

DESENVOLVIMENTO FULL STACK
POLO SETOR O – CEILÂNDIA - DF
PERÍODO 2024.2 FLEX
DISCIPLINA: BACK END SEM BANCO NÃO TEM
MAYCON MOURA

Título da Prática:

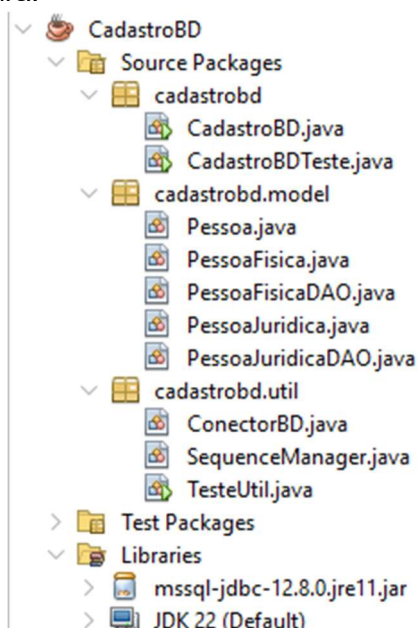
Mapeamento Objeto-Relacional e DAO - SQL Server e Java

Objetivo da Prática:

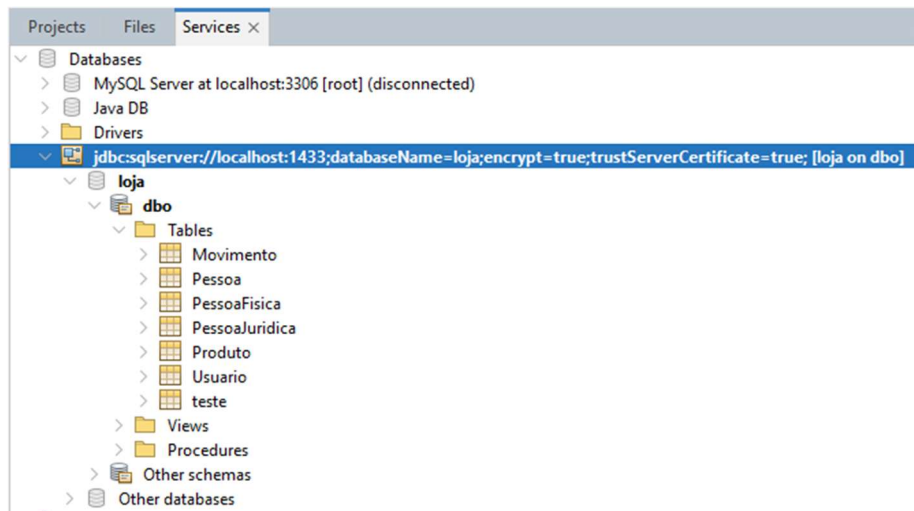
O objetivo desta prática foi desenvolver uma aplicação em Java que realiza operações CRUD (Create, Read, Update, Delete) em um banco de dados SQL Server, utilizando JDBC para a conexão com o banco e aplicando o padrão DAO (Data Access Object) para melhorar a organização e a manutenibilidade do código. Adicionalmente, a prática explorou o uso de herança em um modelo relacional, manipulando dados de pessoas físicas e jurídicas.

Códigos:

Estrutura:



Serviço DB:



Pessoa.java

```
package cadastrobd.model;
```

```
public class Pessoa {
```

```
    private int id;
```

```
    private String nome;
```

```
    private String endereco;
```

```
    private String telefone;
```

```
    private String email;
```

```
    private String tipo;
```

```
    // Construtor padrão
```

```
    public Pessoa() {}
```

```
    // Construtor completo
```

```
    public Pessoa(int id, String nome, String endereco, String telefone, String email) {
```

```
        this.id = id;
```

```
        this.nome = nome;
```

```
        this.endereco = endereco;
```

```
        this.telefone = telefone;
```

```
        this.email = email;
```

```
    }
```

```
    // Getters e Setters
```

```
    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 getEndereco() {
    return endereco;
}

public void setEndereco(String endereceo) {
    this.endereco = endereceo;
}

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;
}

// Método exibir
public void exibir() {
    System.out.println("ID: " + id);
    System.out.println("Nome: " + nome);
    System.out.println("Endereço: " + endereco);
    System.out.println("Telefone: " + telefone);
    System.out.println("Email: " + email);
}

/**
 * @return the tipo
 */
public String getTipo() {
    return tipo;
}

/**
 * @param tipo the tipo to set

```

```

        */
        public void setTipo(String tipo) {
            this.tipo = tipo;
        }
    }
}

```

PessoaFisica.java

```

package cadastrabd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

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

    // Construtor completo
    public PessoaFisica(int id, String nome, String endereceo, String telefone, String
email, String cpf) {
        super(id, nome, endereceo, telefone, email);
        this.cpf = cpf;
    }

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

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

    // Método exibir
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + cpf);
        System.out.println("-----");
    }
}

```

PessoaFisicaDAO.java

```

package cadastrobd.model;

import cadastrobd.util.ConectorBD;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
    public PessoaFisica getPessoa(int id) throws SQLException {
        String sql = "SELECT p.id_pessoa, p.nome, p.endereco, p.telefone, p.email, pf.cpf " +
            "FROM Pessoa p INNER JOIN PessoaFisica pf ON p.id_pessoa = pf.id_pessoa WHERE p.id_pessoa = ?";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return new PessoaFisica(
                        rs.getInt("id_pessoa"),
                        rs.getString("nome"),
                        rs.getString("endereco"),
                        rs.getString("telefone"),
                        rs.getString("email"),
                        rs.getString("cpf")
                    );
                }
            }
        }
        return null;
    }

    public List<PessoaFisica> getPessoas() throws SQLException {
        List<PessoaFisica> pessoas = new ArrayList<>();
        String sql = "SELECT p.id_pessoa, p.nome, p.endereco, p.telefone, p.email, pf.cpf " +
            "FROM Pessoa p INNER JOIN PessoaFisica pf ON p.id_pessoa = pf.id_pessoa";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                pessoas.add(new PessoaFisica(
                    rs.getInt("id_pessoa"),
                    rs.getString("nome"),
                    rs.getString("endereco"),
                    rs.getString("telefone"),

```

```

        rs.getString("email"),
        rs.getString("cpf")
    ));
    }
}
return pessoas;
}

public void incluir(PessoaFisica pessoa) throws SQLException {
    String sqlPessoa = "INSERT INTO Pessoa (id_pessoa, nome, tipo, endereco,
telefone, email) VALUES (?, ?, ?, ?, ?, ?)";
    String sqlPessoaFisica = "INSERT INTO PessoaFisica (id_pessoa, cpf) VALUES
(?, ?)";
    pessoa.setTipo("F");
    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement psPessoa = conn.prepareStatement(sqlPessoa);
        PreparedStatement psPessoaFisica = conn.prepareStatement(sqlPessoaFisica)) {
        conn.setAutoCommit(false);

        psPessoa.setInt(1, pessoa.getId());
        psPessoa.setString(2, pessoa.getNome());
        psPessoa.setString(3, pessoa.getTipo());
        psPessoa.setString(4, pessoa.getEndereco());
        psPessoa.setString(5, pessoa.getTelefone());
        psPessoa.setString(6, pessoa.getEmail());
        psPessoa.executeUpdate();

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

        conn.commit();
    } catch (SQLException e) {
        e.printStackTrace();
        throw e;
    }
}

public void alterar(PessoaFisica pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoa SET nome = ?, endereco = ?, telefone = ?,
email = ? WHERE id_pessoa = ?";
    String sqlPessoaFisica = "UPDATE PessoaFisica SET cpf = ? WHERE
id_pessoa = ?";

    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement psPessoa = conn.prepareStatement(sqlPessoa);

```

```

        PreparedStatement psPessoaFisica =
conn.prepareStatement(sqlPessoaFisica)) {
    conn.setAutoCommit(false);

    psPessoa.setString(1, pessoa.getNome());
    psPessoa.setString(2, pessoa.getEndereco());
    psPessoa.setString(3, pessoa.getTelefone());
    psPessoa.setString(4, pessoa.getEmail());
    psPessoa.setInt(5, pessoa.getId());
    psPessoa.executeUpdate();

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

    conn.commit();
} catch (SQLException e) {
    e.printStackTrace();
    throw e;
}
}

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

    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement psPessoaFisica =
conn.prepareStatement(sqlPessoaFisica);
        PreparedStatement psPessoa = conn.prepareStatement(sqlPessoa)) {
        conn.setAutoCommit(false);

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

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

        conn.commit();
    } catch (SQLException e) {
        e.printStackTrace();
        throw e;
    }
}
}

```

PessoaJuridica.java

```

package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

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

    // Construtor completo
    public PessoaJuridica(int id, String nome, String endereco, String telefone, String
email, String cnpj) {
        super(id, nome, endereco, telefone, email);
        this.cnpj = cnpj;
    }

    // Getter e Setter
    public String getCnpj() {
        return cnpj;
    }

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

    // Método exibir
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + cnpj);
        System.out.println("-----");
    }
}

```

PessoaJuridicaDAO.java

```

package cadastrobd.model;

import cadastrobd.util.ConectorBD;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

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

```



```

        String sql = "SELECT p.id_pessoa, p.nome, p.endereco, p.telefone, p.email,
pj.cnpj " +
        "FROM Pessoa p INNER JOIN PessoaJuridica pj ON p.id_pessoa =
pj.id_pessoa WHERE p.id_pessoa = ?";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return new PessoaJuridica(
                        rs.getInt("id_pessoa"),
                        rs.getString("nome"),
                        rs.getString("endereco"),
                        rs.getString("telefone"),
                        rs.getString("email"),
                        rs.getString("cnpj")
                    );
                }
            }
        }
        return null;
    }
}

```

```

public List<PessoaJuridica> getPessoas() throws SQLException {
    List<PessoaJuridica> pessoas = new ArrayList<>();
    String sql = "SELECT p.id_pessoa, p.nome, p.endereco, p.telefone, p.email,
pj.cnpj " +
    "FROM Pessoa p INNER JOIN PessoaJuridica pj ON p.id_pessoa =
pj.id_pessoa";
    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            pessoas.add(new PessoaJuridica(
                rs.getInt("id_pessoa"),
                rs.getString("nome"),
                rs.getString("endereco"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("cnpj")
            ));
        }
    }
    return pessoas;
}

```

```

public void incluir(PessoaJuridica pessoa) throws SQLException {

```

```

        String sqlPessoa = "INSERT INTO Pessoa (id_pessoa, nome, tipo, endereco,
telefone, email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlPessoaJuridica = "INSERT INTO PessoaJuridica (id_pessoa, cnpj)
VALUES (?, ?)";
        pessoa.setTipo("J");
        try (Connection conn = ConectorBD.getConnection());
            PreparedStatement psPessoa = conn.prepareStatement(sqlPessoa);
            PreparedStatement psPessoaJuridica =
conn.prepareStatement(sqlPessoaJuridica)) {
            conn.setAutoCommit(false);

            psPessoa.setInt(1, pessoa.getId());
            psPessoa.setString(2, pessoa.getNome());
            psPessoa.setString(3, pessoa.getTipo());
            psPessoa.setString(4, pessoa.getEndereco());
            psPessoa.setString(5, pessoa.getTelefone());
            psPessoa.setString(6, pessoa.getEmail());
            psPessoa.executeUpdate();

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

            conn.commit();
        } catch (SQLException e) {
            e.printStackTrace();
            throw e;
        }
    }

    public void alterar(PessoaJuridica pessoa) throws SQLException {
        String sqlPessoa = "UPDATE Pessoa SET nome = ?, endereco = ?, telefone = ?,
email = ? WHERE id_pessoa = ?";
        String sqlPessoaJuridica = "UPDATE PessoaJuridica SET cnpj = ? WHERE
id_pessoa = ?";

        try (Connection conn = ConectorBD.getConnection());
            PreparedStatement psPessoa = conn.prepareStatement(sqlPessoa);
            PreparedStatement psPessoaJuridica =
conn.prepareStatement(sqlPessoaJuridica)) {
            conn.setAutoCommit(false);

            psPessoa.setString(1, pessoa.getNome());
            psPessoa.setString(2, pessoa.getEndereco());
            psPessoa.setString(3, pessoa.getTelefone());
            psPessoa.setString(4, pessoa.getEmail());
            psPessoa.setInt(5, pessoa.getId());
            psPessoa.executeUpdate();

```

```

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

        conn.commit();
    } catch (SQLException e) {
        e.printStackTrace();
        throw e;
    }
}

public void excluir(int id) throws SQLException {
    String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE id_pessoa
= ?";
    String sqlPessoa = "DELETE FROM Pessoa WHERE id_pessoa = ?";

    try (Connection conn = ConectorBD.getConnection();
        PreparedStatement psPessoaJuridica
        = conn.prepareStatement(sqlPessoaJuridica);
        PreparedStatement psPessoa = conn.prepareStatement(sqlPessoa)) {
        conn.setAutoCommit(false);

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

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

        conn.commit();
    } catch (SQLException e) {
        e.printStackTrace();
        throw e;
    }
}
}

```

ConectorBD.java

```

package cadastrbd.util;

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

```

```

public class ConectorBD {
    private          static          final          String          URL          =
"jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertifica
te=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(String sql) throws SQLException {
        Connection conn = getConnection();
        return conn.prepareStatement(sql);
    }

    public static ResultSet getSelect(String sql) throws SQLException {
        Connection conn = getConnection();
        Statement stmt = conn.createStatement();
        return stmt.executeQuery(sql);
    }

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

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

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

```

```

    }
  }
}

```

SequenceManager.java

```

package cadastrobd.util;

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

public class SequenceManager {
    public static int getValue(String sequenceName) throws SQLException {
        String sql = "SELECT NEXT VALUE FOR " + sequenceName;
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                return rs.getInt(1);
            } else {
                throw new SQLException("Não foi possível obter o próximo valor da
sequência.");
            }
        }
    }
}

```

CadastroBDTeste.java

```

package cadastrobd;

import cadastrobd.model.*;
import cadastrobd.util.SequenceManager;

import java.sql.SQLException;
import java.util.List;

public class CadastroBDTeste {
    static String sequencialID = "s_pessoa_id";
    public static void main(String[] args) {
        try {
            // Instanciar e persistir uma pessoa física
            PessoaFisica pessoaFisica = new PessoaFisica(
                SequenceManager.getValue(sequencialID),
                "João Silva",

```

```

        "Rua A, 123",
        "123456789",
        "joao.silva@example.com",
        "12122233234"
    );
    PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
    pessoaFisicaDAO.incluir(pessoaFisica);
    System.out.println("Pessoa Física incluída com sucesso.");

    // Alterar os dados da pessoa física
    pessoaFisica.setNome("João Silva Alterado");
    pessoaFisica.setEmail("joao.silva.alterado@example.com");
    pessoaFisicaDAO.alterar(pessoaFisica);
    System.out.println("Pessoa Física alterada com sucesso.");
    System.out.println("-----");
    // Consultar todas as pessoas físicas e listar no console
    List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
    System.out.println("Lista de Pessoas Físicas:");
    for (PessoaFisica pf : pessoasFisicas) {
        pf.exibir();
    }

    // Excluir a pessoa física criada anteriormente
    pessoaFisicaDAO.excluir(pessoaFisica.getId());
    System.out.println("Pessoa Física excluída com sucesso.");

    // Instanciar e persistir uma pessoa jurídica
    PessoaJuridica pessoaJuridica = new PessoaJuridica(
        SequenceManager.getValue(sequencialID),
        "Empresa XYZ",
        "Rua B, 456",
        "987654321",
        "contato@xyz.com",
        "18345656000199"
    );
    PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();
    pessoaJuridicaDAO.incluir(pessoaJuridica);
    System.out.println("Pessoa Jurídica incluída com sucesso.");

    // Alterar os dados da pessoa jurídica
    pessoaJuridica.setNome("Empresa XYZ Alterada");
    pessoaJuridica.setEmail("contato.alterado@xyz.com");
    pessoaJuridicaDAO.alterar(pessoaJuridica);
    System.out.println("Pessoa Jurídica alterada com sucesso.");

    // Consultar todas as pessoas jurídicas e listar no console
    List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();
    System.out.println("Lista de Pessoas Jurídicas:");

```

```

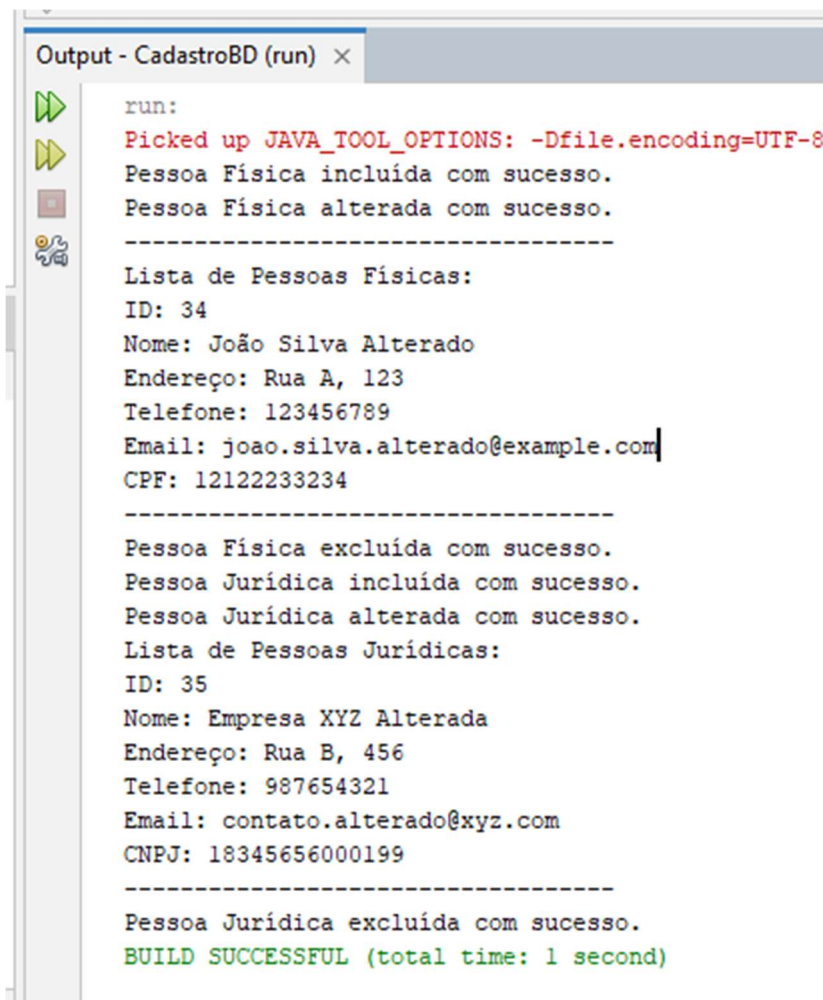
        for (PessoaJuridica pj : pessoasJuridicas) {
            pj.exibir();
        }

        // Excluir a pessoa jurídica criada anteriormente
        pessoaJuridicaDAO.excluir(pessoaJuridica.getId());
        System.out.println("Pessoa Jurídica excluída com sucesso.");

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

```

Resultados da Execução:



```

Output - CadastroBD (run) x
run:
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Pessoa Física incluída com sucesso.
Pessoa Física alterada com sucesso.
-----
Lista de Pessoas Físicas:
ID: 34
Nome: João Silva Alterado
Endereço: Rua A, 123
Telefone: 123456789
Email: joao.silva.alterado@example.com
CPF: 12122233234
-----
Pessoa Física excluída com sucesso.
Pessoa Jurídica incluída com sucesso.
Pessoa Jurídica alterada com sucesso.
Lista de Pessoas Jurídicas:
ID: 35
Nome: Empresa XYZ Alterada
Endereço: Rua B, 456
Telefone: 987654321
Email: contato.alterado@xyz.com
CNPJ: 18345656000199
-----
Pessoa Jurídica excluída com sucesso.
BUILD SUCCESSFUL (total time: 1 second)

```

Análise e Conclusão:

1. Qual a importância dos componentes de middleware, como o JDBC?

Os componentes de middleware, como o JDBC (Java Database Connectivity), são essenciais para a comunicação entre a aplicação Java e o banco de dados. Eles oferecem uma interface padronizada para interagir com diferentes sistemas de gerenciamento de banco de dados (SGBDs), abstraindo detalhes de implementação e permitindo que desenvolvedores escrevam código que pode ser facilmente adaptado a diferentes bancos de dados. O JDBC é responsável por estabelecer a conexão, enviar comandos SQL, e processar os resultados, facilitando a integração da aplicação com o banco de dados e promovendo a portabilidade do código.

2. Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?

A principal diferença entre Statement e PreparedStatement está na segurança e eficiência:

- **Statement:** Permite a execução de instruções SQL diretamente, mas é mais suscetível a ataques de injeção de SQL, uma vez que o SQL é construído como uma string. Além disso, Statement é menos eficiente para consultas repetitivas, pois o SQL precisa ser compilado toda vez que é executado.
- **PreparedStatement:** Este oferece mais segurança contra injeção de SQL, pois permite a parametrização das consultas, evitando que entradas de usuários possam manipular a estrutura da SQL. Além disso, PreparedStatement pode ser pré-compilado pelo banco de dados, o que melhora a performance em consultas que são executadas repetidamente.

3. Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao separar a lógica de acesso ao banco de dados da lógica de negócios. Isso cria uma camada de abstração que facilita a modificação e manutenção do código, pois as mudanças no banco de dados (como troca de SGBD ou modificação da estrutura) podem ser feitas no DAO sem impactar o restante da aplicação. Além disso, a reutilização do código é promovida, e o código torna-se mais fácil de testar e de entender.

4. Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

Em um modelo estritamente relacional, a herança pode ser refletida de diferentes maneiras, sendo as mais comuns:

- **Tabela Única (Single Table):** Todos os campos das classes base e derivadas são armazenados em uma única tabela. Isso simplifica a estrutura do banco, mas pode levar a muitos campos nulos se as subclasses tiverem muitos atributos diferentes.
- **Tabelas por Subclasse (Table per Subclass):** Cada classe derivada tem sua própria tabela, que contém apenas os campos específicos daquela subclasse. A tabela da classe base contém os campos comuns. As consultas frequentemente precisam fazer junções entre as tabelas, o que pode afetar a performance.
- **Tabela por Classe Concreta (Table per Concrete Class):** Cada classe concreta (não abstrata) tem sua própria tabela com todos os campos, incluindo os herdados. Isso evita junções, mas pode levar à duplicação de dados.

Na prática realizada, adotou-se uma abordagem que refletiu a herança através de tabelas separadas para PessoaFisica e PessoaJuridica, ambas referenciando a tabela Pessoa. Essa abordagem equilibra a normalização do banco de dados com a flexibilidade necessária para modelar entidades distintas, mantendo a coerência e a integridade dos dados.