



## Missão Prática - 1º Procedimento

Andrey Haertel Aires - Matrícula: 2021.07.22851-2

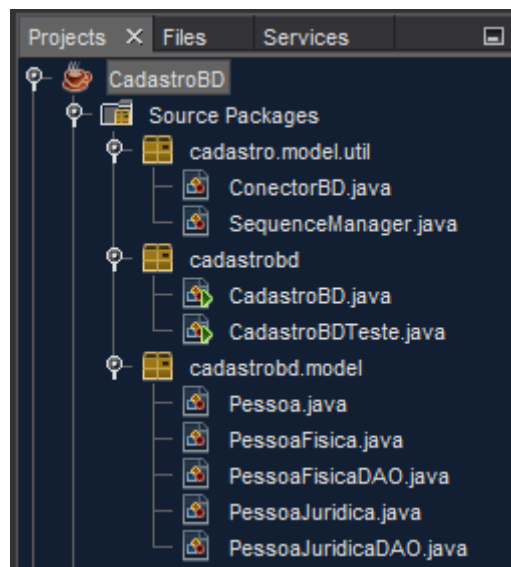
Polo Centro - Palhoça – SC

Nível 3: Back-end Sem Banco Não Tem – T 9001 – 3º Semestre Letivo

### 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.
- No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

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



Projeto criado no Netbeans com o nome 'CadastroBD' e seus pacotes:

**cadastro.model.util**

Com as classes: ConectorBD e SequenceManager.

*Ambas as classes desempenham papéis importantes no contexto de interação com um banco de dados. A ConectorBD cuida da gestão da conexão, enquanto o SequenceManager lida com a obtenção de valores sequenciais.*

## cadastrobd

Com as classes: CadastroBD e CadastroBDTeste.

*CadastroBD é a classe principal, mas ainda não foi implementada nesse primeiro procedimento da missão pratica.*

*A classe CadastroBDTeste é um exemplo de aplicação que utiliza as classes PessoaFisicaDAO e PessoaJuridicaDAO para realizar operações de cadastro e consulta no banco de dados.*

## cadastrobd.model

Com as classes: Pessoa, PessoaFisica, PessoaFisicaDAO, PessoaJuridica, PessoaJuridicaDAO .

*Pessoa representa uma entidade genérica de pessoa, contendo informações como nome, endereço, telefone e email. Uso: Base para as classes especializadas PessoaFisica e PessoaJuridica.*

*PessoaFisica estende a classe Pessoa, adicionando o campo cpf e métodos específicos para manipulação de pessoas físicas.*

*PessoaFisicaDAO interage com o banco de dados para operações relacionadas a pessoas físicas. Fornece métodos para exibir, incluir e excluir registros. Facilita o acesso e manipulação de dados de pessoas físicas no banco de dados.*

*PessoaJuridica estende a classe Pessoa, adicionando o campo cnpj e métodos específicos para manipulação de pessoas jurídicas.*

*PessoaJuridicaDAO interage com o banco de dados para operações relacionadas a pessoas jurídicas. Fornece métodos para exibir, incluir e excluir registros. Facilita o acesso e manipulação de dados de pessoas jurídicas no banco de dados.*

## Codigos das Classes:

### ConectorBD

```
package cadastro.model.util;
/**
 *
 * @author Andrey H Aires
 */
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ConectorBD {
    // Configurações de conexão com o banco de dados
    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";

    // Metodo para obter uma conexão com o banco de dados
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USUARIO, SENHA);
    }

    // Metodo para obter um objeto PreparedStatement a partir de um SQL fornecido
    public static PreparedStatement getPrepared(String sql) throws SQLException {
        return getConnection().prepareStatement(sql);
    }
}
```

```

// Metodo para obter um ResultSet relacionado a uma consulta
public static ResultSet getSelect(String sql) throws SQLException {
    return getPrepared(sql).executeQuery();
}

// Metodo para fechar um Statement
public static void close(Statement statement) {
    if (statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

// Metodo para fechar um ResultSet
public static void close(ResultSet resultSet) {
    if (resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

// Metodo para fechar uma Connection
public static void close(Connection connection) {
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}

```

## SequenceManager

```

package cadastro.model.util;
/**
 *
 * @author Andrey H Aires
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SequenceManager {
    // Metodo para obter o próximo valor de uma sequencia
    public static int getValue(String sequenceName) {
        int nextValue = -1;

        String sql = "SELECT NEXTVAL(?) AS next_value";

        try (Connection connection = ConectorBD.getConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setString(1, sequenceName);

            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                if (resultSet.next()) {
                    nextValue = resultSet.getInt("next_value");
                }
            }
        }
    }
}

```

```

    }

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

    return nextValue;
}
}

```

## CadastroBDTeste

```

package cadastrobd;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import cadastro.model.util.ConectorBD;

import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.sql.Connection;
import java.sql.SQLException;

public class CadastroBDTeste {

    public static void main(String[] args) throws UnsupportedEncodingException {
        System.setOut(new PrintStream(System.out, true, "UTF-8"));
        try (Connection connection = ConectorBD.getConnection()) {
            PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO(connection);
            PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO(connection);

            // Dados para inserção de pessoa física
            PessoaFisica pessoaFisicaParaInserir = new PessoaFisica();
            pessoaFisicaParaInserir.setNome("João da Silva");
            pessoaFisicaParaInserir.setLogradouro("Rua 12, casa 3, Quitanda");
            pessoaFisicaParaInserir.setCidade("Riacho do Sul");
            pessoaFisicaParaInserir.setEstado("PA");
            pessoaFisicaParaInserir.setTelefone("47-32552311");
            pessoaFisicaParaInserir.setEmail("joao@riacho.com");
            pessoaFisicaParaInserir.setCpf("841.329.470-33");

            // Inserir pessoa física
            pessoaFisicaDAO.incluirPessoaFisica(pessoaFisicaParaInserir);
            System.out.println("\n*Cadastro de Pessoa Física Realizado com Sucesso*");

            // Alterar uma Pessoa Física no banco de dados
            alterarPessoaFisica(connection, pessoaFisicaDAO);
            System.out.println("\n*Dados de Pessoa Física Alterados*");

            // Exibir todos os cadastros de pessoa física
            System.out.println("\n----- Exibindo Cadastros de Pessoa Física -----");
            pessoaFisicaDAO.exibirTodosFisica().forEach(System.out::println);

            // Excluir cadastro de Pessoa Física pelo ID
            int idParaExcluirPF = 7; // Substitua pelo ID da Pessoa Física que deseja excluir
            pessoaFisicaDAO.excluirPessoaFisica(idParaExcluirPF);
            System.out.println("\n*Cadastro de Pessoa Fisica Excluido com Sucesso*");

            // Dados para inserção de pessoa jurídica
            PessoaJuridica pessoaJuridicaParaInserir = new PessoaJuridica();
            pessoaJuridicaParaInserir.setNome("Microsoft Informatica Ltda");
            pessoaJuridicaParaInserir.setLogradouro("Av. Presidente Juscelino Kubitscheck, 1.909 - Torre Sul - 16º andar");
            pessoaJuridicaParaInserir.setCidade("São Paulo");
            pessoaJuridicaParaInserir.setEstado("SP");
            pessoaJuridicaParaInserir.setTelefone("48-32431335");

```

```

    pessoaJuridicaParaInserir.setEmail("microsoft@microsoft.com");
    pessoaJuridicaParaInserir.setCnpj("46.857.039/0001-20");

    // Inserir Pessoa Jurídica
    pessoaJuridicaDAO.incluirPessoaJuridica(pessoaJuridicaParaInserir);
    System.out.println("\n*Cadastro de Pessoa Jurídica Realizado com Sucesso*");

    // Alterar uma Pessoa Jurídica no banco de dados
    alterarPessoaJuridica(connection, pessoaJuridicaDAO);
    System.out.println("\n*Dados de Pessoa jurídica Alterados*");

    //Exibir todos os cadastros de pessoa jurídica
    System.out.println("\n----- Exibindo Cadastros de Pessoa Jurídica -----");
    pessoaJuridicaDAO.exibirTodasJuridicas().forEach(System.out::println);

    // Excluir cadastro de Pessoa Jurídica pelo ID
    int idParaExcluirPJ = 8; // Substitua pelo ID da Pessoa Juridica que deseja excluir
    pessoaJuridicaDAO.excluirPessoaJuridica(idParaExcluirPJ);
    System.out.println("\n*Cadastro de Pessoa Jurídica Excluido com Sucesso*");

} catch (SQLException e) {
    System.err.println("Erro ao conectar ao banco de dados: " + e.getMessage());
}
}

// Método para alterar uma Pessoa Física no banco de dados
private static void alterarPessoaFisica(Connection connection, PessoaFisicaDAO pessoaFisicaDAO) {
    int idParaAlterar = 7; // Defina o ID da pessoa que deseja alterar

    try {
        // Verifica se a pessoa com o ID especificado existe no banco
        if (pessoaFisicaDAO.pessoaExiste(idParaAlterar)) {
            // Dados para alteração de pessoa física
            PessoaFisica pessoaFisicaParaAlterar = new PessoaFisica();
            pessoaFisicaParaAlterar.setId(idParaAlterar); // Define o ID da pessoa a ser alterada
            pessoaFisicaParaAlterar.setNome("Maria dos Santos");
            pessoaFisicaParaAlterar.setLogradouro("Rua 15 de Setembro, 45, Centro");
            pessoaFisicaParaAlterar.setCidade("Curitiba");
            pessoaFisicaParaAlterar.setEstado("PR");
            pessoaFisicaParaAlterar.setTelefone("41-32771598");
            pessoaFisicaParaAlterar.setEmail("maria@hotmail.com");
            pessoaFisicaParaAlterar.setCpf("44254189729");

            // Alterar pessoa física no banco
            pessoaFisicaDAO.alterarPessoaFisica(pessoaFisicaParaAlterar);

        } else {
            System.out.println("Pessoa com o ID especificado não encontrada.");
        }
    } catch (SQLException e) {
        System.err.println("Erro ao alterar Pessoa Física no banco de dados: " + e.getMessage());
    }
}

// Método para alterar uma Pessoa Jurídica no banco de dados
private static void alterarPessoaJuridica(Connection connection, PessoaJuridicaDAO pessoaJuridicaDAO) {
    int idParaAlterar = 8; // Defina o ID da pessoa jurídica que deseja alterar

    try {
        // Verifica se a pessoa jurídica com o ID especificado existe no banco
        if (pessoaJuridicaDAO.pessoaExiste(idParaAlterar)) {
            // Dados para alteração de pessoa jurídica
            PessoaJuridica pessoaJuridicaParaAlterar = new PessoaJuridica();
            pessoaJuridicaParaAlterar.setId(idParaAlterar); // Define o ID da pessoa jurídica a ser alterada
            pessoaJuridicaParaAlterar.setNome("IBM");
            pessoaJuridicaParaAlterar.setLogradouro("Rua Nova Veneza, 127, Bela Vista");

```

```

        pessoaJuridicaParaAlterar.setCidade("Ipiranga");
        pessoaJuridicaParaAlterar.setEstado("RS");
        pessoaJuridicaParaAlterar.setTelefone("11-98765432");
        pessoaJuridicaParaAlterar.setEmail("contato@ibm.com");
        pessoaJuridicaParaAlterar.setCnpj("70.673.504/0001-58");

        // Alterar pessoa jurídica no banco
        pessoaJuridicaDAO.alterarPessoaJuridica(pessoaJuridicaParaAlterar);

    } else {
        System.out.println("Pessoa Jurídica com o ID especificado não encontrada.");
    }
} catch (SQLException e) {
    System.err.println("Erro ao alterar Pessoa Jurídica no banco de dados: " + e.getMessage());
}
}
}

```

## Pessoa

```

package cadastrobd.model;
/**
 *
 * @author Andrey H Aires
 */
public class Pessoa {
    // Campos da classe
    private int id_pessoa;
    private String nome_pessoa;
    private String logradouro_pessoa;
    private String cidade_pessoa;
    private String estado_pessoa;
    private String telefone_pessoa;
    private String email_pessoa;

    // Construtor padrao
    public Pessoa() {
    }

    // Construtor completo
    public Pessoa(int id_pessoa, String nome_pessoa, String logradouro_pessoa, String cidade_pessoa, String estado_pessoa, String
    telefone_pessoa, String email_pessoa) {
        this.id_pessoa = id_pessoa;
        this.nome_pessoa = nome_pessoa;
        this.logradouro_pessoa = logradouro_pessoa;
        this.cidade_pessoa = cidade_pessoa;
        this.estado_pessoa = estado_pessoa;
        this.telefone_pessoa = telefone_pessoa;
        this.email_pessoa = email_pessoa;
    }

    // Metodos getters
    public int getId() {
        return id_pessoa;
    }

    public String getNome() {
        return nome_pessoa;
    }

    public String getLogradouro() {
        return logradouro_pessoa;
    }
}

```

```

public String getCidade() {
    return cidade_pessoa;
}

public String getEstado() {
    return estado_pessoa;
}

public String getTelefone() {
    return telefone_pessoa;
}

public String getEmail() {
    return email_pessoa;
}

//Metodo setters
public void setId(int id) {
    this.id_pessoa = id;
}

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

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

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

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

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

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

```

### PessoaFisica

```

package cadastrobd.model;
/**
 *
 * @author Andrey H Aires
 */
public class PessoaFisica extends Pessoa {

    private String cpf;

    // Construtor padrao
    public PessoaFisica() {
        super();
    }

    // Construtor completo

```





```

        resultSet.getString("logradouro_pessoa"),
        resultSet.getString("cidade_pessoa"),
        resultSet.getString("estado_pessoa"),
        resultSet.getString("telefone_pessoa"),
        resultSet.getString("email_pessoa"),
        resultSet.getString("cpf"));

        cadastros.add(cadastro);
    }

    } catch (SQLException e) {
        handleSQLException(e);
    }

    return cadastros;
}

// Verifica se a pessoa com o mesmo ID já existe
public boolean pessoaExiste(int id) throws SQLException {
    String sql = "SELECT id_pessoa FROM Pessoa WHERE id_pessoa = ?";
    try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
        preparedStatement.setInt(1, id);
        try (ResultSet resultSet = preparedStatement.executeQuery()) {
            return resultSet.next();
        }
    }
}

// Inclui um novo registro de Pessoa Física no banco de dados.
public void incluirPessoaFisica(PessoaFisica pessoaFisica) throws SQLException {
    String sqlPessoa = "INSERT INTO Pessoa (nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, telefone_pessoa, email_pessoa) VALUES (?, ?, ?, ?, ?, ?)";
    String sqlPessoaFisica = "INSERT INTO pessoa_fisica (id_pessoa, cpf) VALUES (?, ?)";

    try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa,
        PreparedStatement.RETURN_GENERATED_KEYS);
        PreparedStatement preparedStatementPessoaFisica = connection.prepareStatement(sqlPessoaFisica)) {

        // Verifica se o nome não é nulo antes de inserir
        if (pessoaFisica.getNome() != null) {
            // Iniciar uma transação
            connection.setAutoCommit(false);

            try {
                // Inserir na tabela Pessoa
                preparedStatementPessoa.setString(1, pessoaFisica.getNome());
                preparedStatementPessoa.setString(2, pessoaFisica.getLogradouro());
                preparedStatementPessoa.setString(3, pessoaFisica.getCidade());
                preparedStatementPessoa.setString(4, pessoaFisica.getEstado());
                preparedStatementPessoa.setString(5, pessoaFisica.getTelefone());
                preparedStatementPessoa.setString(6, pessoaFisica.getEmail());
                preparedStatementPessoa.executeUpdate();

                // Obter o ID gerado
                try (ResultSet generatedKeys = preparedStatementPessoa.getGeneratedKeys()) {
                    if (generatedKeys.next()) {
                        int id = generatedKeys.getInt(1);

                        // Inserir na tabela PessoaFisica
                        preparedStatementPessoaFisica.setInt(1, id);
                        preparedStatementPessoaFisica.setString(2, pessoaFisica.getCpf());
                        preparedStatementPessoaFisica.executeUpdate();
                    }
                }
            }
        }

        // Commit da transação
        connection.commit();
    }
}

```

```

    } catch (SQLException e) {
        // Rollback em caso de exceção
        connection.rollback();
        throw e;
    } finally {
        // Restaurar o modo de autocommit
        connection.setAutoCommit(true);
    }
} else {
    System.out.println("Nome da pessoa não pode ser nulo. A inserção foi ignorada.");
}

} catch (SQLException e) {
    handleSQLException(e);
}
}

// Altera um registro de Pessoa Física pelo ID no banco de dados.
public void alterarPessoaFisica(PessoaFisica pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoa SET nome_pessoa=?, logradouro_pessoa=?, cidade_pessoa=?, estado_pessoa=?,
telefone_pessoa=?, email_pessoa=? WHERE id_pessoa=?";
    String sqlPessoaFisica = "UPDATE pessoa_fisica SET cpf=? WHERE id_pessoa=?";

    try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa);
        PreparedStatement preparedStatementPessoaFisica = connection.prepareStatement(sqlPessoaFisica)) {

        // Verifica se o nome não é nulo antes de atualizar
        if (pessoa.getNome() != null) {
            // Iniciar uma transação
            connection.setAutoCommit(false);

            try {
                // Atualizar na tabela Pessoa
                preparedStatementPessoa.setString(1, pessoa.getNome());
                preparedStatementPessoa.setString(2, pessoa.getLogradouro());
                preparedStatementPessoa.setString(3, pessoa.getCidade());
                preparedStatementPessoa.setString(4, pessoa.getEstado());
                preparedStatementPessoa.setString(5, pessoa.getTelefone());
                preparedStatementPessoa.setString(6, pessoa.getEmail());
                preparedStatementPessoa.setInt(7, pessoa.getId());
                preparedStatementPessoa.executeUpdate();

                // Atualizar na tabela PessoaFisica
                preparedStatementPessoaFisica.setString(1, pessoa.getCpf());
                preparedStatementPessoaFisica.setInt(2, pessoa.getId());
                preparedStatementPessoaFisica.executeUpdate();

                // Commit da transação
                connection.commit();
            } catch (SQLException e) {
                // Rollback em caso de exceção
                connection.rollback();
                throw e;
            } finally {
                // Restaurar o modo de autocommit
                connection.setAutoCommit(true);
            }
        } else {
            System.out.println("Nome da pessoa não pode ser nulo. A atualização foi ignorada.");
        }
    } catch (SQLException e) {
        handleSQLException(e);
    }
}

```

```

// Exclui um registro de Pessoa Física pelo ID no banco de dados.
public void excluirPessoaFisica(int id) throws SQLException {
    String sqlPessoaFisica = "DELETE FROM pessoa_fisica WHERE id_pessoa = ?";
    String sqlPessoa = "DELETE FROM Pessoa WHERE id_pessoa = ?";

    try (PreparedStatement preparedStatementPessoaFisica = connection.prepareStatement(sqlPessoaFisica);
        PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa)) {

        // Iniciar uma transação
        connection.setAutoCommit(false);

        try {
            // Excluir da tabela PessoaFisica
            preparedStatementPessoaFisica.setInt(1, id);
            preparedStatementPessoaFisica.executeUpdate();

            // Excluir da tabela Pessoa
            preparedStatementPessoa.setInt(1, id);
            preparedStatementPessoa.executeUpdate();

            // Commit da transação
            connection.commit();
        } catch (SQLException e) {
            // Rollback em caso de exceção
            connection.rollback();
            throw e;
        } finally {
            // Restaurar o modo de autocommit
            connection.setAutoCommit(true);
        }
    }
}

// Trata exceções de SQL, imprimindo o rastreamento de pilha.
private void handleSQLException(SQLException e) {
    e.printStackTrace();
    throw new RuntimeException("Erro na execução da consulta SQL", e);
}
}

```

## PessoaJuridica

```

package cadastrobd.model;
/**
 *
 * @author Andrey H Aires
 */
public class PessoaJuridica extends Pessoa {

    private String cnpj;

    // Construtor padrao
    public PessoaJuridica() {
        super(); // Chama o construtor da classe pai (Pessoa)
    }

    // Construtor completo para PessoaJuridica
    public PessoaJuridica(int id, String nome_pessoa, String logradouro_pessoa, String cidade_pessoa, String estado_pessoa, String telefone_pessoa, String email_pessoa, String cnpj) {
        super(id, nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, telefone_pessoa, email_pessoa);
        this.cnpj = cnpj;
    }

    // Getter e setter para CNPJ
    public String getCnpj() {

```

```

        return cnpj;
    }

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

    // Retorna uma representação de string formatada do objeto.
    @Override
    public String toString() {
        return super.toString() + ", CNPJ: " + cnpj;
    }
}

```

### PessoaJuridicaDAO

```

package cadastrobd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

/**
 * Classe para acessar e manipular dados de Pessoa Jurídica no banco de dados.
 *
 * Autor: Andrey H. Aires
 */
public class PessoaJuridicaDAO {

    private final Connection connection;

    // Construtor que recebe uma conexão como parâmetro.
    public PessoaJuridicaDAO(Connection connection) {
        this.connection = connection;
    }

    // Exibir todos os cadastros de Pessoa Jurídica.
    public List<String> exibirTodasJuridicas() {
        List<String> cadastros = new ArrayList<>();
        String sql = "SELECT p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa, p.cidade_pessoa, p.estado_pessoa, " +
            "p.telefone_pessoa, p.email_pessoa, pj.cnpj " +
            "FROM Pessoa p JOIN pessoa_juridica pj ON p.id_pessoa = pj.id_pessoa";

        try (PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                String cadastro = String.format("ID: %d\nNome: %s\nLogradouro: %s\nCidade: %s\nEstado: %s\n" +
                    "Telefone: %s\nEmail: %s\nCNPJ: %s\n",
                    resultSet.getInt("id_pessoa"),
                    resultSet.getString("nome_pessoa"),
                    resultSet.getString("logradouro_pessoa"),
                    resultSet.getString("cidade_pessoa"),
                    resultSet.getString("estado_pessoa"),
                    resultSet.getString("telefone_pessoa"),
                    resultSet.getString("email_pessoa"),
                    resultSet.getString("cnpj"));

                cadastros.add(cadastro);
            }

        } catch (SQLException e) {
            handleSQLException(e);
        }
    }
}

```

```

    return cadastros;
}

// Verifica se a pessoa com o mesmo ID já existe
public boolean pessoaExiste(int id) throws SQLException {
    String sql = "SELECT id_pessoa FROM Pessoa WHERE id_pessoa = ?";
    try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
        preparedStatement.setInt(1, id);
        try (ResultSet resultSet = preparedStatement.executeQuery()) {
            return resultSet.next();
        }
    }
}

// Incluir um novo cadastro de Pessoa Jurídica no banco de dados.
public void incluirPessoaJuridica(PessoaJuridica pessoaJuridica) throws SQLException {
    String sqlPessoa = "INSERT INTO Pessoa (nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, " +
        "telefone_pessoa, email_pessoa) VALUES (?, ?, ?, ?, ?, ?)";
    String sqlPessoaJuridica = "INSERT INTO pessoa_juridica (id_pessoa, cnpj) VALUES (?, ?)";

    try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa,
        PreparedStatement.RETURN_GENERATED_KEYS);
        PreparedStatement preparedStatementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica)) {

        if (pessoaJuridica.getNome() != null) {
            connection.setAutoCommit(false);

            try {
                preparedStatementPessoa.setString(1, pessoaJuridica.getNome());
                preparedStatementPessoa.setString(2, pessoaJuridica.getLogradouro());
                preparedStatementPessoa.setString(3, pessoaJuridica.getCidade());
                preparedStatementPessoa.setString(4, pessoaJuridica.getEstado());
                preparedStatementPessoa.setString(5, pessoaJuridica.getTelefone());
                preparedStatementPessoa.setString(6, pessoaJuridica.getEmail());
                preparedStatementPessoa.executeUpdate();

                try (ResultSet generatedKeys = preparedStatementPessoa.getGeneratedKeys()) {
                    if (generatedKeys.next()) {
                        int id = generatedKeys.getInt(1);
                        preparedStatementPessoaJuridica.setInt(1, id);
                        preparedStatementPessoaJuridica.setString(2, pessoaJuridica.getCnpj());
                        preparedStatementPessoaJuridica.executeUpdate();
                    }
                }

                connection.commit();
            } catch (SQLException e) {
                connection.rollback();
                throw e;
            } finally {
                connection.setAutoCommit(true);
            }
        } else {
            System.out.println("Nome da pessoa não pode ser nulo. A inserção foi ignorada.");
        }
    } catch (SQLException e) {
        handleSQLException(e);
    }
}

// Altera um registro de Pessoa Jurídica pelo ID no banco de dados.
public void alterarPessoaJuridica(PessoaJuridica pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoa SET nome_pessoa=?, logradouro_pessoa=?, cidade_pessoa=?, estado_pessoa=?,
    telefone_pessoa=?, email_pessoa=? WHERE id_pessoa=?";
    String sqlPessoaJuridica = "UPDATE pessoa_juridica SET cnpj=? WHERE id_pessoa=?";

```

```

try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa);
    PreparedStatement preparedStatementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica)) {

    // Verifica se o nome não é nulo antes de atualizar
    if (pessoa.getNome() != null) {
        // Iniciar uma transação
        connection.setAutoCommit(false);

        try {
            // Atualizar na tabela Pessoa
            preparedStatementPessoa.setString(1, pessoa.getNome());
            preparedStatementPessoa.setString(2, pessoa.getLogradouro());
            preparedStatementPessoa.setString(3, pessoa.getCidade());
            preparedStatementPessoa.setString(4, pessoa.getEstado());
            preparedStatementPessoa.setString(5, pessoa.getTelefone());
            preparedStatementPessoa.setString(6, pessoa.getEmail());
            preparedStatementPessoa.setInt(7, pessoa.getId());
            preparedStatementPessoa.executeUpdate();

            // Atualizar na tabela PessoaJuridica
            preparedStatementPessoaJuridica.setString(1, pessoa.getCnpj());
            preparedStatementPessoaJuridica.setInt(2, pessoa.getId());
            preparedStatementPessoaJuridica.executeUpdate();

            // Commit da transação
            connection.commit();
        } catch (SQLException e) {
            // Rollback em caso de exceção
            connection.rollback();
            throw e;
        } finally {
            // Restaurar o modo de autocommit
            connection.setAutoCommit(true);
        }
    } else {
        System.out.println("Nome da pessoa não pode ser nulo. A atualização foi ignorada.");
    }

} catch (SQLException e) {
    handleSQLException(e);
}

}

// Exclui um registro de Pessoa Jurídica pelo ID no banco de dados.
public void excluirPessoaJuridica(int id) throws SQLException {
    String sqlPessoaJuridica = "DELETE FROM pessoa_juridica WHERE id_pessoa = ?";
    String sqlPessoa = "DELETE FROM Pessoa WHERE id_pessoa = ?";

    try (PreparedStatement preparedStatementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica);
        PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa)) {

        // Iniciar uma transação
        connection.setAutoCommit(false);

        try {
            // Excluir da tabela PessoaJuridica
            preparedStatementPessoaJuridica.setInt(1, id);
            preparedStatementPessoaJuridica.executeUpdate();

            // Excluir da tabela Pessoa
            preparedStatementPessoa.setInt(1, id);
            preparedStatementPessoa.executeUpdate();

            // Commit da transação
            connection.commit();
        } catch (SQLException e) {

```

```

        // Rollback em caso de exceção
        connection.rollback();
        throw e;
    } finally {
        // Restaurar o modo de autocommit
        connection.setAutoCommit(true);
    }
}

// Tratar exceções relacionadas a consultas SQL.
private void handleSQLException(SQLException e) {
    e.printStackTrace();
    throw new RuntimeException("Erro na execução da consulta SQL", e);
}
}

```

### Resultados de saída do sistema:

- Instanciar uma pessoa física e persistir no banco de dados.
- Alterar os dados da pessoa física no banco.
- Consultar todas as pessoas físicas do banco de dados e listar no console.
- Excluir a pessoa física criada anteriormente no banco.

```

run:

*Cadastro de Pessoa Física Realizado com Sucesso*

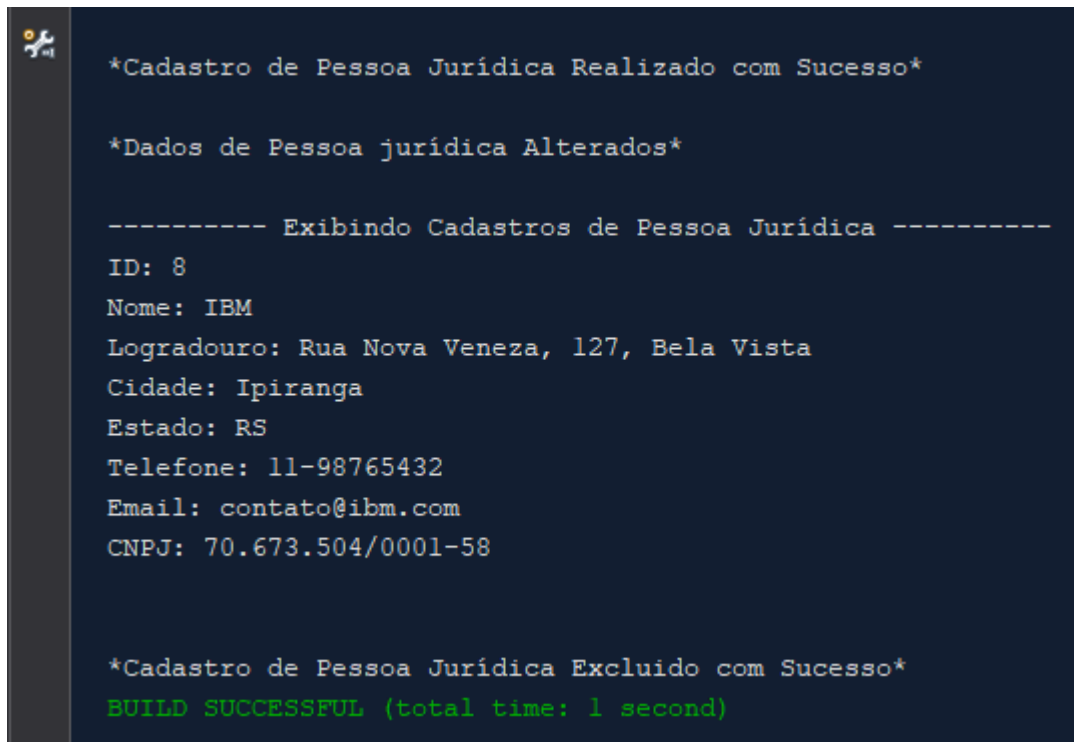
*Dados de Pessoa Física Alterados*

----- Exibindo Cadastros de Pessoa Física -----
ID: 7
Nome: Maria dos Santos
Logradouro: Rua 15 de Setembro, 45, Centro
Cidade: Curitiba
Estado: PR
Telefone: 41-32771598
Email: maria@hotmail.com
CPF: 44254189729

*Cadastro de Pessoa Fisica Excluido com Sucesso*

```

- a) Instanciar uma pessoa jurídica e persistir no banco de dados.
- b) Alterar os dados da pessoa jurídica no banco.
- c) Consultar todas as pessoas jurídicas do banco e listar no console.
- d) Excluir a pessoa jurídica criada anteriormente no banco.



```
*Cadastro de Pessoa Jurídica Realizado com Sucesso*

*Dados de Pessoa jurídica Alterados*

----- Exibindo Cadastros de Pessoa Jurídica -----
ID: 8
Nome: IBM
Logradouro: Rua Nova Veneza, 127, Bela Vista
Cidade: Ipiranga
Estado: RS
Telefone: 11-98765432
Email: contato@ibm.com
CNPJ: 70.673.504/0001-58

*Cadastro de Pessoa Jurídica Excluido com Sucesso*
BUILD SUCCESSFUL (total time: 1 second)
```

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

*O JDBC atua como uma camada de abstração que facilita a interação de aplicações Java com diferentes sistemas de gerenciamento de banco de dados (SGBDs).*

*Sua importância reside na:*

**Portabilidade:** *Permite que o código Java seja consistente em vários SGBDs.*

**Conectividade Dinâmica:** *Facilita a conexão com diferentes ambientes de banco de dados.*

**Gerenciamento de Conexões:** *Controla eficientemente a abertura e fechamento de conexões.*

**Execução de Consultas SQL:** *Possibilita a execução de consultas SQL a partir de código Java.*

**Tratamento de Exceções:** *Lida com exceções relacionadas a operações de banco de dados.*

**Suporte a Transações:** *Permite o tratamento de operações como unidades atômicas.*

*Em resumo, o JDBC simplifica o acesso a bancos de dados, promovendo a portabilidade e flexibilidade em aplicações Java.*



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

***Statement:***

- *Menos seguro contra injeção SQL.*
- *Recompila a consulta a cada execução.*
- *Sintaxe com concatenação direta.*

***PreparedStatement:***

- *Mais seguro contra injeção SQL.*
- *Pré-compila a consulta, melhorando o desempenho.*
- *Sintaxe com espaços reservados ('?').*

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

*O padrão DAO melhora a manutenibilidade do software ao separar o acesso a dados da lógica de negócios, proporcionando abstração do banco de dados, reusabilidade, facilidade em lidar com mudanças estruturais, maior testabilidade e promoção da manutenção evolutiva.*

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

*No modelo estritamente relacional, a herança é geralmente refletida usando uma tabela única para a hierarquia de classes, onde cada linha representa uma instância de uma classe específica e contém colunas para todos os atributos da hierarquia. As colunas não aplicáveis a uma instância específica são preenchidas com valores nulos.*

## 2º Procedimento | Alimentando a Base

Neste 2º procedimento foi implementada a classe *CadastroBD* como main que introduziu o menu interativo, para que o usuário possa incluir, alterar, excluir, buscar por id e exibir todos os registros.

Foi alterado a classe '*Pessoa.java*' para utilizar o método '*toString*' que entrega as consultas de uma maneira personalizada.

Algumas alterações nas classes DAO foram necessárias para que o resultado fosse alcançado.

### Códigos:

#### ConectorBD

```
package cadastrobd;
package cadastrobd;

import cadastro.model.util.ConectorBD;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Scanner;

public class CadastroBD {

    public static void main(String[] args) throws UnsupportedEncodingException {
        System.setOut(new PrintStream(System.out, true, "UTF-8"));
        try (Connection connection = ConectorBD.getConnection()) {
            PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO(connection);
            PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO(connection);

            try (Scanner scanner = new Scanner(System.in)) {
                int opcao;
                do {
                    System.out.println("=====");
                    System.out.println("Escolha uma opção:");
                    System.out.println("1 - Incluir Pessoa");
                    System.out.println("2 - Alterar Pessoa");
                    System.out.println("3 - Excluir Pessoa");
                    System.out.println("4 - Buscar pelo ID");
                    System.out.println("5 - Exibir todos");
                    System.out.println("0 - Finalizar Programa");
                    System.out.println("=====");

                    // Captura da opção escolhida pelo usuário
                    opcao = scanner.nextInt();
                    scanner.nextLine(); // Consumir a quebra de linha

                    // Execução da opção escolhida
                    switch (opcao) {
                        case 1:
                            incluirPessoa(scanner, pessoaFisicaDAO, pessoaJuridicaDAO);
                            break;
                        case 2:
                            alterarPessoa(scanner, pessoaFisicaDAO, pessoaJuridicaDAO);
                            break;
```

```

        case 3:
            excluirPessoa(scanner, pessoaFisicaDAO, pessoaJuridicaDAO);
            break;
        case 4:
            buscarPeloid(scanner, pessoaFisicaDAO, pessoaJuridicaDAO);
            break;
        case 5:
            exibirTodos(pessoaFisicaDAO, pessoaJuridicaDAO);
            break;
        case 0:
            System.out.println("Programa finalizado.");
            scanner.close();
            return;
        default:
            System.out.println("***** Opção inválida. Tente novamente. *****");
            break;
    }
} while (opcao != 0);
} catch (Exception e) {
    e.printStackTrace();
    System.err.println("Erro durante a execução do sistema.");
}
} catch (SQLException e) {
    e.printStackTrace();
    System.err.println("Erro ao conectar ao banco de dados.");
}
}

//Incluir Pessoa
private static void incluirPessoa(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipo = "";
    while (!tipo.equals("F") && !tipo.equals("J")) {
        System.out.println("Escolha o tipo de pessoa:");
        System.out.printf("F - Pessoa Física | J - Pessoa Jurídica: ");
        tipo = scanner.nextLine().toUpperCase();
        if (!tipo.equals("F") && !tipo.equals("J")) {
            System.out.println("Opção inválida. Por favor, escolha F ou J.");
        }
    }

    try {
        if (tipo.equals("F")) {
            // Lógica para incluir Pessoa Física
            System.out.println("Incluir Pessoa Física");

            // Dados para inserção de pessoa física
            PessoaFisica pessoaFisicaParaInserir = new PessoaFisica();
            System.out.print("Nome: ");
            pessoaFisicaParaInserir.setNome(scanner.nextLine());
            System.out.print("Logradouro: ");
            pessoaFisicaParaInserir.setLogradouro(scanner.nextLine());
            System.out.print("Cidade: ");
            pessoaFisicaParaInserir.setCidade(scanner.nextLine());
            System.out.print("Estado: ");
            pessoaFisicaParaInserir.setEstado(scanner.nextLine());
            System.out.print("Telefone: ");
            pessoaFisicaParaInserir.setTelefone(scanner.nextLine());
            System.out.print("E-mail: ");
            pessoaFisicaParaInserir.setEmail(scanner.nextLine());
            System.out.print("CPF: ");
            pessoaFisicaParaInserir.setCpf(scanner.nextLine());

            // Inserir pessoa física
            pessoaFisicaDAO.incluirPessoaFisica(pessoaFisicaParaInserir);
        } else if (tipo.equals("J")) {
            // Lógica para incluir Pessoa Jurídica

```

```

System.out.println("Incluir Pessoa Jurídica");

// Dados para inserção de pessoa jurídica
PessoaJuridica pessoaJuridicaParaInserir = new PessoaJuridica();
System.out.print("Nome: ");
pessoaJuridicaParaInserir.setNome(scanner.nextLine());
System.out.print("Logradouro: ");
pessoaJuridicaParaInserir.setLogradouro(scanner.nextLine());
System.out.print("Cidade: ");
pessoaJuridicaParaInserir.setCidade(scanner.nextLine());
System.out.print("Estado: ");
pessoaJuridicaParaInserir.setEstado(scanner.nextLine());
System.out.print("Telefone: ");
pessoaJuridicaParaInserir.setTelefone(scanner.nextLine());
System.out.print("e-mail: ");
pessoaJuridicaParaInserir.setEmail(scanner.nextLine());
System.out.print("CNPJ: ");
pessoaJuridicaParaInserir.setCnpj(scanner.nextLine());

// Inserir pessoa jurídica
pessoaJuridicaDAO.incluirPessoaJuridica(pessoaJuridicaParaInserir);
}
} catch (SQLException e) {
    e.printStackTrace();
    System.err.println("Erro ao incluir Pessoa no banco de dados.");
}
}

//Alterar Pessoa
private static void alterarPessoa(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipo = "";
    while (!tipo.equals("F") && !tipo.equals("J")) {
        System.out.println("Escolha o tipo de pessoa:");
        System.out.printf("F - Pessoa Física | J - Pessoa Jurídica: ");
        tipo = scanner.nextLine().toUpperCase();
        if (!tipo.equals("F") && !tipo.equals("J")) {
            System.out.println("Opção inválida. Por favor, escolha F ou J.");
        }
    }

    if (tipo.equals("F")) {
        // Método para alterar uma Pessoa Física no banco de dados
        System.out.println("Alterar Pessoa Física");

        System.out.print("ID: ");
        int idParaAlterarPF = scanner.nextInt(); // Defina o ID da pessoa que deseja alterar
        scanner.nextLine(); // Consumir a quebra de linha
        try {
            // Verifica se a pessoa com o ID especificado existe no banco
            if (pessoaFisicaDAO.pessoaExiste(idParaAlterarPF)) {
                // Dados para alteração de pessoa física
                PessoaFisica pessoaFisicaParaAlterar = new PessoaFisica();
                pessoaFisicaParaAlterar.setId(idParaAlterarPF); // Define o ID da pessoa a ser alterada
                System.out.println("Preencha com os novos dados");
                System.out.print("Nome: ");
                pessoaFisicaParaAlterar.setNome(scanner.nextLine());
                System.out.print("Logradouro: ");
                pessoaFisicaParaAlterar.setLogradouro(scanner.nextLine());
                System.out.print("Cidade: ");
                pessoaFisicaParaAlterar.setCidade(scanner.nextLine());
                System.out.print("Estado: ");
                pessoaFisicaParaAlterar.setEstado(scanner.nextLine());
                System.out.print("Telefone: ");
                pessoaFisicaParaAlterar.setTelefone(scanner.nextLine());
                System.out.print("E-mail: ");
                pessoaFisicaParaAlterar.setEmail(scanner.nextLine());
                System.out.print("CPF: ");
            }
        }
    }
}

```

```

        pessoaFisicaParaAlterar.setCpf(scanner.nextLine());

        // Alterar pessoa física no banco
        pessoaFisicaDAO.alterarPessoaFisica(pessoaFisicaParaAlterar);
        System.out.println("\n*Dados Alterados de Pessoa Física ID(" + idParaAlterarPF + ")*");
        System.out.println();

    } else {
        System.out.println("Pessoa com o ID especificado não encontrada.");
    }
} catch (SQLException e) {
    System.err.println("Erro ao alterar Pessoa Física no banco de dados: " + e.getMessage());
}

} else if (tipo.equals("J")) {
    // Método para alterar uma Pessoa Jurídica no banco de dados
    System.out.println("Alterar Pessoa Jurídica");

    System.out.print("ID: ");
    int idParaAlterarPJ = scanner.nextInt(); // Defina o ID da pessoa que deseja alterar
    scanner.nextLine(); // Consumir a quebra de linha
    try {
        // Verifica se a pessoa com o ID especificado existe no banco
        if (pessoaFisicaDAO.pessoaExiste(idParaAlterarPJ)) {
            // Dados para alteração de pessoa jurídica
            PessoaJuridica pessoaJuridicaParaAlterar = new PessoaJuridica();
            pessoaJuridicaParaAlterar.setId(idParaAlterarPJ); // Define o ID da pessoa a ser alterada
            System.out.println("Preencha com os novos dados");
            System.out.print("Nome: ");
            pessoaJuridicaParaAlterar.setNome(scanner.nextLine());
            System.out.print("Logradouro: ");
            pessoaJuridicaParaAlterar.setLogradouro(scanner.nextLine());
            System.out.print("Cidade: ");
            pessoaJuridicaParaAlterar.setCidade(scanner.nextLine());
            System.out.print("Estado: ");
            pessoaJuridicaParaAlterar.setEstado(scanner.nextLine());
            System.out.print("Telefone: ");
            pessoaJuridicaParaAlterar.setTelefone(scanner.nextLine());
            System.out.print("E-mail: ");
            pessoaJuridicaParaAlterar.setEmail(scanner.nextLine());
            System.out.print("CNPJ: ");
            pessoaJuridicaParaAlterar.setCnpj(scanner.nextLine());

            // Alterar pessoa jurídica no banco
            pessoaJuridicaDAO.alterarPessoaJuridica(pessoaJuridicaParaAlterar);
            System.out.println("\n*Dados Alterados de Pessoa Física ID(" + idParaAlterarPJ + ")*");
            System.out.println();

        } else {
            System.out.println("Pessoa com o ID especificado não encontrada.");
        }
    } catch (SQLException e) {
        System.err.println("Erro ao alterar Pessoa Física no banco de dados: " + e.getMessage());
    }

}

}

//Excluir Pessoa
private static void excluirPessoa(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipo = "";
    while (!tipo.equals("F") && !tipo.equals("J")) {
        System.out.println("Escolha o tipo de pessoa:");
        System.out.printf("F - Pessoa Física | J - Pessoa Jurídica: ");
        tipo = scanner.nextLine().toUpperCase();
        if (!tipo.equals("F") && !tipo.equals("J")) {
            System.out.println("Opção inválida. Por favor, escolha F ou J.");
        }
    }
}

```

```

    }
}

if (tipo.equals("F")) {
    // Método para excluir uma Pessoa Física no banco de dados
    System.out.println("Excluir Pessoa Física");

    System.out.print("ID: ");
    int idParaExcluirPF = scanner.nextInt(); // Defina o ID da pessoa que deseja excluir
    scanner.nextLine(); // Consumir a quebra de linha
    try {
        // Verifica se a pessoa com o ID especificado existe no banco
        if (pessoaFisicaDAO.pessoaExiste(idParaExcluirPF)) {
            pessoaFisicaDAO.excluirPessoaFisica(idParaExcluirPF);
            System.out.println("\n*Cadastro de Pessoa Fisica Excluido com Sucesso*");

        } else {
            System.out.println("Pessoa com o ID especificado não encontrada.");
        }
    } catch (SQLException e) {
        System.err.println("Erro ao excluir Pessoa Física no banco de dados: " + e.getMessage());
    }

} else if (tipo.equals("J")) {
    // Método para alterar uma Pessoa Jurídica no banco de dados
    System.out.println("Excluir Pessoa Jurídica");

    System.out.print("ID: ");
    int idParaExcluirPJ = scanner.nextInt(); // Defina o ID da pessoa que deseja excluir
    scanner.nextLine(); // Consumir a quebra de linha
    try {
        // Verifica se a pessoa com o ID especificado existe no banco
        if (pessoaJuridicaDAO.pessoaExiste(idParaExcluirPJ)) {
            pessoaJuridicaDAO.excluirPessoaJuridica(idParaExcluirPJ);
            System.out.println("\n*Cadastro de Pessoa Jurídica Excluido com Sucesso*");
        } else {
            System.out.println("Pessoa com o ID especificado não encontrada.");
        }
    } catch (SQLException e) {
        System.err.println("Erro ao excluir Pessoa Jurídica no banco de dados: " + e.getMessage());
    }

}

}

//Buscar por ID
private static void buscarPeloid(Scanner scanner, PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) {
    String tipo = "";
    while (!tipo.equals("F") && !tipo.equals("J")) {
        System.out.println("Escolha o tipo de pessoa:");
        System.out.printf("F - Pessoa Física | J - Pessoa Jurídica: ");
        tipo = scanner.nextLine().toUpperCase();
        if (!tipo.equals("F") && !tipo.equals("J")) {
            System.out.println("Opção inválida. Por favor, escolha F ou J.");
        }
    }

    System.out.print("Digite o ID da pessoa que deseja procurar: ");
    int buscarPeloid = scanner.nextInt(); // Defina o ID da pessoa que deseja buscar
    scanner.nextLine(); // Consumir a quebra de linha

    try {
        if (tipo.equals("F")) {
            PessoaFisica pessoaFisicaEncontrada = pessoaFisicaDAO.buscarPessoaFisicaPorId(buscarPeloid);
            if (pessoaFisicaEncontrada != null) {
                System.out.println("\n----- Detalhes da Pessoa Física -----");
                System.out.println(pessoaFisicaEncontrada);
            }
        }
    }
}

```

```

    } else {
        System.out.println("Pessoa Física com o ID especificado não encontrada.");
    }
} else if (tipo.equals("J")) {
    PessoaJuridica pessoaJuridicaEncontrada = pessoaJuridicaDAO.buscarPessoaJuridicaPorId(buscarPeloid);
    if (pessoaJuridicaEncontrada != null) {
        System.out.println("\n----- Detalhes da Pessoa Jurídica -----");
        System.out.println(pessoaJuridicaEncontrada);
    } else {
        System.out.println("Pessoa Jurídica com o ID especificado não encontrada.");
    }
} else {
    System.out.println("Opção inválida.");
}
} catch (Exception e) {
    System.err.println("Erro ao buscar pelo ID: " + e.getMessage());
}
}

// Exibir todos os cadastros
private static void exibirTodos(PessoaFisicaDAO pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) {
    System.out.println("----- Cadastros de Pessoa Física -----");
    pessoaFisicaDAO.exibirTodosFisica().forEach(System.out::println);
    // Exibir todos os cadastros de pessoa jurídica
    System.out.println("\n----- Cadastros de Pessoa Jurídica -----");
    pessoaJuridicaDAO.exibirTodasJuridicas().forEach(System.out::println);
}
}

```

## Pessoa

```

package cadastrobd.model;
/**
 *
 * @author Andrey H Aires
 */
public class Pessoa {
    // Campos da classe
    private int id_pessoa;
    private String nome_pessoa;
    private String logradouro_pessoa;
    private String cidade_pessoa;
    private String estado_pessoa;
    private String telefone_pessoa;
    private String email_pessoa;

    // Construtor padrao
    public Pessoa() {
    }

    // Construtor completo
    public Pessoa(int id_pessoa, String nome_pessoa, String logradouro_pessoa, String cidade_pessoa, String estado_pessoa, String telefone_pessoa, String email_pessoa) {
        this.id_pessoa = id_pessoa;
        this.nome_pessoa = nome_pessoa;
        this.logradouro_pessoa = logradouro_pessoa;
        this.cidade_pessoa = cidade_pessoa;
        this.estado_pessoa = estado_pessoa;
        this.telefone_pessoa = telefone_pessoa;
        this.email_pessoa = email_pessoa;
    }

    // Metodos getters
    public int getId() {
        return id_pessoa;
    }
}

```

```
public String getNome() {
    return nome_pessoa;
}

public String getLogradouro() {
    return logradouro_pessoa;
}

public String getCidade() {
    return cidade_pessoa;
}

public String getEstado() {
    return estado_pessoa;
}

public String getTelefone() {
    return telefone_pessoa;
}

public String getEmail() {
    return email_pessoa;
}

//Metodo setters
public void setId(int id) {
    this.id_pessoa = id;
}

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

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

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

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

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

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

// Retorna uma representação de string formatada do objeto.
@Override
public String toString() {
    return String.format("ID: %d\nNome: %s\nLogradouro: %s\nCidade: %s\nEstado: %s\nTelefone: %s\nEmail: %s\n",
        getId(),
        getNome(),
        getLogradouro(),
        getCidade(),
        getEstado(),
        getTelefone(),
        getEmail());
}
```



## PessoaFisica

```
package cadastrobd.model;

/**
 *
 * author Andrey H Aires
 */
public class PessoaFisica extends Pessoa {

    private String cpf;

    // Construtor padrão
    public PessoaFisica() {
        super();
    }

    // Construtor completo
    public PessoaFisica(int id_pessoa, String nome_pessoa, String logradouro_pessoa, String cidade_pessoa, String estado_pessoa,
String telefone_pessoa, String email_pessoa, String cpf) {
        super(id_pessoa, nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, telefone_pessoa, email_pessoa);
        this.cpf = cpf;
    }

    // Getter e setter
    public String getCpf() {
        return cpf;
    }

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

    // Retorna uma representação de string formatada do objeto.
    @Override
    public String toString() {
        return super.toString() + "CPF: " + getCpf();
    }
}
```

## PessoaFisicaDAO

```
package cadastrobd.model;

/**
 * Classe que interage com o banco de dados para operações relacionadas à Pessoa Física.
 * Fornece métodos para exibir todos os registros de Pessoa Física e incluir novos registros.
 * Utiliza a tabela Pessoa para dados gerais e a tabela pessoa_fisica para dados específicos de Pessoa Física.
 * Cada registro é formatado como uma string para fácil exibição.
 *
 * @author Andrey H Aires
 */
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 final Connection connection;

    // Construtor da classe PessoaFisicaDAO.
    public PessoaFisicaDAO(Connection connection) { // Conexão com o banco de dados.
```

```

        this.connection = connection;
    }

    // Obtém todos os registros de Pessoa Física do banco de dados e os retorna como uma lista de strings formatadas.
    public List<String> exibirTodosFisica() {
        List<String> cadastros = new ArrayList<>();
        String sql = "SELECT p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa, p.cidade_pessoa, p.estado_pessoa,
p.telefone_pessoa, p.email_pessoa, pf.cpf " +
            "FROM Pessoa p " +
            "JOIN pessoa_fisica pf ON p.id_pessoa = pf.id_pessoa";

        try (PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                String cadastro = String.format("ID: %d\nNome: %s\nLogradouro: %s\nCidade: %s\nEstado: %s\nTelefone:
%s\nEmail: %s\nCPF: %s\n",
                    resultSet.getInt("id_pessoa"),
                    resultSet.getString("nome_pessoa"),
                    resultSet.getString("logradouro_pessoa"),
                    resultSet.getString("cidade_pessoa"),
                    resultSet.getString("estado_pessoa"),
                    resultSet.getString("telefone_pessoa"),
                    resultSet.getString("email_pessoa"),
                    resultSet.getString("cpf"));

                cadastros.add(cadastro);
            }

        } catch (SQLException e) {
            handleSQLException(e);
        }

        return cadastros;
    }

    // Verifica se a pessoa com o mesmo ID já existe
    public boolean pessoaExiste(int id) throws SQLException {
        String sql = "SELECT id_pessoa FROM Pessoa WHERE id_pessoa = ?";
        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            preparedStatement.setInt(1, id);
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                return resultSet.next();
            }
        }
    }

    // Inclui um novo registro de Pessoa Física no banco de dados.
    public void incluirPessoaFisica(PessoaFisica pessoaFisica) throws SQLException {
        String sqlPessoa = "INSERT INTO Pessoa (nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, telefone_pessoa,
email_pessoa) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlPessoaFisica = "INSERT INTO pessoa_fisica (id_pessoa, cpf) VALUES (?, ?)";

        try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa,
PreparedStatement.RETURN_GENERATED_KEYS);
            PreparedStatement preparedStatementPessoaFisica = connection.prepareStatement(sqlPessoaFisica)) {

            // Verifica se o nome não é nulo antes de inserir
            if (pessoaFisica.getNome() != null) {
                // Iniciar uma transação
                connection.setAutoCommit(false);

                try {
                    // Inserir na tabela Pessoa
                    preparedStatementPessoa.setString(1, pessoaFisica.getNome());
                    preparedStatementPessoa.setString(2, pessoaFisica.getLogradouro());
                    preparedStatementPessoa.setString(3, pessoaFisica.getCidade());

```

```

        preparedStatementPessoa.setString(4, pessoaFisica.getEstado());
        preparedStatementPessoa.setString(5, pessoaFisica.getTelefone());
        preparedStatementPessoa.setString(6, pessoaFisica.getEmail());
        preparedStatementPessoa.executeUpdate();

        // Obter o ID gerado
        try (ResultSet generatedKeys = preparedStatementPessoa.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                int id = generatedKeys.getInt(1);

                // Inserir na tabela PessoaFisica
                preparedStatementPessoaFisica.setInt(1, id);
                preparedStatementPessoaFisica.setString(2, pessoaFisica.getCpf());
                preparedStatementPessoaFisica.executeUpdate();
            }
        }

        // Commit da transação
        connection.commit();
    } catch (SQLException e) {
        // Rollback em caso de exceção
        connection.rollback();
        throw e;
    } finally {
        // Restaurar o modo de autocommit
        connection.setAutoCommit(true);
    }
} else {
    System.out.println("Nome da pessoa não pode ser nulo. A inserção foi ignorada.");
}

} catch (SQLException e) {
    handleSQLException(e);
}
}

// Altera um registro de Pessoa Física pelo ID no banco de dados.
public void alterarPessoaFisica(PessoaFisica pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoa SET nome_pessoa=?, logradouro_pessoa=?, cidade_pessoa=?, estado_pessoa=?,
telefone_pessoa=?, email_pessoa=? WHERE id_pessoa=?";
    String sqlPessoaFisica = "UPDATE pessoa_fisica SET cpf=? WHERE id_pessoa=?";

    try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa);
        PreparedStatement preparedStatementPessoaFisica = connection.prepareStatement(sqlPessoaFisica)) {

        // Verifica se o nome não é nulo antes de atualizar
        if (pessoa.getNome() != null) {
            // Iniciar uma transação
            connection.setAutoCommit(false);

            try {
                // Atualizar na tabela Pessoa
                preparedStatementPessoa.setString(1, pessoa.getNome());
                preparedStatementPessoa.setString(2, pessoa.getLogradouro());
                preparedStatementPessoa.setString(3, pessoa.getCidade());
                preparedStatementPessoa.setString(4, pessoa.getEstado());
                preparedStatementPessoa.setString(5, pessoa.getTelefone());
                preparedStatementPessoa.setString(6, pessoa.getEmail());
                preparedStatementPessoa.setInt(7, pessoa.getId());
                preparedStatementPessoa.executeUpdate();

                // Atualizar na tabela PessoaFisica
                preparedStatementPessoaFisica.setString(1, pessoa.getCpf());
                preparedStatementPessoaFisica.setInt(2, pessoa.getId());
                preparedStatementPessoaFisica.executeUpdate();

                // Commit da transação
            }
        }
    }
}

```

```

        connection.commit();
    } catch (SQLException e) {
        // Rollback em caso de exceção
        connection.rollback();
        throw e;
    } finally {
        // Restaurar o modo de autocommit
        connection.setAutoCommit(true);
    }
} else {
    System.out.println("Nome da pessoa não pode ser nulo. A atualização foi ignorada.");
}

```

```

} catch (SQLException e) {
    handleSQLException(e);
}
}

```

// Exclui um registro de Pessoa Física pelo ID no banco de dados.

```

public void excluirPessoaFisica(int id) throws SQLException {
    String sqlPessoaFisica = "DELETE FROM pessoa_fisica WHERE id_pessoa = ?";
    String sqlPessoa = "DELETE FROM Pessoa WHERE id_pessoa = ?";

```

```

    try (PreparedStatement preparedStatementPessoaFisica = connection.prepareStatement(sqlPessoaFisica);
        PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa)) {

```

```

        // Iniciar uma transação
        connection.setAutoCommit(false);

```

```

        try {
            // Excluir da tabela PessoaFisica
            preparedStatementPessoaFisica.setInt(1, id);
            preparedStatementPessoaFisica.executeUpdate();

```

```

            // Excluir da tabela Pessoa
            preparedStatementPessoa.setInt(1, id);
            preparedStatementPessoa.executeUpdate();

```

```

            // Commit da transação
            connection.commit();
        } catch (SQLException e) {
            // Rollback em caso de exceção
            connection.rollback();
            throw e;
        } finally {
            // Restaurar o modo de autocommit
            connection.setAutoCommit(true);
        }
    }
}

```

// Faz a busca de Pessoa Física por ID.

```

public PessoaFisica buscarPessoaFisicaPorId(int id) {
    String sql = "SELECT p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa, p.cidade_pessoa, p.estado_pessoa, p.telefone_pessoa,
p.email_pessoa, pf.cpf " +
        "FROM Pessoa p " +
        "JOIN pessoa_fisica pf ON p.id_pessoa = pf.id_pessoa " +
        "WHERE p.id_pessoa = ?";

```

```

    try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
        preparedStatement.setInt(1, id);

```

```

    try (ResultSet resultSet = preparedStatement.executeQuery()) {
        if (resultSet.next()) {
            PessoaFisica pessoaFisica = new PessoaFisica();
            pessoaFisica.setId(resultSet.getInt("id_pessoa"));

```

```

        pessoaFisica.setNome(resultSet.getString("nome_pessoa"));
        pessoaFisica.setLogradouro(resultSet.getString("logradouro_pessoa"));
        pessoaFisica.setCidade(resultSet.getString("cidade_pessoa"));
        pessoaFisica.setEstado(resultSet.getString("estado_pessoa"));
        pessoaFisica.setTelefone(resultSet.getString("telefone_pessoa"));
        pessoaFisica.setEmail(resultSet.getString("email_pessoa"));
        pessoaFisica.setCpf(resultSet.getString("cpf"));

        return pessoaFisica;
    }
} catch (SQLException e) {
    handleSQLException(e);
}

return null; // Retorna null se não encontrar a pessoa com o ID especificado
}

// Trata exceções de SQL, imprimindo o rastreamento de pilha.
private void handleSQLException(SQLException e) {
    e.printStackTrace();
    throw new RuntimeException("Erro na execução da consulta SQL", e);
}
}

```

### PessoaJuridica

```

package cadastrobd.model;

/**
 *
 * @author Andrey H Aires
 */
public class PessoaJuridica extends Pessoa {

    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {
        super(); // Chama o construtor da classe pai (Pessoa)
    }

    // Construtor completo para PessoaJuridica
    public PessoaJuridica(int id, String nome_pessoa, String logradouro_pessoa, String cidade_pessoa, String estado_pessoa, String
    telefone_pessoa, String email_pessoa, String cnpj) {
        super(id, nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, telefone_pessoa, email_pessoa);
        this.cnpj = cnpj;
    }

    // Getter e setter para CNPJ
    public String getCnpj() {
        return cnpj;
    }

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

    // Retorna uma representação de string formatada do objeto.
    @Override
    public String toString() {
        return super.toString() + "CNPJ: " + cnpj;
    }
}

```

## PessoaJuridicaDAO

```
package cadastrobd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

/**
 * Classe para acessar e manipular dados de Pessoa Jurídica no banco de dados.
 *
 * Autor: Andrey H. Aires
 */
public class PessoaJuridicaDAO {

    private final Connection connection;

    // Construtor que recebe uma conexão como parâmetro.
    public PessoaJuridicaDAO(Connection connection) {
        this.connection = connection;
    }

    // Exibir todos os cadastros de Pessoa Jurídica.
    public List<String> exibirTodasJuridicas() {
        List<String> cadastros = new ArrayList<>();
        String sql = "SELECT p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa, p.cidade_pessoa, p.estado_pessoa, " +
            "p.telefone_pessoa, p.email_pessoa, pj.cnpj " +
            "FROM Pessoa p JOIN pessoa_juridica pj ON p.id_pessoa = pj.id_pessoa";

        try (PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                String cadastro = String.format("ID: %d\nNome: %s\nLogradouro: %s\nCidade: %s\nEstado: %s\n" +
                    "Telefone: %s\nEmail: %s\nCNPJ: %s\n",
                    resultSet.getInt("id_pessoa"),
                    resultSet.getString("nome_pessoa"),
                    resultSet.getString("logradouro_pessoa"),
                    resultSet.getString("cidade_pessoa"),
                    resultSet.getString("estado_pessoa"),
                    resultSet.getString("telefone_pessoa"),
                    resultSet.getString("email_pessoa"),
                    resultSet.getString("cnpj"));

                cadastros.add(cadastro);
            }

        } catch (SQLException e) {
            handleSQLException(e);
        }

        return cadastros;
    }

    // Verifica se a pessoa com o mesmo ID já existe
    public boolean pessoaExiste(int id) throws SQLException {
        String sql = "SELECT id_pessoa FROM Pessoa WHERE id_pessoa = ?";
        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            preparedStatement.setInt(1, id);
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                return resultSet.next();
            }
        }
    }
}
```

```

// Incluir um novo cadastro de Pessoa Jurídica no banco de dados.
public void incluirPessoaJuridica(PessoaJuridica pessoaJuridica) throws SQLException {
    String sqlPessoa = "INSERT INTO Pessoa (nome_pessoa, logradouro_pessoa, cidade_pessoa, estado_pessoa, " +
        "telefone_pessoa, email_pessoa) VALUES (?, ?, ?, ?, ?, ?)";
    String sqlPessoaJuridica = "INSERT INTO pessoa_juridica (id_pessoa, cnpj) VALUES (?, ?)";

    try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa,
        PreparedStatement.RETURN_GENERATED_KEYS);
        PreparedStatement preparedStatementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica)) {

        if (pessoaJuridica.getNome() != null) {
            connection.setAutoCommit(false);

            try {
                preparedStatementPessoa.setString(1, pessoaJuridica.getNome());
                preparedStatementPessoa.setString(2, pessoaJuridica.getLogradouro());
                preparedStatementPessoa.setString(3, pessoaJuridica.getCidade());
                preparedStatementPessoa.setString(4, pessoaJuridica.getEstado());
                preparedStatementPessoa.setString(5, pessoaJuridica.getTelefone());
                preparedStatementPessoa.setString(6, pessoaJuridica.getEmail());
                preparedStatementPessoa.executeUpdate();

                try (ResultSet generatedKeys = preparedStatementPessoa.getGeneratedKeys()) {
                    if (generatedKeys.next()) {
                        int id = generatedKeys.getInt(1);
                        preparedStatementPessoaJuridica.setInt(1, id);
                        preparedStatementPessoaJuridica.setString(2, pessoaJuridica.getCnpj());
                        preparedStatementPessoaJuridica.executeUpdate();
                    }
                }

                connection.commit();
            } catch (SQLException e) {
                connection.rollback();
                throw e;
            } finally {
                connection.setAutoCommit(true);
            }
        } else {
            System.out.println("Nome da pessoa não pode ser nulo. A inserção foi ignorada.");
        }

    } catch (SQLException e) {
        handleSQLException(e);
    }
}

// Altera um registro de Pessoa Jurídica pelo ID no banco de dados.
public void alterarPessoaJuridica(PessoaJuridica pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoa SET nome_pessoa=?, logradouro_pessoa=?, cidade_pessoa=?, estado_pessoa=?,
        telefone_pessoa=?, email_pessoa=? WHERE id_pessoa=?";
    String sqlPessoaJuridica = "UPDATE pessoa_juridica SET cnpj=? WHERE id_pessoa=?";

    try (PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa);
        PreparedStatement preparedStatementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica)) {

        // Verifica se o nome não é nulo antes de atualizar
        if (pessoa.getNome() != null) {
            // Iniciar uma transação
            connection.setAutoCommit(false);

            try {
                // Atualizar na tabela Pessoa
                preparedStatementPessoa.setString(1, pessoa.getNome());
                preparedStatementPessoa.setString(2, pessoa.getLogradouro());
                preparedStatementPessoa.setString(3, pessoa.getCidade());
                preparedStatementPessoa.setString(4, pessoa.getEstado());
            }
        }
    }
}

```

```

        preparedStatementPessoa.setString(5, pessoa.getTelefone());
        preparedStatementPessoa.setString(6, pessoa.getEmail());
        preparedStatementPessoa.setInt(7, pessoa.getId());
        preparedStatementPessoa.executeUpdate();

        // Atualizar na tabela PessoaJuridica
        preparedStatementPessoaJuridica.setString(1, pessoa.getCnpj());
        preparedStatementPessoaJuridica.setInt(2, pessoa.getId());
        preparedStatementPessoaJuridica.executeUpdate();

        // Commit da transação
        connection.commit();
    } catch (SQLException e) {
        // Rollback em caso de exceção
        connection.rollback();
        throw e;
    } finally {
        // Restaurar o modo de autocommit
        connection.setAutoCommit(true);
    }
} else {
    System.out.println("Nome da pessoa não pode ser nulo. A atualização foi ignorada.");
}

} catch (SQLException e) {
    handleSQLException(e);
}
}

// Exclui um registro de Pessoa Jurídica pelo ID no banco de dados.
public void excluirPessoaJuridica(int id) throws SQLException {
    String sqlPessoaJuridica = "DELETE FROM pessoa_juridica WHERE id_pessoa = ?";
    String sqlPessoa = "DELETE FROM Pessoa WHERE id_pessoa = ?";

    try (PreparedStatement preparedStatementPessoaJuridica = connection.prepareStatement(sqlPessoaJuridica);
        PreparedStatement preparedStatementPessoa = connection.prepareStatement(sqlPessoa)) {

        // Iniciar uma transação
        connection.setAutoCommit(false);

        try {
            // Excluir da tabela PessoaJuridica
            preparedStatementPessoaJuridica.setInt(1, id);
            preparedStatementPessoaJuridica.executeUpdate();

            // Excluir da tabela Pessoa
            preparedStatementPessoa.setInt(1, id);
            preparedStatementPessoa.executeUpdate();

            // Commit da transação
            connection.commit();
        } catch (SQLException e) {
            // Rollback em caso de exceção
            connection.rollback();
            throw e;
        } finally {
            // Restaurar o modo de autocommit
            connection.setAutoCommit(true);
        }
    }
}

// Busca um registro de pessoa Jurídica pelo ID.
// Faz a busca de Pessoa Jurídica por ID.
public PessoaJuridica buscarPessoaJuridicaPorId(int id) {
    String sql = "SELECT p.id_pessoa, p.nome_pessoa, p.logradouro_pessoa, p.cidade_pessoa, p.estado_pessoa, p.telefone_pessoa, p.email_pessoa, pj.cnpj " +

```



```

"FROM Pessoa p " +
"JOIN pessoa_juridica pj ON p.id_pessoa = pj.id_pessoa " +
"WHERE p.id_pessoa = ?";

try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
    preparedStatement.setInt(1, id);

    try (ResultSet resultSet = preparedStatement.executeQuery()) {
        if (resultSet.next()) {
            PessoaJuridica pessoaJuridica = new PessoaJuridica();
            pessoaJuridica.setId(resultSet.getInt("id_pessoa"));
            pessoaJuridica.setNome(resultSet.getString("nome_pessoa"));
            pessoaJuridica.setLogradouro(resultSet.getString("logradouro_pessoa"));
            pessoaJuridica.setCidade(resultSet.getString("cidade_pessoa"));
            pessoaJuridica.setEstado(resultSet.getString("estado_pessoa"));
            pessoaJuridica.setTelefone(resultSet.getString("telefone_pessoa"));
            pessoaJuridica.setEmail(resultSet.getString("email_pessoa"));
            pessoaJuridica.setCnpj(resultSet.getString("cnpj"));

            return pessoaJuridica;
        }
    }
} catch (SQLException e) {
    handleSQLException(e);
}

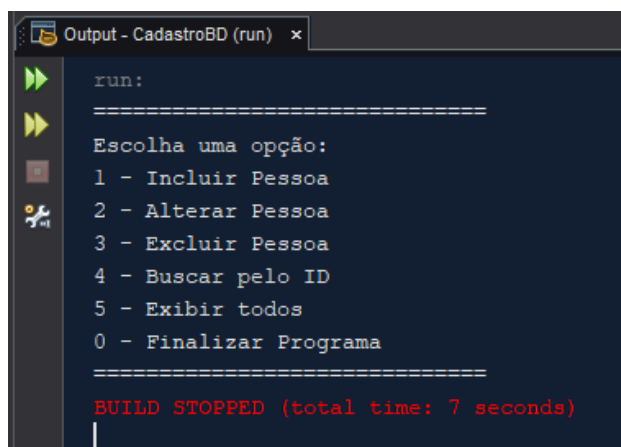
return null; // Retorna null se não encontrar a pessoa com o ID especificado
}

// Tratar exceções relacionadas a consultas SQL.
private void handleSQLException(SQLException e) {
    e.printStackTrace();
    throw new RuntimeException("Erro na execução da consulta SQL", e);
}
}

```

## Resultados de saída do sistema:

- Efetuar as diversas operações disponibilizadas, tanto para pessoa jurídica quanto para pessoa física.
- Feitas as operações, verificar os dados no SQL Server, com a utilização da aba Services, divisão Databases, do NetBeans, ou através do SQL Server Management Studio.



```

Output - CadastroBD (run) x
run:
=====
Escolha uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar Programa
=====
BUILD STOPPED (total time: 7 seconds)

```

### 1º Cadastrando Pessoa Física

```
=====
1
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: f
Incluir Pessoa Física
Nome: João dos Santos
Logradouro: Rua das Arvores, 74
Cidade: Curitiba
Estado: PR
Telefone: 41-32271520
E-mail: joao@gmail.com
CPF: 43548020070
=====
```

### 2º Consultando dados

```
=====
5
----- Cadastros de Pessoa Física -----
ID: 7
Nome: João dos Santos
Logradouro: Rua das Arvores, 74
Cidade: Curitiba
Estado: PR
Telefone: 41-32271520
Email: joao@gmail.com
CPF: 43548020070
```

```
----- Cadastros de Pessoa Jurídica -----
=====
```

### 3º Alterando cadastro Pessoa Física

```
=====
2
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: f
Alterar Pessoa Física
ID: 7
Preencha com os novos dados
Nome: José da Silva
Logradouro: Rua das Ameixas
Cidade: Florianópolis
Estado: SC
Telefone: 48-32234030
E-mail: jose@gmail.com
CPF: 91867055120
```

\*Dados Alterados de Pessoa Física ID(7)\*

### 4º Cadastro outra Pessoa Física

```
=====
1
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: f
Incluir Pessoa Física
Nome: Maria Soares
Logradouro: Rua da independencia, 42
Cidade: São José
Estado: SC
Telefone: 48-91708090
E-mail: mariasoaresh@gmail.com
CPF: 54131755157
=====
```

### 5º Consultando dados

```
=====
5
----- Cadastros de Pessoa Física -----
ID: 7
Nome: José da Silva
Logradouro: Rua das Ameixas
Cidade: Florianópolis
Estado: SC
Telefone: 48-32234030
Email: jose@gmail.com
CPF: 91867055120
```

```
ID: 8
Nome: Maria Soares
Logradouro: Rua da independencia, 42
Cidade: São José
Estado: SC
Telefone: 48-91708090
Email: mariasoaresh@gmail.com
CPF: 54131755157
```

```
----- Cadastros de Pessoa Jurídica -----
=====
```

### 6º Buscando dados por ID

```
=====
4
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: f
Digite o ID da pessoa que deseja procurar: 7

----- Detalhes da Pessoa Física -----
ID: 7
Nome: José da Silva
Logradouro: Rua das Ameixas
Cidade: Florianópolis
Estado: SC
Telefone: 48-32234030
Email: jose@gmail.com
CPF: 91867055120
=====
```

### 7º Cadastrando Pessoa Juridica

```
=====
1
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: j
Incluir Pessoa Jurídica
Nome: Intelbras
Logradouro: Rua 13 de Maio, 45
Cidade: São José
Estado: SC
Telefone: 48-32422730
e-mail: intelbras@intelbras.com
CNPJ: 43.829.310/0001-26
=====
```

### 8º Cadastrando outra Pessoa Juridica

```
=====
1
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: j
Incluir Pessoa Jurídica
Nome: Lojas Americanas
Logradouro: Av. 12 de Junho, 103
Cidade: Florianópolis
Estado: SC
Telefone: 48-32702515
e-mail: contato@americanas.com.br
CNPJ: 01.323.609/0001-64
=====
```

### 9º Consultando dados

```
----- Cadastros de Pessoa Jurídica -----
ID: 9
Nome: Intelbras
Logradouro: Rua 13 de Maio, 45
Cidade: São José
Estado: SC
Telefone: 48-32422730
Email: intelbras@intelbras.com
CNPJ: 43.829.310/0001-26

ID: 10
Nome: Lojas Americanas
Logradouro: Av. 12 de Junho, 103
Cidade: Florianópolis
Estado: SC
Telefone: 48-32702515
Email: contato@americanas.com.br
CNPJ: 01.323.609/0001-64
=====
```

### 10º Alterando dados Pessoa Jurídica

```
=====
2
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: j
Alterar Pessoa Jurídica
ID: 10
Preencha com os novos dados
Nome: Mercado Livre
Logradouro: Geral dos Campos, 63
Cidade: Curitiba
Estado: PR
Telefone: 41-32412313
E-mail: ml@mercadolivre.com.br
CNPJ: 57.040.264/0001-96
```

\*Dados Alterados de Pessoa Física ID(10)\*

### 11º Consultando o banco pelo SQL Server

PROCEDIMENTO2 - CO...S.Loja (loja (60)) CLEAR.sql - DESKTOP...SS.Loja (loja (54))

```
19 p.cidade_pessoa, p.estado_pessoa, p.telefone_pessoa,
20 p.email_pessoa, pf.id_pessoa, pf.cpf
21 FROM
```

100 %

Resultados Mensagens

	id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa	id_pessoa	cpf
1	7	José da Silva	Rua das Ameixas	Florianópolis	SC	48-32234030	jose@gmail.com	7	91867055120
2	8	Maria Soares	Rua da independencia, 42	São José	SC	48-91708090	mariasoaress@gmail.com	8	54131755157

	id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa	id_pessoa	cnnpj
1	9	Intelbras	Rua 13 de Maio, 45	São José	SC	48-32422730	intelbras@intelbras.com	9	43.829.310/0001-26
2	10	Mercado Livre	Geral dos Campos, 63	Curitiba	PR	41-32412313	ml@mercadolivre.com.br	10	57.040.264/0001-96

### 12° Excluindo dados P. Fisica

```
=====
3
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: f
Excluir Pessoa Física
ID: 7

*Cadastro de Pessoa Fisica Excluido com Sucesso*
=====
```

### 13° Excluindo dados P. Juridica

```
=====
3
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: j
Excluir Pessoa Juridica
ID: 9

*Cadastro de Pessoa Juridica Excluido com Sucesso*
=====
```

### 14° Consultando dados

```
=====
5
----- Cadastros de Pessoa Fisica -----
ID: 8
Nome: Maria Soares
Logradouro: Rua da independencia, 42
Cidade: São José
Estado: SC
Telefone: 48-91708090
Email: mariasoares@gmail.com
CPF: 54131755157

----- Cadastros de Pessoa Juridica -----
ID: 10
Nome: Mercado Livre
Logradouro: Geral dos Campos, 63
Cidade: Curitiba
Estado: PR
Telefone: 41-32412313
Email: ml@mercadolivre.com.br
CNPJ: 57.040.264/0001-96

=====
```

### 15° Consultando dados no SQL Server

PROCEDIMENTO2 - CO...S.Loja (loja (60)) - X CLEAR.sql - DESKTOP...SS.Loja (loja (54))

```
19 p.cidade_pessoa, p.estado_pessoa, p.telefone_pessoa,
20 p.email_pessoa, pf.id_pessoa, pf.cpf
21 FROM
```

100 %

	id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa	id_pessoa	cpf
1	8	Maria Soares	Rua da independencia, 42	São José	SC	48-91708090	mariasoaes@gmail.com	8	54131755157

	id_pessoa	nome_pessoa	logradouro_pessoa	cidade_pessoa	estado_pessoa	telefone_pessoa	email_pessoa	id_pessoa	cnpj
1	10	Mercado Livre	Geral dos Campos, 63	Curitiba	PR	41-32412313	ml@mercadolivre.com.br	10	57.040.264/0001-96

## 16º Opção de finalizar

```
=====
Escolha uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo ID
5 - Exibir todos
0 - Finalizar Programa
=====
0
Programa finalizado.
BUILD SUCCESSFUL (total time: 31 seconds)
```

## Análise e Conclusão:

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

#### Persistência em Arquivo:

- Dados armazenados em arquivos no sistema de arquivos.
- Recuperação envolve leitura do arquivo, menos eficiente para consultas complexas.
- Pode ter problemas de concorrência e geralmente não oferece suporte a transações.
- Menos escalável e pode ser desafiador para grandes volumes de dados.

#### Persistência em Banco de Dados:

- Dados armazenados em estruturas de banco de dados gerenciadas por um SGBD.
- Oferece consultas poderosas usando SQL, eficiente para consultas complexas.
- Suporta controle de concorrência e transações (conceito ACID).
- Mais escalável, com recursos de replicação e particionamento.
- Fornece recursos avançados de segurança, backup e manutenção.

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

*O uso de lambdas no Java simplifica a impressão de valores em entidades, proporcionando uma sintaxe mais concisa e expressiva para operações como iterações em coleções.*

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

*Porque o método main é estático e não pode chamar métodos de instância diretamente. Métodos estáticos pertencem à classe e podem ser invocados sem criar uma instância da classe.*