



Centro Universitário Estácio

Curso: Desenvolvimento Full Stack

Disciplina: Back-end sem Banco Não Tem

Semestre: 3º

Aluno: Jonnatha Walben Saldanha da Silva

1º Procedimento – Mapeamento Objeto-Relacional DAO

O objetivo da prática realizada é entender e demonstrar como conectar um aplicativo Java a um banco de dados SQL Server, realizar operações de inserção, leitura, atualização e exclusão (CRUD), e exibir os resultados no console do NetBeans. Esta prática ajuda a:

Estabelecer Conexão: Aprender a configurar e estabelecer uma conexão segura com um banco de dados SQL Server usando JDBC.

Realizar Operações CRUD: Executar operações básicas de criação, leitura, atualização e exclusão em tabelas do banco de dados.

Executar Consultas SQL: Executar e manipular resultados de consultas SQL em Java.

Tratar Exceções: Gerenciar exceções SQL para lidar com erros de forma controlada.

Boas Práticas: Aplicar boas práticas, como uso de PreparedStatements para evitar SQL Injection e fechamento adequado de recursos.

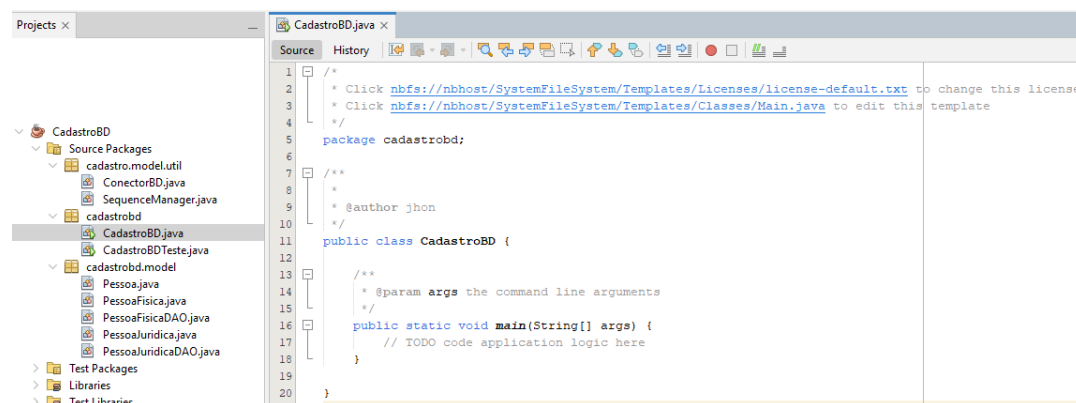
Integração com Ferramentas: Utilizar o NetBeans como IDE para desenvolvimento, depuração e execução do aplicativo Java.

Em resumo, a prática prepara para o desenvolvimento de aplicações empresariais, ensinando a conectar e interagir com bancos de dados, realizar operações básicas, e tratar exceções de maneira eficaz.

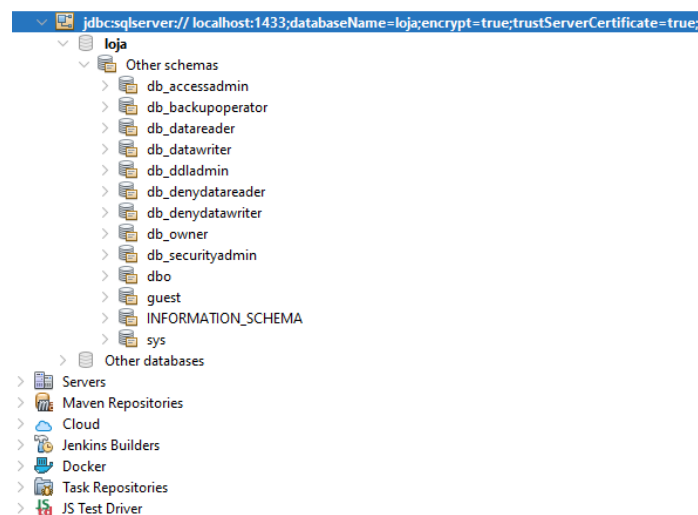
Códigos Solicitados:

Aqui está um resumo dos códigos solicitados e apresentados ao longo do procedimento:

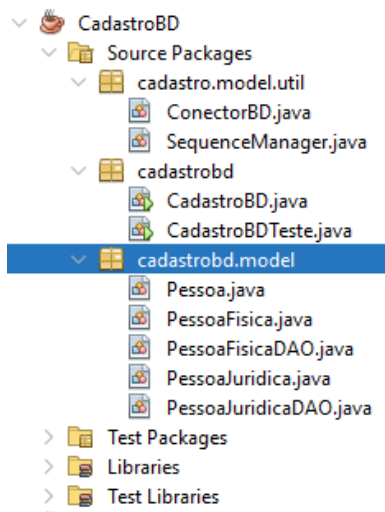
CadastroBD



Conexão com Banco de Dados



Cadastrobd.model



Classe Pessoa

```
package cadastrobd.model;

/**
 *
 * @author jhon
 */
public class Pessoa {
    private int id;
    private String nome;
    private String logradouro;
    private String cidade;
    private String estado;
    private String telefone;
    private String email;

    public Pessoa() {
        // Construtor padrão
    }

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

    public void exibir() {
        System.out.println("ID: " + id);
        System.out.println("Nome: " + nome);
        System.out.println("Logradouro: " + logradouro);
        System.out.println("Cidade: " + cidade);
        System.out.println("Estado: " + estado);
        System.out.println("Telefone: " + telefone);
        System.out.println("E-mail: " + email);
    }

    public int getId() {
        return id;
    }

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

    public String getNome() {
        return nome;
    }

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

```

    }

    public String getLogradouro() {
        return logradouro;
    }

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

    public String getCidade() {
        return cidade;
    }

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

    public String getEstado() {
        return estado;
    }

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

    public String getTelefone() {
        return telefone;
    }

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

    public String getEmail() {
        return email;
    }

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

```

Classe PessoaFisica

```

5 package cadastrobd.model;
6 /**
7  *
8  * @author jhon
9  */
10 public class PessoaFisica extends Pessoa {
11     private String cpf;
12
13     public PessoaFisica() {
14         super();
15     }
16
17     public PessoaFisica(int id, String nome, String logradouro, String cidade, String estado, String telefone, String email, String cpf)
18     {
19         super(id, nome, logradouro, cidade, estado, telefone, email);
20         this.cpf = cpf;
21     }
22
23     @Override
24     public void exibir() {
25         super.exibir();
26         System.out.println("CPF: " + cpf);
27     }
28 }

```

Classe PessoaJuridica

```

5 package cadastrobd.model;
6 /**
7  *
8  * @author jhon
9  */
10 public class PessoaJuridica extends Pessoa {
11     private String cnpj;
12
13     public PessoaJuridica() {
14         // Construtor padrão
15     }
16
17     public PessoaJuridica(int id, String nome, String logradouro, String cidade, String estado, String telefone, String email, String cnpj)
18     {
19         super(id, nome, logradouro, cidade, estado, telefone, email);
20         this.cnpj = cnpj;
21     }
22
23     @Override
24     public void exibir() {
25         super.exibir();
26         System.out.println("CNPJ: " + cnpj);
27     }
28 }

```

ConectorBD

```
import java.sql.*;

public class ConectorBD {

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

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

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

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

    public static void close(Statement stmt) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
            }
        }
    }

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

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

Classe SequenceManager

```
5 package cadastro.model.util;
6
7 /**
8  *
9  * @author jhon
10 */
11 import java.sql.Connection;
12 import java.sql.PreparedStatement;
13 import java.sql.ResultSet;
14 import java.sql.SQLException;
15
16 public class SequenceManager {
17
18     public static int getValue(String sequenceName) {
19         int nextValue = -1;
20         String sql = "SELECT NEXT VALUE FOR " + sequenceName + " AS nextval";
21         try (Connection conn = ConectorBD.getConnection();
22             PreparedStatement stmt = ConectorBD.getPrepared(conn, sql);
23             ResultSet rs = ConectorBD.getSelect(stmt)) {
24
25             if (rs.next()) {
26                 nextValue = rs.getInt("nextval");
27             }
28         } catch (SQLException e) {
29             //
30         }
31         return nextValue;
32     }
33 }
```

Classe PessoaFisicaDAO

```
package cadastrobd.model;

/**
 *
 * @author jhon
 */
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {

    public PessoaFisica getPessoa(int id) {
        String sql = "SELECT p.*, pf.CPF FROM Pessoa p JOIN PessoaFisica pf ON p.ID_Pessoa = pf.ID_Pessoa WHERE p.ID_Pessoa = ?";
        PessoaFisica pessoa = null;
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = ConectorBD.getPrepared(conn, sql)) {

            stmt.setInt(1, id);
            try (ResultSet rs = ConectorBD.getSelect(stmt)) {
                if (rs.next()) {
                    pessoa = new PessoaFisica(
                        rs.getInt("ID_Pessoa"),
                        rs.getString("Nome"),
                        rs.getString("Logradouro"),
                        rs.getString("Cidade"),
                        rs.getString("Estado"),
                        rs.getString("Telefone"),
                        rs.getString("Email"),
                        rs.getString("CPF")
                    );
                }
            }
        } catch (SQLException e) {
            //
        }
        return pessoa;
    }

    public List<PessoaFisica> getPessoas() {
        String sql = "SELECT p.*, pf.CPF FROM Pessoa p JOIN PessoaFisica pf ON p.ID_Pessoa = pf.ID_Pessoa";
        List<PessoaFisica> pessoas = new ArrayList<>();
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = ConectorBD.getPrepared(conn, sql);
            ResultSet rs = ConectorBD.getSelect(stmt)) {

            while (rs.next()) {

```

```

        PessoaFisica pessoa = new PessoaFisica(
            rs.getInt("ID_Pessoa"),
            rs.getString("Nome"),
            rs.getString("Logradouro"),
            rs.getString("Cidade"),
            rs.getString("Estado"),
            rs.getString("Telefone"),
            rs.getString("Email"),
            rs.getString("CPF")
        );
        pessoas.add(pessoa);
    }
} catch (SQLException e) {
}
return pessoas;
}

public boolean incluir(PessoaFisica pessoa) {
    boolean result = false;
    String sqlPessoa = "INSERT INTO Pessoa (ID_Pessoa, Nome, Telefone, Logradouro, Cidade, Estado, Email) VALUES (?, ?, ?, ?, ?, ?, ?)";
    String sqlPessoaFisica = "INSERT INTO PessoaFisica (ID_Pessoa, CPF) VALUES (?, ?)";

    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement stmtPessoa = ConectorBD.getPrepared(conn, sqlPessoa);
        PreparedStatement stmtPessoaFisica = ConectorBD.getPrepared(conn, sqlPessoaFisica)) {

        conn.setAutoCommit(false);
        int nextId = SequenceManager.getValue("Seq_ID_Pessoa");
        stmtPessoa.setInt(1, nextId);
        stmtPessoa.setString(2, pessoa.getNome());
        stmtPessoa.setString(3, pessoa.getTelefone());
        stmtPessoa.setString(4, pessoa.getLogradouro());
        stmtPessoa.setString(5, pessoa.getCidade());
        stmtPessoa.setString(6, pessoa.getEstado());
        stmtPessoa.setString(7, pessoa.getEmail());
        stmtPessoa.executeUpdate();

        stmtPessoaFisica.setInt(1, nextId);
        stmtPessoaFisica.setString(2, pessoa.getCpf());
        stmtPessoaFisica.executeUpdate();

        conn.commit();
        result = true;
    } catch (SQLException e) {
        try (Connection conn = ConectorBD.getConnection()) {
            conn.rollback();
        } catch (SQLException ex) {
        }
    }
    return result;
}

```

```
        stmtPessoa.setInt(1, id);
        stmtPessoa.executeUpdate();

        conn.commit();
        result = true;
    } catch (SQLException e) {
        try (Connection conn = ConectorBD.getConnection()) {
            conn.rollback();
        } catch (SQLException ex) {
        }
    }
    return result;
}
}
```



```

    }

    public boolean alterar(PessoaFisica pessoa) {
        boolean result = false;
        String sqlPessoa = "UPDATE Pessoa SET Nome = ?, Telefone = ?, Logradouro
= ?, Cidade = ?, Estado = ?, Email = ? WHERE ID_Pessoa = ?";
        String sqlPessoaFisica = "UPDATE PessoaFisica SET CPF = ? WHERE
ID_Pessoa = ?";

        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmtPessoa = ConectorBD.getPrepared(conn,
sqlPessoa);
            PreparedStatement stmtPessoaFisica = ConectorBD.getPrepared(conn,
sqlPessoaFisica)) {

            conn.setAutoCommit(false);
            stmtPessoa.setString(1, pessoa.getNome());
            stmtPessoa.setString(2, pessoa.getTelefone());
            stmtPessoa.setString(3, pessoa.getLogradouro());
            stmtPessoa.setString(4, pessoa.getCidade());
            stmtPessoa.setString(5, pessoa.getEstado());
            stmtPessoa.setString(6, pessoa.getEmail());
            stmtPessoa.setInt(7, pessoa.getId());
            stmtPessoa.executeUpdate();

            stmtPessoaFisica.setString(1, pessoa.getCpf());
            stmtPessoaFisica.setInt(2, pessoa.getId());
            stmtPessoaFisica.executeUpdate();

            conn.commit();
            result = true;
        } catch (SQLException e) {
            try (Connection conn = ConectorBD.getConnection()) {
                conn.rollback();
            } catch (SQLException ex) {
            }
        }
        return result;
    }

    public boolean excluir(int id) {
        boolean result = false;
        String sqlPessoaFisica = "DELETE FROM PessoaFisica WHERE ID_Pessoa = ?";
        String sqlPessoa = "DELETE FROM Pessoa WHERE ID_Pessoa = ?";

        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmtPessoaFisica = ConectorBD.getPrepared(conn,
sqlPessoaFisica);
            PreparedStatement stmtPessoa = ConectorBD.getPrepared(conn,
sqlPessoa)) {

            conn.setAutoCommit(false);
            stmtPessoaFisica.setInt(1, id);
            stmtPessoaFisica.executeUpdate();

```

Classe PessoaJuridica

```
package cadastrobd.model;

/**
 *
 * @author jhon
 */
import cadastro.model.util.SequenceManager;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaDAO {

    public PessoaJuridica getPessoa(int id) {
        String sql = "SELECT p.*, pj.CNPJ FROM Pessoa p JOIN PessoaJuridica pj ON p.ID_Pessoa = pj.ID_Pessoa WHERE p.ID_Pessoa = ?";
        PessoaJuridica pessoa = null;
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = ConectorBD.getPrepared(conn, sql)) {

            stmt.setInt(1, id);
            try (ResultSet rs = ConectorBD.getSelect(stmt)) {
                if (rs.next()) {
                    pessoa = new PessoaJuridica(
                        rs.getInt("ID_Pessoa"),
                        rs.getString("Nome"),
                        rs.getString("Logradouro"),
                        rs.getString("Cidade"),
                        rs.getString("Estado"),
                        rs.getString("Telefone"),
                        rs.getString("Email"),
                        rs.getString("CNPJ")
                    );
                }
            }
        } catch (SQLException e) {
        }
        return pessoa;
    }

    public List<PessoaJuridica> getPessoas() {
        String sql = "SELECT p.*, pj.CNPJ FROM Pessoa p JOIN PessoaJuridica pj ON p.ID_Pessoa = pj.ID_Pessoa";
        List<PessoaJuridica> pessoas = new ArrayList<>();
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = ConectorBD.getPrepared(conn, sql);
            ResultSet rs = ConectorBD.getSelect(stmt)) {

            while (rs.next()) {
```

```

        PessoaJuridica pessoa = new PessoaJuridica(
            rs.getInt("ID_Pessoa"),
            rs.getString("Nome"),
            rs.getString("Logradouro"),
            rs.getString("Cidade"),
            rs.getString("Estado"),
            rs.getString("Telefone"),
            rs.getString("Email"),
            rs.getString("CNPJ")
        );
        pessoas.add(pessoa);
    }
} catch (SQLException e) {
}
return pessoas;
}

public boolean incluir(PessoaJuridica pessoa) {
    boolean result = false;
    String sqlPessoa = "INSERT INTO Pessoa (ID_Pessoa, Nome, Telefone, Logradouro, Cidade, Estado, Email) VALUES (?, ?, ?, ?, ?, ?, ?)";
    String sqlPessoaJuridica = "INSERT INTO PessoaJuridica (ID_Pessoa, CNPJ) VALUES (?, ?)";

    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement stmtPessoa = ConectorBD.getPrepared(conn, sqlPessoa);
        PreparedStatement stmtPessoaJuridica = ConectorBD.getPrepared(conn, sqlPessoaJuridica)) {

        conn.setAutoCommit(false);
        int nextId = SequenceManager.getValue("Seq_ID_Pessoa");
        stmtPessoa.setInt(1, nextId);
        stmtPessoa.setString(2, pessoa.getNome());
        stmtPessoa.setString(3, pessoa.getTelefone());
        stmtPessoa.setString(4, pessoa.getLogradouro());
        stmtPessoa.setString(5, pessoa.getCidade());
        stmtPessoa.setString(6, pessoa.getEstado());
        stmtPessoa.setString(7, pessoa.getEmail());
        stmtPessoa.executeUpdate();

        stmtPessoaJuridica.setInt(1, nextId);
        stmtPessoaJuridica.setString(2, pessoa.getCnpj());

        conn.commit();
        result = true;
    } catch (SQLException e) {
        try (Connection conn = ConectorBD.getConnection()) {
            conn.rollback();
        } catch (SQLException ex) {
        }
    }
    return result;
}

```

```

    public boolean alterar(PessoaJuridica pessoa) {
        boolean result = false;
        String sqlPessoa = "UPDATE Pessoa SET Nome = ?, Telefone = ?, Logradouro
= ?, Cidade = ?, Estado = ?, Email = ? WHERE ID_Pessoa = ?";
        String sqlPessoaJuridica = "UPDATE PessoaJuridica SET CNPJ = ? WHERE
ID_Pessoa = ?";

        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmtPessoa = ConectorBD.getPrepared(conn,
sqlPessoa);
            PreparedStatement stmtPessoaJuridica = ConectorBD.getPrepared(conn,
sqlPessoaJuridica)) {

            conn.setAutoCommit(false);
            stmtPessoa.setString(1, pessoa.getNome());
            stmtPessoa.setString(2, pessoa.getTelefone());
            stmtPessoa.setString(3, pessoa.getLogradouro());
            stmtPessoa.setString(4, pessoa.getCidade());
            stmtPessoa.setString(5, pessoa.getEstado());
            stmtPessoa.setString(6, pessoa.getEmail());
            stmtPessoa.setInt(7, pessoa.getId());
            stmtPessoa.executeUpdate();

            stmtPessoaJuridica.setString(1, pessoa.getCnpj());
            stmtPessoaJuridica.setInt(2, pessoa.getId());
            stmtPessoaJuridica.executeUpdate();

            conn.commit();
            result = true;
        } catch (SQLException e) {
            try (Connection conn = ConectorBD.getConnection()) {
                conn.rollback();
            } catch (SQLException ex) {
            }
        }
        return result;
    }

    public boolean excluir(int id) {
        boolean result = false;
        String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE ID_Pessoa =
?";
        String sqlPessoa = "DELETE FROM Pessoa WHERE ID_Pessoa = ?";

        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmtPessoaJuridica = ConectorBD.getPrepared(conn,
sqlPessoaJuridica);
            PreparedStatement stmtPessoa = ConectorBD.getPrepared(conn,
sqlPessoa)) {

            conn.setAutoCommit(false);
            stmtPessoaJuridica.setInt(1, id);
            stmtPessoaJuridica.executeUpdate();

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

            conn.commit();
            result = true;
        } catch (SQLException e) {
            try (Connection conn = ConectorBD.getConnection()) {
                conn.rollback();
            } catch (SQLException ex) {
            }
        }
        return result;
    }
}

```

Classe CadatroBDTeste

```
package cadastrobd;

/**
 *
 * @author jhon
 */

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;

public class CadastroBDTeste {

    public static void main(String[] args) {
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

        // Instanciar uma pessoa fisica e persistir no banco de dados
        PessoaFisica pessoaFisica = new PessoaFisica(0, "João", "Rua B", "Cidade
A", "RJ", "1121-1111", "joao@rracho.com", "11111121111");
        pessoaFisicaDAO.incluir(pessoaFisica);

        // Alterar os dados da pessoa fisica no banco
        pessoaFisica.setNome("João Alterado");
        pessoaFisicaDAO.alterar(pessoaFisica);

        // Consultar todas as pessoas fisicas do banco de dados e listar no
console
        for (PessoaFisica pf : pessoaFisicaDAO.getPessoas()) {
            pf.exibir();
        }

        // Excluir a pessoa fisica criada anteriormente no banco
        pessoaFisicaDAO.excluir(pessoaFisica.getId());

        // Instanciar uma pessoa juridica e persistir no banco de dados
        PessoaJuridica pessoaJuridica = new PessoaJuridica(0, "Empresa X", "Rua
B", "Cidade B", "PA", "1212-1212", "emaesa@riacho.com", "1111111121111");
        pessoaJuridicaDAO.incluir(pessoaJuridica);

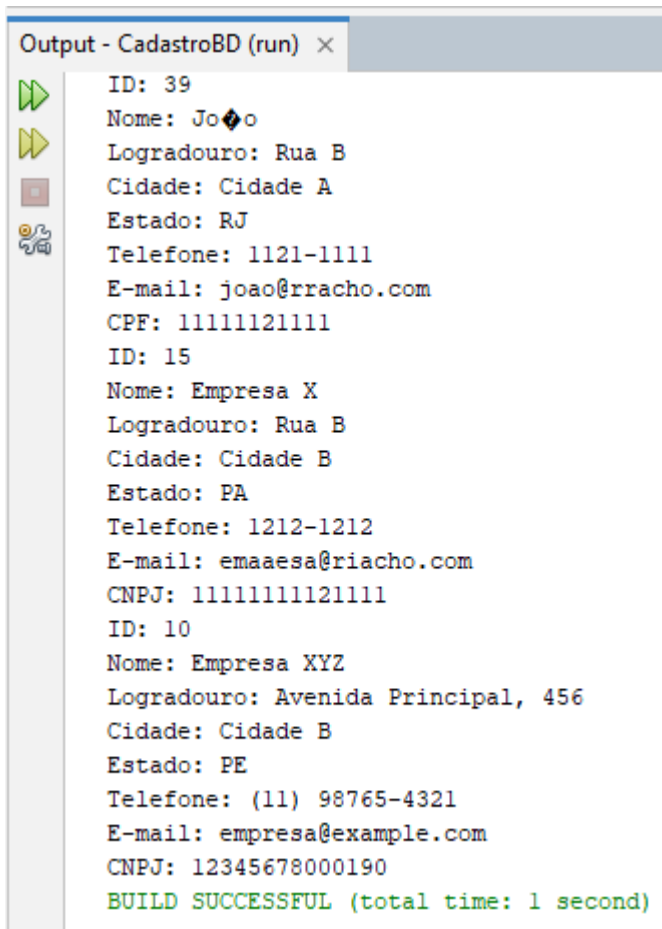
        // Alterar os dados da pessoa juridica no banco
        pessoaJuridica.setNome("Empresa X Alterada");
        pessoaJuridicaDAO.alterar(pessoaJuridica);

        // Consultar todas as pessoas juridicas do banco de dados e listar no
console
        for (PessoaJuridica pj : pessoaJuridicaDAO.getPessoas()) {
            pj.exibir();
        }

        // Excluir a pessoa juridica criada anteriormente no banco
        pessoaJuridicaDAO.excluir(pessoaJuridica.getId());
    }

}
```

Resultado CadastroBDTeste



```
Output - CadastroBD (run) ×
ID: 39
Nome: João
Logradouro: Rua B
Cidade: Cidade A
Estado: RJ
Telefone: 1121-1111
E-mail: joao@rracho.com
CPF: 11111121111
ID: 15
Nome: Empresa X
Logradouro: Rua B
Cidade: Cidade B
Estado: PA
Telefone: 1212-1212
E-mail: emaaesa@riacho.com
CNPJ: 1111111121111
ID: 10
Nome: Empresa XYZ
Logradouro: Avenida Principal, 456
Cidade: Cidade B
Estado: PE
Telefone: (11) 98765-4321
E-mail: empresa@example.com
CNPJ: 12345678000190
BUILD SUCCESSFUL (total time: 1 second)
```

Análise e Conclusão

1. Importância dos Componentes de Middleware como o JDBC

JDBC (Java Database Connectivity) é uma API que permite a execução de operações SQL em bancos de dados a partir da linguagem Java. Ele funciona como um middleware, oferecendo uma interface padrão para interagir com diferentes sistemas de gerenciamento de banco de dados (SGBDs).

Importância:

- **Abstração e Padronização:** JDBC fornece uma interface padronizada para acessar e manipular dados em vários tipos de bancos de dados. Isso abstrai os detalhes específicos do SGBD e facilita a portabilidade de aplicações Java entre diferentes bancos de dados.

- **Conectividade:** Facilita a conexão a bancos de dados a partir de aplicações Java, permitindo a execução de consultas, atualizações, e manipulação de dados.
- **Flexibilidade:** Suporta diversas operações de banco de dados, incluindo transações, execução de comandos SQL, e manipulação de resultados.
- **Segurança:** Permite a utilização de PreparedStatement, que previne ataques de SQL Injection, aumentando a segurança da aplicação.

2. Diferença no Uso de Statement ou PreparedStatement para Manipulação de Dados

Statement:

- **Uso:** Utilizado para executar instruções SQL estáticas que não necessitam de parâmetros.
- **Segurança:** Mais vulnerável a ataques de SQL Injection, pois concatena strings para formar a instrução SQL.
- **Desempenho:** Menos eficiente em cenários onde a mesma instrução SQL é executada repetidamente com diferentes parâmetros, pois o SQL é recompilado e otimizado pelo banco de dados a cada execução.

PreparedStatement:

- **Uso:** Utilizado para instruções SQL parametrizadas. Permite definir a estrutura da instrução SQL uma vez e fornecer valores para os parâmetros em tempo de execução.
- **Segurança:** Mais seguro contra ataques de SQL Injection, pois os parâmetros são tratados separadamente do código SQL.
- **Desempenho:** Mais eficiente para instruções repetitivas, pois o SQL é pré-compilado e otimizado pelo banco de dados apenas uma vez, reutilizando o plano de execução para cada nova execução com diferentes parâmetros.

3. Como o Padrão DAO Melhora a Manutenibilidade do Software

DAO (Data Access Object):

- **Encapsulamento:** Isola o acesso aos dados do restante da aplicação, encapsulando toda a lógica de acesso a dados em uma única camada.
- **Reutilização:** Permite a reutilização de código de acesso a dados em diferentes partes da aplicação.
- **Facilidade de Manutenção:** Simplifica a manutenção e evolução da aplicação, permitindo que alterações na lógica de acesso a dados sejam feitas em um único lugar sem impactar outras partes da aplicação.

- **Testabilidade:** Facilita a criação de testes unitários, permitindo a criação de mocks ou stubs para testar a lógica de negócio sem acessar o banco de dados real.
- **Independência de Banco de Dados:** Facilita a troca de SGBD ou a adoção de novas tecnologias de armazenamento de dados, pois a lógica de acesso a dados está centralizada e pode ser adaptada sem afetar o restante da aplicação.

4. Como a Herança é Refletida no Banco de Dados em um Modelo Estritamente Relacional

Herança no Modelo Relacional:

No modelo relacional, a herança pode ser implementada de várias formas para refletir a estrutura hierárquica das classes de uma aplicação orientada a objetos:

- **Table per Hierarchy (TPH):** Todas as classes da hierarquia são mapeadas para uma única tabela. Uma coluna discriminadora é usada para identificar o tipo de cada registro.
 - **Vantagens:** Simplicidade, não há necessidade de joins para consultas.
 - **Desvantagens:** Pode resultar em muitas colunas nulas, dificuldades de validação de integridade de dados.
- **Table per Concrete Class (TPC):** Cada classe concreta da hierarquia tem sua própria tabela, incluindo colunas para todos os campos, mesmo aqueles herdados.
 - **Vantagens:** Simplicidade nas consultas.
 - **Desvantagens:** Redundância de dados, dificuldade na manutenção e nas operações de atualização em classes base.
- **Table per Subclass (TPS):** Cada classe na hierarquia tem sua própria tabela. As tabelas das subclasses contêm apenas os campos específicos da subclasse, enquanto a tabela da classe base contém os campos comuns.
 - **Vantagens:** Normalização, sem redundância de dados.
 - **Desvantagens:** Necessidade de joins para consultas, maior complexidade nas operações CRUD.

Conclusão:

Os componentes de middleware como JDBC são cruciais para a integração de aplicações Java com bancos de dados, fornecendo uma interface padronizada e facilitando a segurança e a performance da aplicação. A escolha entre **Statement** e **PreparedStatement** afeta diretamente a segurança e a eficiência da manipulação de dados. O padrão DAO contribui significativamente para a manutenção,

testabilidade e escalabilidade do software. A herança, quando refletida em um modelo relacional, pode ser implementada de várias formas, cada uma com seus prós e contras, e a escolha da estratégia adequada depende dos requisitos específicos da aplicação.

2º Procedimento – Alimentando a Base

Código da alteração **main**:

```
package cadastrobd;

/**
 *
 * @author jhon
 */
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;

public class CadastroBDTeste {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

        while (true) {
            System.out.println("Selecione uma opção:");
            System.out.println("1. Incluir");
            System.out.println("2. Alterar");
            System.out.println("3. Excluir");
            System.out.println("4. Exibir pelo ID");
            System.out.println("5. Exibir todos");
            System.out.println("0. Finalizar");
            int opcao = scanner.nextInt();
            scanner.nextLine(); // Consumir nova linha

            try {
                switch (opcao) {
                    case 1 -> incluir(scanner, pessoaFisicaDAO,
pessoaJuridicaDAO);
                    case 2 -> alterar(scanner, pessoaFisicaDAO,
pessoaJuridicaDAO);
                    case 3 -> excluir(scanner, pessoaFisicaDAO,
pessoaJuridicaDAO);
                    case 4 -> exibirPorId(scanner, pessoaFisicaDAO,
pessoaJuridicaDAO);
                    case 5 -> exibirTodos(scanner, pessoaFisicaDAO,
pessoaJuridicaDAO);
                    case 0 -> {
                        System.out.println("Encerrando o programa.");
                        scanner.close();
                        return;
                    }
                    default -> System.out.println("Opção inválida. Tente
novamente.");
                }
            } catch (SQLException e) {
```

```

        System.out.println("Erro ao executar operações no banco de
dados: " + e.getMessage());
    }
}

private static void incluir(Scanner scanner, PessoaFisicaDAO
pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {
    System.out.println("Escolha o tipo (1. Física, 2. Jurídica:");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir nova linha
    switch (tipo) {
        case 1 -> {
            PessoaFisica pessoaFisica = new PessoaFisica();
            System.out.println("Nome:");
            pessoaFisica.setNome(scanner.nextLine());
            System.out.println("Endereço:");
            pessoaFisica.setLogradouro(scanner.nextLine());
            System.out.println("Cidade:");
            pessoaFisica.setCidade(scanner.nextLine());
            System.out.println("Estado:");
            pessoaFisica.setEstado(scanner.nextLine());
            System.out.println("Telefone:");
            pessoaFisica.setTelefone(scanner.nextLine());
            System.out.println("Email:");
            pessoaFisica.setEmail(scanner.nextLine());
            System.out.println("CPF:");
            pessoaFisica.setCpf(scanner.nextLine());
            pessoaFisicaDAO.incluir(pessoaFisica);
        }
        case 2 -> {
            PessoaJuridica pessoaJuridica = new PessoaJuridica();
            System.out.println("Nome:");
            pessoaJuridica.setNome(scanner.nextLine());
            System.out.println("Endereço:");
            pessoaJuridica.setLogradouro(scanner.nextLine());
            System.out.println("Cidade:");
            pessoaJuridica.setCidade(scanner.nextLine());
            System.out.println("Estado:");
            pessoaJuridica.setEstado(scanner.nextLine());
            System.out.println("Telefone:");
            pessoaJuridica.setTelefone(scanner.nextLine());
            System.out.println("Email:");
            pessoaJuridica.setEmail(scanner.nextLine());
            System.out.println("CNPJ:");
            pessoaJuridica.setCnpj(scanner.nextLine());
            pessoaJuridicaDAO.incluir(pessoaJuridica);
        }
        default -> System.out.println("Tipo inválido.");
    }
}

private static void alterar(Scanner scanner, PessoaFisicaDAO
pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {

```

```

System.out.println("Escolha o tipo (1. Física, 2. Jurídica):");
int tipo = scanner.nextInt();
scanner.nextLine(); // Consumir nova linha
switch (tipo) {
    case 1 -> {
        System.out.println("ID da pessoa física:");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consumir nova linha
        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);
        if (pessoaFisica != null) {
            System.out.println("Dados atuais: ");
            pessoaFisica.exibir();
            System.out.println("Novos dados:");
            System.out.println("Nome:");
            pessoaFisica.setNome(scanner.nextLine());
            System.out.println("Endereço:");
            pessoaFisica.setLogradouro(scanner.nextLine());
            System.out.println("Cidade:");
            pessoaFisica.setCidade(scanner.nextLine());
            System.out.println("Estado:");
            pessoaFisica.setEstado(scanner.nextLine());
            System.out.println("Telefone:");
            pessoaFisica.setTelefone(scanner.nextLine());
            System.out.println("Email:");
            pessoaFisica.setEmail(scanner.nextLine());
            System.out.println("CPF:");
            pessoaFisica.setCpf(scanner.nextLine());
            pessoaFisicaDAO.alterar(pessoaFisica);
        } else {
            System.out.println("Pessoa física não encontrada.");
        }
    }
    case 2 -> {
        System.out.println("ID da pessoa jurídica:");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consumir nova linha
        PessoaJuridica pessoaJuridica =
        pessoaJuridicaDAO.getPessoa(id);
        if (pessoaJuridica != null) {
            System.out.println("Dados atuais: ");
            pessoaJuridica.exibir();
            System.out.println("Novos dados:");
            System.out.println("Nome:");
            pessoaJuridica.setNome(scanner.nextLine());
            System.out.println("Endereço:");
            pessoaJuridica.setLogradouro(scanner.nextLine());
            System.out.println("Cidade:");
            pessoaJuridica.setCidade(scanner.nextLine());
            System.out.println("Estado:");
            pessoaJuridica.setEstado(scanner.nextLine());
            System.out.println("Telefone:");
            pessoaJuridica.setTelefone(scanner.nextLine());
            System.out.println("Email:");
            pessoaJuridica.setEmail(scanner.nextLine());
            System.out.println("CNPJ:");

```

```

        pessoaJuridica.setCnpj(scanner.nextLine());
        pessoaJuridicaDAO.alterar(pessoaJuridica);
    } else {
        System.out.println("Pessoa jurídica não encontrada.");
    }
    default -> System.out.println("Tipo inválido.");
}
}

private static void excluir(Scanner scanner, PessoaFisicaDAO
pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {
    System.out.println("Escolha o tipo (1. Física, 2. Jurídica:");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir nova linha
    switch (tipo) {
        case 1 -> {
            System.out.println("ID da pessoa física:");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consumir nova linha
            pessoaFisicaDAO.excluir(id);
        }
        case 2 -> {
            System.out.println("ID da pessoa jurídica:");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consumir nova linha
            pessoaJuridicaDAO.excluir(id);
        }
        default -> System.out.println("Tipo inválido.");
    }
}

private static void exibirPorId(Scanner scanner, PessoaFisicaDAO
pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {
    System.out.println("Escolha o tipo (1. Física, 2. Jurídica:");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir nova linha
    switch (tipo) {
        case 1 -> {
            System.out.println("ID da pessoa física:");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consumir nova linha
            PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);
            if (pessoaFisica != null) {
                pessoaFisica.exibir();
            } else {
                System.out.println("Pessoa física não encontrada.");
            }
        }
        case 2 -> {
            System.out.println("ID da pessoa jurídica:");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consumir nova linha
            PessoaJuridica pessoaJuridica =
pessoaJuridicaDAO.getPessoa(id);
            if (pessoaJuridica != null) {

```

Análise e Conclusão:

1. Diferenças entre Persistência em Arquivo e Banco de Dados:

- **Persistência em Arquivo:** Os dados são armazenados em arquivos no sistema de arquivos do computador. É geralmente mais simples de implementar e adequado para pequenas quantidades de dados. No entanto, pode ser menos eficiente para consultas e manipulações complexas.
- **Persistência em Banco de Dados:** Os dados são armazenados em um sistema de gerenciamento de banco de dados, como o SQL Server, MySQL ou PostgreSQL. Oferece recursos avançados de consulta, indexação e transações. É mais adequado para aplicativos que exigem manipulação intensiva de dados e necessitam de suporte para várias operações simultâneas.

2. Uso de Operador Lambda para Simplificar a Impressão de Valores:

- O uso de operadores lambda introduzido no Java 8 simplifica a escrita de código ao lidar com tarefas repetitivas, como a impressão de valores em uma coleção. Ele permite que você forneça uma implementação de uma interface funcional diretamente no local de uso, eliminando a necessidade de definir uma classe separada ou uma expressão de classe anônima.

3. Métodos Acionados Diretamente pelo Método Main Marcação como Static:

- Os métodos acionados diretamente pelo método `main` em uma classe Java precisam ser marcados como **static** porque o método `main` é estático. Isso significa que ele pertence à classe em si e não a uma instância específica da classe. Para ser acessado diretamente de um contexto estático, como o método `main`, outros métodos também precisam ser estáticos, garantindo que possam ser chamados sem a necessidade de uma instância da classe.

```

        pessoaJuridica.exibir();
    } else {
        System.out.println("Pessoa jurídica não encontrada.");
    }
    default -> System.out.println("Tipo inválido.");
}
}

private static void exibirTodos(Scanner scanner, PessoaFisicaDAO
pessoaFisicaDAO, PessoaJuridicaDAO pessoaJuridicaDAO) throws SQLException {
    System.out.println("Escolha o tipo (1. Física, 2. Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Consumir nova linha
    switch (tipo) {
        case 1 -> {
            List<PessoaFisica> pessoasFisicas =
pessoaFisicaDAO.getPessoas();
            for (PessoaFisica pf : pessoasFisicas) {
                pf.exibir();
            }
        }
        case 2 -> {
            List<PessoaJuridica> pessoasJuridicas =
pessoaJuridicaDAO.getPessoas();
            for (PessoaJuridica pj : pessoasJuridicas) {
                pj.exibir();
            }
        }
        default -> System.out.println("Tipo inválido.");
    }
}
}
}

```

Análise e Conclusão:

4. Diferenças entre Persistência em Arquivo e Banco de Dados:

- **Persistência em Arquivo:** Os dados são armazenados em arquivos no sistema de arquivos do computador. É geralmente mais simples de implementar e adequado para pequenas quantidades de dados. No entanto, pode ser menos eficiente para consultas e manipulações complexas.
- **Persistência em Banco de Dados:** Os dados são armazenados em um sistema de gerenciamento de banco de dados, como o SQL Server, MySQL ou PostgreSQL. Oferece recursos avançados de consulta, indexação e transações. É mais adequado para aplicativos que exigem manipulação intensiva de dados e necessitam de suporte para várias operações simultâneas.

5. Uso de Operador Lambda para Simplificar a Impressão de Valores:

- O uso de operadores lambda introduzido no Java 8 simplifica a escrita de código ao lidar com tarefas repetitivas, como a impressão de valores em uma coleção. Ele permite que você forneça uma implementação de uma interface funcional diretamente no local de uso, eliminando a necessidade de definir uma classe separada ou uma expressão de classe anônima.

6. Métodos Acionados Diretamente pelo Método Main Marcação como Static:

- Os métodos acionados diretamente pelo método **main** em uma classe Java precisam ser marcados como **static** porque o método **main** é estático. Isso significa que ele pertence à classe em si e não a uma instância específica da classe. Para ser acessado diretamente de um contexto estático, como o método **main**, outros métodos também precisam ser estáticos, garantindo que possam ser chamados sem a necessidade de uma instância da classe.