close(resultSet);
    conectorBD.close(statement);
    conectorBD.close(connection);
}

return pessoasFisicas;
}

public void incluirPessoaFisica(PessoaFisica pessoaFisica) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = conectorBD.getConnection();
        connection.setAutoCommit(false);

        SequenceManager sequenceManager = new SequenceManager();
        int idPessoa = sequenceManager.getNextValue("PessoaSequence");

        String sqlPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
        preparedStatement = connection.prepareStatement(sqlPessoa);

        preparedStatement.setInt(1, idPessoa);
        preparedStatement.setString(2, pessoaFisica.getNome());
        preparedStatement.setString(3, pessoaFisica.getLogradouro());
        preparedStatement.setString(4, pessoaFisica.getCidade());
        preparedStatement.setString(5, pessoaFisica.getEstado());
        preparedStatement.setString(7, pessoaFisica.getEmail());
        preparedStatement.setString(6, pessoaFisica.getTelefone());
    }
}

```

```

        preparedStatement.execute();

        String sql = "INSERT INTO PessoaFisica (idPessoa, CPF) VALUES (?, ?)";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt(1, idPessoa);
        preparedStatement.setString(2, pessoaFisica.getCpf());
        preparedStatement.execute();

        try {
            connection.commit();
        } catch (SQLException e) {
            System.out.println("Erro no commit");
        }

    } catch (SQLException e) {
        if (connection != null) {
            try {
                connection.rollback();
            } catch (SQLException ex) {
                System.out.println("Erro ao fazer rollback: " + ex);
            }
        }

        System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
    } finally {
        conectorBD.close(preparedStatement);
        conectorBD.close(connection);
    }
}

public void alterarPessoaFisica(PessoaFisica pessoaFisicaAntiga, PessoaFisica pessoaFisicaNova) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = conectorBD.getConnection();
        connection.setAutoCommit(false);

        String sqlConsulta = "SELECT idPessoa FROM Pessoa WHERE idPessoa = ?";

```

```

preparedStatement = connection.prepareStatement(sqlConsulta);
preparedStatement.setInt(1, pessoaFisicaAntiga.getId());
ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

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

    preparedStatement = connection.prepareStatement(sqlPessoa);
    preparedStatement.setString(1, pessoaFisicaNova.getNome());
    preparedStatement.setString(2, pessoaFisicaNova.getLogradouro());
    preparedStatement.setString(3, pessoaFisicaNova.getCidade());
    preparedStatement.setString(4, pessoaFisicaNova.getEstado());
    preparedStatement.setString(5, pessoaFisicaNova.getEmail());
    preparedStatement.setString(6, pessoaFisicaNova.getTelefone());
    preparedStatement.setInt(7, pessoaFisicaAntiga.getId()); //

    preparedStatement.executeUpdate();

    String sqlConsultaPF = "SELECT idPessoa FROM PessoaFisica WHERE idPessoa = ?";
    preparedStatement = connection.prepareStatement(sqlConsultaPF);
    preparedStatement.setInt(1, pessoaFisicaAntiga.getId());
    resultSet = preparedStatement.executeQuery();

    if (resultSet.next()) {
        String sqlPessoaFisica = "UPDATE PessoaFisica SET CPF = ? WHERE idPessoa = ?";
        preparedStatement = connection.prepareStatement(sqlPessoaFisica);
        preparedStatement.setString(1, pessoaFisicaNova.getCpf());
        preparedStatement.setInt(2, pessoaFisicaAntiga.getId());

        preparedStatement.executeUpdate();

        try {
            connection.commit();
        } catch (SQLException e) {
            System.out.println("Erro no commit");
        }
    } else {
        System.out.println("ID não encontrado para Pessoa Física na base de dados.");
    }
}

```

```

    }
} else {
    System.out.println("ID não encontrado para Pessoa na base de dados.");
}
} catch (SQLException e) {
    if (connection != null) {
        try {
            connection.rollback();
        } catch (SQLException ex) {
            System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
            System.out.println("Código de Erro SQL: " + e.getErrorCode());
        }
    }
} finally {
    conectorBD.close(preparedStatement);
    conectorBD.close(connection);
}
}

```

// Opção 4 do Menu

```

public PessoaFisica getPessoaFisicaById(int id) {
    PessoaFisica pessoaFisica = null;
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;

    try {
        connection = conectorBD.getConnection();

        String sql = "SELECT * FROM Pessoa RIGHT JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa WHERE Pessoa.idPessoa = ?";
        statement = connection.prepareStatement(sql);
        statement.setInt(1, id);
        resultSet = statement.executeQuery();

        if (resultSet.next()) {

            pessoaFisica = new PessoaFisica();
            pessoaFisica.setId(resultSet.getInt("idPessoa"));

```

```

        pessoaFisica.setNome(resultSet.getString("nome"));
        pessoaFisica.setLogradouro(resultSet.getString("logradouro"));
        pessoaFisica.setCidade(resultSet.getString("cidade"));
        pessoaFisica.setEstado(resultSet.getString("estado"));
        pessoaFisica.setEmail(resultSet.getString("email"));
        pessoaFisica.setTelefone(resultSet.getString("telefone"));
        pessoaFisica.setCpf(resultSet.getString("CPF"));

    }
} catch (SQLException e) {
    System.out.println(e);
    System.out.println("Código de Erro SQL: " + e.getErrorCode());
} finally {
    conectorBD.close(resultSet);
    conectorBD.close(statement);
    conectorBD.close(connection);
}

return pessoaFisica;
}

public void excluirPessoaFisica(int id) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = conectorBD.getConnection();
        connection.setAutoCommit(false);

        String sqlConsultaPF = "SELECT idPessoa FROM PessoaFisica WHERE idPessoa = ?";
        preparedStatement = connection.prepareStatement(sqlConsultaPF);
        preparedStatement.setInt(1, id);
        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            String sqlDeletaPF = "DELETE FROM PessoaFisica WHERE idPessoa = ?";
            preparedStatement = connection.prepareStatement(sqlDeletaPF);
            preparedStatement.setInt(1, id);

```

```

preparedStatement.executeUpdate();

String sqlConsultaPessoa = "SELECT idPessoa FROM Pessoa WHERE idPessoa = ?";
preparedStatement = connection.prepareStatement(sqlConsultaPessoa);
preparedStatement.setInt(1, id);
resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {
    String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
    preparedStatement = connection.prepareStatement(sqlPessoa);
    preparedStatement.setInt(1, id);
    preparedStatement.executeUpdate();

    try {
        connection.commit();
        System.out.println("Commit realizado");
    } catch (SQLException e) {
        System.out.println("Erro no commit");
    }

} else {
    System.out.println("ID não encontrado para Pessoa na base de dados.");
}

} else {
    System.out.println("ID não encontrado para Pessoa Física na base de dados.");
}

} catch (SQLException e) {
    System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
    System.out.println("Código de Erro SQL: " + e.getErrorCode());

    if (connection != null) {
        try {
            connection.rollback();
        } catch (SQLException ex) {
            System.out.println("Erro ao fazer rollback: " + ex);
        }
    }
} finally {
    conectorBD.close(preparedStatement);
}

```

```
        conectorBD.close(connection);
    }
}
}
```

PessoaJuridicaDAO

```
package cadastro.model;

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

/**
 *
 * @author Maiara
 */

public class PessoaJuridicaDAO {

    private final ConectorBD conectorBD;

    public PessoaJuridicaDAO() {
        this.conectorBD = new ConectorBD();
    }

    public PessoaJuridica getPessoaJuridica(int id) {
        PessoaJuridica pessoaJuridica = null;
        Connection connection = null;
        PreparedStatement statementPessoa = null;
        PreparedStatement statementPessoaJuridica = null;
```



```
ResultSet resultSetPessoa = null;
ResultSet resultSetPessoaJuridica = null;

try {
    connection = conectorBD.getConnection();

    String sqlPessoa = "SELECT * FROM Pessoa WHERE idPessoa = ?";

    statementPessoa = connection.prepareStatement(sqlPessoa);
    statementPessoa.setInt(1, id);
    resultSetPessoa = statementPessoa.executeQuery();

    if (resultSetPessoa.next()) {
        pessoaJuridica = new PessoaJuridica();
        pessoaJuridica.setId(resultSetPessoa.getInt("idPessoa"));
        pessoaJuridica.setNome(resultSetPessoa.getString("nome"));
        pessoaJuridica.setLogradouro(resultSetPessoa.getString("logradouro"));
        pessoaJuridica.setCidade(resultSetPessoa.getString("cidade"));
        pessoaJuridica.setEstado(resultSetPessoa.getString("estado"));
        pessoaJuridica.setEmail(resultSetPessoa.getString("email"));
        pessoaJuridica.setTelefone(resultSetPessoa.getString("telefone"));

    }

    String sqlPessoaJuridica = "SELECT * FROM PessoaJuridica WHERE idPessoa = ?";
    statementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica);
    statementPessoaJuridica.setInt(1, id);
    resultSetPessoaJuridica = statementPessoaJuridica.executeQuery();

    if (resultSetPessoaJuridica.next()) {
        pessoaJuridica.setCnpj(resultSetPessoaJuridica.getString("CNPJ"));
    }
    statementPessoaJuridica.execute();
} catch (SQLException e) {
    System.out.println("Erro. Não foi possível concluir a solicitação " + e);
} finally {
    conectorBD.close(resultSetPessoaJuridica);
    conectorBD.close(statementPessoaJuridica);
    conectorBD.close(resultSetPessoa);
}
```

```

        conectorBD.close(statementPessoa);
        conectorBD.close(connection);
    }

    return pessoaJuridica;
}

public List<PessoaJuridica> listarTodasPessoasJuridicas() {
    List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;

    try {
        connection = conectorBD.getConnection();

        String sql = "SELECT * FROM Pessoa RIGHT JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoa";
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();

        while (resultSet.next()) {
            PessoaJuridica pessoaJuridica = new PessoaJuridica();
            pessoaJuridica.setId(resultSet.getInt("idPessoa"));
            pessoaJuridica.setNome(resultSet.getString("nome"));
            pessoaJuridica.setLogradouro(resultSet.getString("logradouro"));
            pessoaJuridica.setCidade(resultSet.getString("cidade"));
            pessoaJuridica.setEstado(resultSet.getString("estado"));
            pessoaJuridica.setEmail(resultSet.getString("email"));
            pessoaJuridica.setTelefone(resultSet.getString("telefone"));
            pessoaJuridica.setCnpj(resultSet.getString("CNPJ"));

            pessoasJuridicas.add(pessoaJuridica);
        }
    } catch (SQLException e) {
        System.out.println("Erro. Não foi possível concluir a solicitação "+e);
    } finally {
        conectorBD.close(resultSet);
    }
}

```

```

        conectorBD.close(statement);
        conectorBD.close(connection);
    }

    return pessoasJuridicas;
}

public void incluirPessoaJuridica(PessoaJuridica pessoaJuridica) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = conectorBD.getConnection();
        connection.setAutoCommit(false);

        SequenceManager sequenceManager = new SequenceManager();
        int idPessoa = sequenceManager.getNextValue("PessoaSequence");

        String sqlPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
        preparedStatement = connection.prepareStatement(sqlPessoa);

        preparedStatement.setInt(1, idPessoa);
        preparedStatement.setString(2, pessoaJuridica.getNome());
        preparedStatement.setString(3, pessoaJuridica.getLogradouro());
        preparedStatement.setString(4, pessoaJuridica.getCidade());
        preparedStatement.setString(5, pessoaJuridica.getEstado());
        preparedStatement.setString(7, pessoaJuridica.getEmail());
        preparedStatement.setString(6, pessoaJuridica.getTelefone());

        preparedStatement.execute();

        String sql = "INSERT INTO PessoaJuridica (idPessoa, CNPJ) VALUES (?, ?)";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt(1, idPessoa);
        preparedStatement.setString(2, pessoaJuridica.getCnpj());
        preparedStatement.execute();
    }
}

```

```

        connection.commit();
    } catch (SQLException e) {
        System.out.println("Erro no commit");
    }

} catch (SQLException e) {
    if (connection != null) {
        try {
            connection.rollback();
        } catch (SQLException ex) {
            System.out.println("Erro ao fazer rollback: " + ex);
        }
    }
    System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
} finally {
    conectorBD.close(preparedStatement);
    conectorBD.close(connection);
}
}

public void alterarPessoaJuridica(PessoaJuridica pessoaJuridicaAntiga, PessoaJuridica
pessoaJuridicaNova) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = conectorBD.getConnection();
        connection.setAutoCommit(false);

        String sqlConsulta = "SELECT idPessoa FROM Pessoa WHERE idPessoa = ?";
        preparedStatement = connection.prepareStatement(sqlConsulta);
        preparedStatement.setInt(1, pessoaJuridicaAntiga.getId());
        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {

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

```

```

preparedStatement = connection.prepareStatement(sqlPessoa);
preparedStatement.setString(1, pessoaJuridicaNova.getNome());
preparedStatement.setString(2, pessoaJuridicaNova.getLogradouro());
preparedStatement.setString(3, pessoaJuridicaNova.getCidade());
preparedStatement.setString(4, pessoaJuridicaNova.getEstado());
preparedStatement.setString(5, pessoaJuridicaNova.getEmail());
preparedStatement.setString(6, pessoaJuridicaNova.getTelefone());
preparedStatement.setInt(7, pessoaJuridicaAntiga.getId());

preparedStatement.executeUpdate();

String sqlConsultaPJ = "SELECT idPessoa FROM PessoaJuridica WHERE idPessoa = ?";
preparedStatement = connection.prepareStatement(sqlConsultaPJ);
preparedStatement.setInt(1, pessoaJuridicaAntiga.getId());
resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {
    String sqlPessoaJuridica = "UPDATE PessoaJuridica SET CNPJ = ? WHERE idPessoa = ?";
    preparedStatement = connection.prepareStatement(sqlPessoaJuridica);
    preparedStatement.setString(1, pessoaJuridicaNova.getCnpj());
    preparedStatement.setInt(2, pessoaJuridicaAntiga.getId());

    preparedStatement.executeUpdate();

    try {
        connection.commit();
    } catch (SQLException e) {
        System.out.println("Erro no commit");
    }
} else {
    System.out.println("ID não encontrado para Pessoa Jurídica na base de dados.");
}
} else {
    System.out.println("ID não encontrado para Pessoa na base de dados.");
}
} catch (SQLException e) {
    if (connection != null) {
        try {
            connection.rollback();

```

```

        } catch (SQLException ex) {
            System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
        }
    }
} finally {
    conectorBD.close(preparedStatement);
    conectorBD.close(connection);
}
}

public PessoaJuridica getPessoaJuridicaById(int id) {
    PessoaJuridica pessoaJuridica = null;
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;

    try {
        connection = conectorBD.getConnection();

        String sql = "SELECT * FROM Pessoa RIGHT JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoa WHERE Pessoa.idPessoa = ?";
        statement = connection.prepareStatement(sql);
        statement.setInt(1, id);
        resultSet = statement.executeQuery();

        if (resultSet.next()) {

            pessoaJuridica = new PessoaJuridica();
            pessoaJuridica.setId(resultSet.getInt("idPessoa"));
            pessoaJuridica.setNome(resultSet.getString("nome"));
            pessoaJuridica.setLogradouro(resultSet.getString("logradouro"));
            pessoaJuridica.setCidade(resultSet.getString("cidade"));
            pessoaJuridica.setEstado(resultSet.getString("estado"));
            pessoaJuridica.setEmail(resultSet.getString("email"));
            pessoaJuridica.setTelefone(resultSet.getString("telefone"));
            pessoaJuridica.setCnpj(resultSet.getString("CNPJ"));

        }
    }
}

```

```

    } catch (SQLException e) {
        System.out.println("Erro. Não foi possível concluir a solicitação "+e);
    } finally {
        conectorBD.close(resultSet);
        conectorBD.close(statement);
        conectorBD.close(connection);
    }

    return pessoaJuridica;
}

public void excluirPessoaJuridica(int id) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = conectorBD.getConnection();
        connection.setAutoCommit(false);

        String sqlConsultaPJ = "SELECT idPessoa FROM PessoaJuridica WHERE idPessoa = ?";
        preparedStatement = connection.prepareStatement(sqlConsultaPJ);
        preparedStatement.setInt(1, id);
        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            String sqlDeletaPJ = "DELETE FROM PessoaJuridica WHERE idPessoa = ?";
            preparedStatement = connection.prepareStatement(sqlDeletaPJ);
            preparedStatement.setInt(1, id);
            preparedStatement.executeUpdate();

            String sqlConsultaPessoa = "SELECT idPessoa FROM Pessoa WHERE idPessoa = ?";
            preparedStatement = connection.prepareStatement(sqlConsultaPessoa);
            preparedStatement.setInt(1, id);
            resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
                preparedStatement = connection.prepareStatement(sqlPessoa);

```

```

        preparedStatement.setInt(1, id);
        preparedStatement.executeUpdate();

        try {
            connection.commit();
            System.out.println("Commit realizado");
        } catch (SQLException e) {
            System.out.println("Erro no commit");
        }

        } else {
            System.out.println("ID não encontrado para Pessoa na base de dados.");
        }
    } else {
        System.out.println("ID não encontrado para Pessoa Jurídica na base de dados.");
    }
} catch (SQLException e) {
    System.out.println("Erro. Não foi possível concluir a solicitação: " + e);
    System.out.println("Código de Erro SQL: " + e.getErrorCode());

    if (connection != null) {
        try {
            connection.rollback();
        } catch (SQLException ex) {
            System.out.println("Erro ao fazer rollback: " + ex);
        }
    }
} finally {
    conectorBD.close(preparedStatement);
    conectorBD.close(connection);
}
}
}

```

Análise e Conclusão:

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

Resposta: os middleware são utilizados como uma camada intermediária que realizam a comunicação entre o Java e o Banco de dados. Os middlewares permitem a integração entre diferentes sistemas e podem traduzir diferentes protocolos de comunicação, formatos de dados e sistemas operacionais.

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

Resposta: O PreparedStatement é compilado e parametrizado previamente. Dessa forma as consultas podem ser criadas com espaços reservados para parâmetros por meio do símbolo de interrogação e os parâmetros podem ser definidos separados em seguida. Isso evita SQL injection.

O Statement não realiza tratamento especial das consultas, o que torna a aplicação vulnerável a ataques de injeção de SQL injection.

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

Resposta: Organização do código e separação de responsabilidades, que permite alteração da lógica de acesso aos dados sem afetar outras partes do código. Dessa forma é possível alterar o tipo de banco de dados, reutilizar códigos e introduzir recursos adicionais sem necessidade de revisitar toda a aplicação.

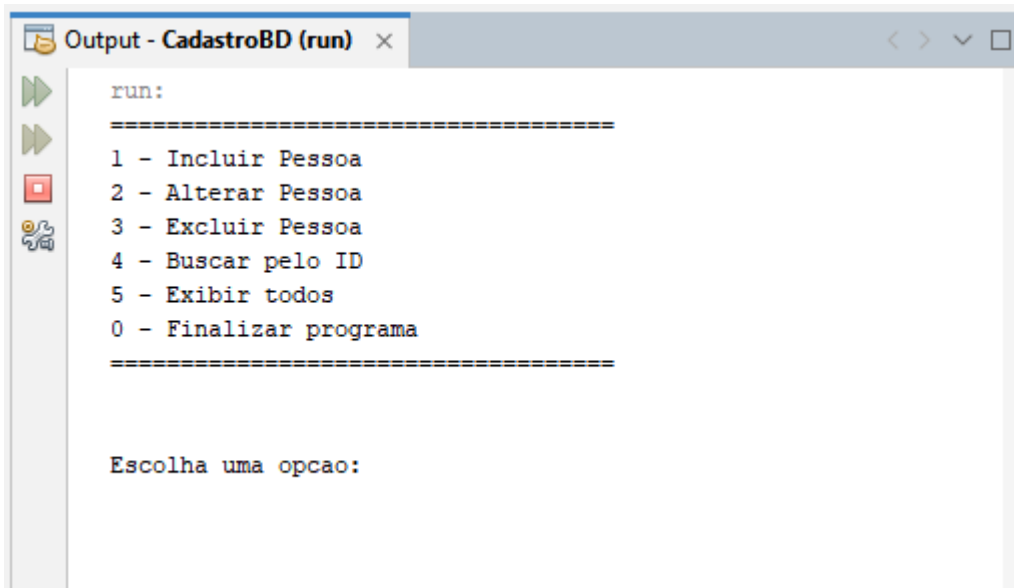
- d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

Resposta: A herança pode ser mapeada de diversas formas como Classe, Tabela de tipo único e tabela por subclasse. Neste trabalho, segui a forma de tabela por subclasse, o que significa que cada classe possui uma tabela referente.

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.

2º Procedimento – Alimentando a Base

Rodando a aplicação



```
Output - CadastroBD (run) x
run:
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====

Escolha uma opcao:
```

Incluir Pessoa Física:

```
run:
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====

Escolha uma opcao: 1

F - Pessoa Fisica | J - Pessoa Juridica f
Digite o nome da pessoa fisica: Karlach
Digite o CPF da pessoa fisica: 12345678985
Digite o Logradouro da pessoa fisica: Avenus 9
Digite o cidade da pessoa fisica: Faerun
Digite o estado da pessoa fisica: FE
Digite o telefone da pessoa fisica: 529456128245
Digite o email da pessoa fisica: karlachavenus@gmail.com
Pessoa fisica criada.
```

Incluir Pessoa Jurídica:

run:

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====
```

Escolha uma opcao: 1

```
F - Pessoa Fisica | J - Pessoa Juridica j
Digite o nome da pessoa jurídica: Larian Studios
Digite o CNPJ da pessoa jurídica: 00.123.321/0001-52
Digite o Logradouro da pessoa jurídica: Av. Avenus, 9
Digite a cidade da pessoa jurídica: Nova York
Digite o estado da pessoa jurídica: NY
Digite o telefone da pessoa jurídica: 856985695
Digite o email da pessoa jurídica: contato@larianstudios.com
Pessoa jurídica incluída com sucesso.
```

Alterar Pessoa Física:

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====
```

Escolha uma opcao: 2

```
F - Pessoa Fisica | J - Pessoa Juridica f
Digite o ID da pessoa física a ser alterada: 80
Pessoa física encontrada:
Id: 80
Nome: Karlach
logradouro: Avenus 9
cidade: Faerun
estado: FE
telefone: 529456128245
email: karlachavenus@gmail.com
CPF: 12345678985
Digite o nome da pessoa física: Karlach Hells
Digite o CPF da pessoa física: 12345678985
Digite o Logradouro da pessoa física: Avenus 9
Digite o cidade da pessoa física: Faerun
Digite o estado da pessoa física: FE
Digite o telefone da pessoa física: 529456128245
Digite o email da pessoa física: karlachhells@gmail.com
Pessoa física alterada com sucesso.
```

Alterar Pessoa Jurídica:

Escolha uma opcao: 2

F - Pessoa Fisica | J - Pessoa Juridica j

Digite o ID da pessoa juridica a ser alterada: 82

Pessoa juridica encontrada:

Id: 82

Nome: Larian Studios

logradouro: Av. Avenus, 9

cidade: Nova York

estado: NY

telefone: 856985695

email: contato@larianstudios.com

CNPJ: 00.123.321/0001-52

Digite o nome da pessoa fisica: Larian Studios

Digite o CNPJ da pessoa fisica: 00.123.321/0001-52

Digite o Logradouro da pessoa fisica: Grey Street, 52

Digite o cidade da pessoa fisica: Nova York

Digite o estado da pessoa fisica: NY

Digite o telefone da pessoa fisica: 856985695

Digite o email da pessoa fisica: contato@larianstudios.com

Pessoa fisica alterada com sucesso.

Excluir Pessoa Física

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====
```

Escolha uma opcao: 3

F - Pessoa Fisica | J - Pessoa Juridica f

Digite o ID da pessoa fisica para exclusão: 83

Commit realizado

Pessoa fisica excluida com sucesso.

Excluir Pessoa Jurídica

```
Escolha uma opcao: 3

F - Pessoa Fisica | J - Pessoa Juridica j
Digite o ID da pessoa jurídica para exclusão: 4
Commit realizado
Pessoa jurídica excluída com sucesso.
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====
```

Pessoa Física Buscar por ID

```
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====

Escolha uma opcao: 4

F - Pessoa Fisica | J - Pessoa Juridica f
Digite o ID da pessoa Física para exibição:
80
=====

Id: 80
Nome: Karlach Hells
logradouro: Avenus 9
cidade: Faerun
estado: FE
telefone: 529456128245
email: karlachhells@gmail.com
CPF: 12345678985
```

Pessoa Jurídica Buscar por ID

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
=====
```

Escolha uma opção: 4

F - Pessoa Física | J - Pessoa Jurídica j
Digite o ID da pessoa Jurídica para exibição:
82
=====

Id: 82
Nome: Larian Studios
logradouro: Grey Street, 52
cidade: Nova York
estado: NY
telefone: 856985695
email: contato@larianstudios.com
CNPJ: 00.123.321/0001-52

Pessoa Física – Exibir todos

- 1 - Incluir Pessoa
- 2 - Alterar Pessoa
- 3 - Excluir Pessoa
- 4 - Buscar pelo ID
- 5 - Exibir todos
- 0 - Finalizar programa

=====

Escolha uma opcao: 5

F - Pessoa Fisica | J - Pessoa Juridica f

Lista de Pessoas Físicas

=====

Id: 80

Nome: Karlach Hells

logradouro: Avenus 9

cidade: Faerun

estado: FE

telefone: 529456128245

email: karlachhells@gmail.com

CPF: 12345678985

=====

Id: 84

Nome: Nathan Drake

logradouro: Rua Topazio, 13

cidade: Esmeralda do Sul

estado: PO

telefone: 4578913456

email: drakenate@gmail.com

CPF: 4567891345

=====

Pessoa Física – Exibir todos

- 1 - Incluir Pessoa
- 2 - Alterar Pessoa
- 3 - Excluir Pessoa
- 4 - Buscar pelo ID
- 5 - Exibir todos
- 0 - Finalizar programa

=====

Escolha uma opcao: 5

F - Pessoa Fisica | J - Pessoa Juridica j

Lista de Pessoas Jurídicas

=====

Id: 5
Nome: ABC
logradouro: Av. Alm. Pedrosa
cidade: Rio de Janeiro
estado: RJ
telefone: 3333-2222
email: abc@riacho.com
CNPJ: 22.111.122/0001-00

=====

Id: 6
Nome: Frutalicia LTDA
logradouro: Rua Paz
cidade: Rio de Janeiro
estado: RJ
telefone: 4444-1111
email: Frutalicia@riacho.com
CNPJ: 11.352.168/0001-00

=====

Id: 82
Nome: Larian Studios
logradouro: Grey Street, 52
cidade: Nova York
estado: NY
telefone: 856985695
email: contato@larianstudios.com
CNPJ: 00.123.321/0001-52

=====

Sair

```
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar programa
```

```
=====
```

```
Escolha uma opcao: 0
```

```
BUILD SUCCESSFUL (total time: 11 minutes 10 seconds)
```

```
|
```

Análise e Conclusão:

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

Resposta:

Na persistência em arquivo, os dados são armazenados em arquivos de diferentes formatos, como csv, xml, json etc.

Na persistência em banco de dados, os dados são armazenados em tabelas dentro de um SGBD.

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

Resposta: Antes da introdução dos lambdas, era necessário escrever explicitamente classes anônimas ou implementar interfaces funcionais para realizar tarefas simples, como imprimir os valores de um objeto. Os operadores lambda tornaram essa tarefa mais concisa e legível.

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

Resposta: Porque o método main é inicializado pelo sistema antes de qualquer outro objeto. O termo static indica que que método main pertence a classe e, por isso, pode ser chamado sem a criação de outro objeto.

Conclusão

Nesta atividade, revisei as duas atividades anteriores (nível 1 e 2) criando a conexão do que havia sido construído até o momento e aplicando os novos conhecimentos adquiridos durante o estudo do material.

Primeiro, aprendi como realizar a conexão do banco de dados criado no SGBD SQL Server ao IDE na classe ConectorBD com o middleware JDBC.

Além das classes dos objetos Pessoa, PessoaFisica e PessoaJuridica no pacote modelo, implementei as classes PessoaFisicaDAO e PessoaJuridicaDAO para persistência dos dados no SGBD e independência da lógica de dados.

Também foi criada uma classe SequenceManager para realizar o gerenciamento da criação dos índices e chaves primárias das tabelas.

Na classe CadastroBD, foi criado o método main e toda a lógica da aplicação.

Foi necessário a criação de um novo banco de dados, pois as dependências criadas no banco da atividade 2 geravam conflito com ações de exclusão, o que foi percebido por meio da aplicação de tratamentos de erro.