Nome do campus: 3366 Polo Centro – Garopaba - SC.

Nome do curso: Desenvolvimento Full Stack.

Nome da disciplina: Back-end Sem Banco Não Tem.

Número da turma: 9001.

Nome: Daniel dos Santos Pereira.

Endereço: DaniielDev/Mapeamento-Objeto-Relacional-e-DAO

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

- Intermediação entre aplicação e banco de dados: O JDBC atua como um middleware que permite que aplicações Java comuniquem-se com diferentes sistemas gerenciadores de banco de dados via uma API comum.
- Portabilidade: Permite que o código Java para acesso a dados seja escrito de forma independente do SGBD subjacente, facilitando troca ou atualização da base.
- Abstração e simplificação: Oferece uma interface de alto nível para executar comandos SQL, gerenciar conexões, tratar transações, sem que o desenvolvedor precise se preocupar com detalhes específicos do protocolo do banco.
- Gerenciamento de recursos e segurança: Facilitam o controle de conexões, otimização de acesso e evitam vazamentos de recursos, aumentando robustez da aplicação.

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

- Statement: Executa comandos SQL simples diretamente e Suscetível a SQL Injection se concatenar strings.
- PreparedStatement: Executa comandos SQL parametrizados e Reduz riscos, pois usa parâmetros e não concatena.

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

- Separação de responsabilidades: DAO abstrai toda a lógica de acesso a dados, separando-a da lógica de negócio e da interface.
- Menor acoplamento: Alterações no banco ou nas queries afetam somente as classes DAO, sem impactar outras camadas.
- Facilita testes unitários: Permite isolamento da camada de dados para testes e mock.
- Reutilização: DAO pode ser reutilizado em vários pontos da aplicação.
- Padronização e organização: Câmara de trabalhos para CRUD, parâmetros e tratamento de exceções padronizados, simplificando a evolução do projeto.

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

- Implementação por tabelas relacionadas (tabelas separadas): O padrão mais comum é criar uma tabela para a superclasse (Pessoa) e tabelas adicionais para cada subclasse (PessoaFisica, PessoaJuridica), onde as tabelas filhas referenciam a tabela pai via chave estrangeira.
- Chave primária compartilhada: Usar o mesmo identificador nas tabelas filhas para indicar que o registro é uma extensão daquele da tabela pai.
- Normalização: Essa abordagem evita redundância, mantém integridade e permite consultas polimórficas.
- Compromisso com relacionamentos: A consulta para uma subclasse normalmente exige join entre as tabelas pai e filha, pois as colunas estão distribuídas.

Código:

CadastroBD/

├── cadastrobd/

│ └── CadastroBDTeste.java

├── cadastrobd/model/

│ ├── Pessoa.java

│ ├── PessoaFisica.java

│ ├── PessoaJuridica.java

│ ├── PessoaFisicaDAO.java

│ └── PessoaJuridicaDAO.java

└── cadastrobd/model/util/

    ├── ConectorBD.java

    └── SequenceManager.java

```
package cadastrobd.model;


public class Pessoa {
protected int id;     protected
String nome;     protected
String logradouro;     protected
String cidade;     protected
String estado;     protected
String telefone;     protected
String email;


   public Pessoa() {}


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


   public 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("Email: " + email);
    }
}
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    public PessoaFisica() {}

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

    public String getCpf() {
return cpf;
    }

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

```
    @Override
    public void exibir() {
super.exibir();
        System.out.println("CPF: " + cpf);
    }
}
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {}

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

    public String getCnpj() {
return cnpj;
    }

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

    @Override
```

```
    public void exibir() {
super.exibir();
        System.out.println("CNPJ: " + cnpj);
    }
}

package cadastrobd.model.util;

import java.sql.*;

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(Connection conn, String sql) throws
SQLException {         return conn.prepareStatement(sql);
    }

    public static ResultSet getSelect(Connection conn, String sql) throws SQLException {
return getPrepared(conn, sql).executeQuery();
    }

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

```
    }

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

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

package cadastrobd.model.util;

import java.sql.*;

public class SequenceManager {     public static int
getValue(String sequenceName) {
        int value = -1;
        Connection conn = null;
        PreparedStatement stmt = null;
ResultSet rs = null;
        try {
            conn = ConectorBD.getConnection();            stmt =
conn.prepareStatement("SELECT NEXT VALUE FOR " + sequenceName);
            rs = stmt.executeQuery();
            if (rs.next()) {
value = rs.getInt(1);            }

        } catch (SQLException e) {
```

```
            e.printStackTrace();
        } finally {
            ConectorBD.close(rs);
            ConectorBD.close(stmt);
            ConectorBD.close(conn);
        }
        return value;
    }
}

package cadastrobd.model;

import cadastrobd.model.util.*;
import java.sql.*; import
java.util.ArrayList;

public class PessoaFisicaDAO {

    public PessoaFisica getPessoa(int id) {
PessoaFisica pf = null;
        try (Connection conn = ConectorBD.getConnection()) {
            String sql = "SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.id = pf.id
WHERE p.id = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                pf = new PessoaFisica(                    rs.getInt("id"), rs.getString("nome"),
rs.getString("logradouro"),                  rs.getString("cidade"),
```

```
rs.getString("estado"), rs.getString("telefone"),                rs.getString("email"),
rs.getString("cpf"));
          }
      } catch (SQLException e) {
          e.printStackTrace();
      }
      return pf;
  }

  public ArrayList<PessoaFisica> getPessoas() {
ArrayList<PessoaFisica> lista = new ArrayList<>();        try
(Connection conn = ConectorBD.getConnection()) {
          String sql = "SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.id = pf.id";
ResultSet rs = ConectorBD.getSelect(conn, sql);
          while (rs.next()) {
              PessoaFisica pf = new PessoaFisica(                rs.getInt("id"),
rs.getString("nome"), rs.getString("logradouro"),
rs.getString("cidade"), rs.getString("estado"), rs.getString("telefone"),
rs.getString("email"), rs.getString("cpf"));
              lista.add(pf);
          }
      } catch (SQLException e) {
          e.printStackTrace();
      }
      return lista;
  }

  public void incluir(PessoaFisica pf) {        int id =
SequenceManager.getValue("seq_id");
```

```
        pf.id = id;
        try (Connection conn = ConectorBD.getConnection()) {
            PreparedStatement stmt1 = conn.prepareStatement("INSERT INTO Pessoa
VALUES (?, ?, ?, ?, ?, ?, ?)");           stmt1.setInt(1, id);           stmt1.setString(2,
pf.nome);           stmt1.setString(3, pf.logradouro);           stmt1.setString(4,
pf.cidade);           stmt1.setString(5, pf.estado);           stmt1.setString(6,
pf.telefone);           stmt1.setString(7, pf.email);           stmt1.executeUpdate();

            PreparedStatement stmt2 = conn.prepareStatement("INSERT INTO PessoaFisica
VALUES (?, ?)");
            stmt2.setInt(1, id);
stmt2.setString(2, pf.getCpf());
stmt2.executeUpdate();        } catch
(SQLException e) {
            e.printStackTrace();
        }
    }

    public void alterar(PessoaFisica pf) {        try (Connection
conn = ConectorBD.getConnection()) {
            PreparedStatement stmt1 = conn.prepareStatement("UPDATE Pessoa SET
nome=?, logradouro=?, cidade=?, estado=?, telefone=?, email=? WHERE id=?");
            stmt1.setString(1, pf.nome);
            stmt1.setString(2, pf.logradouro);
stmt1.setString(3, pf.cidade);
stmt1.setString(4, pf.estado);
stmt1.setString(5, pf.telefone);
stmt1.setString(6, pf.email);           stmt1.setInt(7,
pf.id);           stmt1.executeUpdate();
```

```
        PreparedStatement stmt2 = conn.prepareStatement("UPDATE PessoaFisica SET
cpf=? WHERE id=?");
        stmt2.setString(1,
pf.getCpf());          stmt2.setInt(2,
pf.id);          stmt2.executeUpdate();
} catch (SQLException e) {
        e.printStackTrace();
    }
  }

  public void excluir(int id) {
    try (Connection conn = ConectorBD.getConnection()) {
        PreparedStatement stmt1 = conn.prepareStatement("DELETE FROM
PessoaFisica WHERE id=?");
        stmt1.setInt(1, id);
stmt1.executeUpdate();

        PreparedStatement stmt2 = conn.prepareStatement("DELETE FROM Pessoa
WHERE id=?");
        stmt2.setInt(1, id);
stmt2.executeUpdate();       }
catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
package cadastrobd;

import cadastrobd.model.*;
import java.util.ArrayList;
```

```
public class CadastroBDTeste {     public
static void main(String[] args) {
        PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();
        PessoaFisica pf = new PessoaFisica(0, "Maria Oliveira", "Rua B", "São Paulo", "SP",
"2222-2222", "maria@email.com", "98765432100");

        System.out.println("Incluindo pessoa física...");
pfDAO.incluir(pf);

        System.out.println("Alterando nome...");
pf.setNome("Maria Oliveira da Silva");        pfDAO.alterar(pf);

        System.out.println("Listando pessoas físicas:");
ArrayList<PessoaFisica> lista = pfDAO.getPessoas();
        for (PessoaFisica p : lista) {
           p.exibir();
           System.out.println("-----");
        }

        System.out.println("Excluindo pessoa física...");
pfDAO.excluir(pf.id);
    }
}
```

Procedimento 2:

1. **Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?**

- Persistência em Arquivo: Dados armazenados de forma simples (texto, binário) sem esquema rigoroso.

- Persistência em Banco de Dados: Dados organizados em tabelas com esquema definido (modelo relacional ou outro).

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

- O operador lambda introduzido no Java 8 permite escrever código funcional e mais conciso, substituindo estruturas verbosas como loops for ou iteradores explícitos.

3. **Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static?**
   - O método main é o ponto de entrada do programa e é chamado pela JVM sem criar antes uma instância da classe.

   - Métodos não estáticos (instance methods) precisam de um objeto para serem invocados.

   - Já métodos static pertencem à classe (e não a um objeto), logo podem ser chamados diretamente dentro do main sem a necessidade de instanciar um objeto .
   - Portanto, para que o main possa chamar métodos auxiliares sem criar objetos, esses métodos precisam ser definidos como static.

   Código:

```
package cadastrobd;

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

public class CadastroBDTeste {
   private static Scanner scanner = new Scanner(System.in);     private
static PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();     private
static PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();

   public static void main(String[] args) {
      int opcao;
do {
         System.out.println("\n=== MENU PRINCIPAL ===");
```

```
            System.out.println("1 - Incluir");
            System.out.println("2 - Alterar");
            System.out.println("3 - Excluir");
            System.out.println("4 - Exibir por ID");
            System.out.println("5 - Exibir todos");
            System.out.println("0 - Sair");
            System.out.print("Escolha uma opção: ");
            opcao = Integer.parseInt(scanner.nextLine());
             try {               switch
(opcao) {                   case 1 ->
incluir();                  case 2 ->
alterar();                  case 3 ->
excluir();                  case 4 ->
exibirPorId();                    case 5 ->
exibirTodos();
                  case 0 -> System.out.println("Saindo do sistema...");
default -> System.out.println("Opção inválida.");
                }
            } catch (Exception e) {
               System.out.println("Erro: " + e.getMessage());
e.printStackTrace();
            }
        } while (opcao != 0);
    }

    private static void incluir() {
        System.out.print("Pessoa Física (F) ou Jurídica (J)?
");        String tipo = scanner.nextLine().toUpperCase();
if (tipo.equals("F")) {
            PessoaFisica pf = lerPessoaFisica();
            pfDAO.incluir(pf);
            System.out.println("Pessoa física incluída com sucesso!");
        } else if (tipo.equals("J")) {
            PessoaJuridica pj = lerPessoaJuridica();
            pjDAO.incluir(pj);
            System.out.println("Pessoa jurídica incluída com sucesso!");
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    private static void alterar() {
        System.out.print("Pessoa Física (F) ou Jurídica (J)? ");
```

```
        String tipo = scanner.nextLine().toUpperCase();
System.out.print("ID: ");
        int id = Integer.parseInt(scanner.nextLine());

        if (tipo.equals("F")) {
            PessoaFisica pf = pfDAO.getPessoa(id);
            if (pf != null) {
                System.out.println("Dados atuais:");
                pf.exibir();
                PessoaFisica novaPf = lerPessoaFisica();
                novaPf.setId(id);
pfDAO.alterar(novaPf);
                System.out.println("Pessoa física alterada com sucesso!");
            } else {
                System.out.println("Pessoa física não encontrada.");
            }
        } else if (tipo.equals("J")) {
            PessoaJuridica pj = pjDAO.getPessoa(id);
            if (pj != null) {
                System.out.println("Dados atuais:");
                pj.exibir();
                PessoaJuridica novaPj = lerPessoaJuridica();
                novaPj.setId(id);
pjDAO.alterar(novaPj);
                System.out.println("Pessoa jurídica alterada com sucesso!");
            } else {
                System.out.println("Pessoa jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    private static void excluir() {
        System.out.print("Pessoa Física (F) ou Jurídica (J)? ");
        String tipo = scanner.nextLine().toUpperCase();
System.out.print("ID: ");
        int id = Integer.parseInt(scanner.nextLine());

        if (tipo.equals("F")) {
pfDAO.excluir(id);
            System.out.println("Pessoa física excluída com sucesso!");
```

```
        } else if (tipo.equals("J")) {
pjDAO.excluir(id);
            System.out.println("Pessoa jurídica excluída com sucesso!");
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    private static void exibirPorId() {
        System.out.print("Pessoa Física (F) ou Jurídica (J)? ");
        String tipo = scanner.nextLine().toUpperCase();
System.out.print("ID: ");
        int id = Integer.parseInt(scanner.nextLine());

        if (tipo.equals("F")) {
            PessoaFisica pf = pfDAO.getPessoa(id);
            if (pf != null) {
pf.exibir();
            } else {
                System.out.println("Pessoa física não encontrada.");
            }
        } else if (tipo.equals("J")) {
            PessoaJuridica pj = pjDAO.getPessoa(id);
            if (pj != null) {
pj.exibir();
            } else {
                System.out.println("Pessoa jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    private static void exibirTodos() {
        System.out.print("Pessoa Física (F) ou Jurídica (J)? ");
        String tipo = scanner.nextLine().toUpperCase();

        if (tipo.equals("F")) {
            List<PessoaFisica> listaPf = pfDAO.getPessoas();
            if (listaPf.isEmpty()) {
                System.out.println("Nenhuma pessoa física cadastrada.");
            } else {
                for (PessoaFisica pf : listaPf) {
```

```
                pf.exibir();
            }
        }
    } else if (tipo.equals("J")) {
        List<PessoaJuridica> listaPj = pjDAO.getPessoas();
        if (listaPj.isEmpty()) {
            System.out.println("Nenhuma pessoa jurídica cadastrada.");
        } else {
            for (PessoaJuridica pj : listaPj) {
                pj.exibir();
            }
        }
    } else {
        System.out.println("Tipo inválido.");
    }
}

private static PessoaFisica lerPessoaFisica() {
    System.out.print("Nome: ");
    String nome = scanner.nextLine();
    System.out.print("Logradouro: ");
    String logradouro = scanner.nextLine();
    System.out.print("Cidade: ");
    String cidade = scanner.nextLine();
    System.out.print("Estado: ");
    String estado = scanner.nextLine();
    System.out.print("Telefone: ");
    String telefone = scanner.nextLine();
    System.out.print("Email: ");
    String email = scanner.nextLine();
    System.out.print("CPF: ");
String cpf = scanner.nextLine();
    return new PessoaFisica(0, nome, logradouro, cidade, estado, telefone, email,
cpf);
}

private static PessoaJuridica lerPessoaJuridica() {
    System.out.print("Nome: ");
    String nome = scanner.nextLine();
    System.out.print("Logradouro: ");
    String logradouro = scanner.nextLine();
    System.out.print("Cidade: ");
    String cidade = scanner.nextLine();
```

```
        System.out.print("Estado: ");
        String estado = scanner.nextLine();
        System.out.print("Telefone: ");
        String telefone = scanner.nextLine();
        System.out.print("Email: ");
        String email = scanner.nextLine();
        System.out.print("CNPJ: ");
String cnpj = scanner.nextLine();
        return new PessoaJuridica(0, nome, logradouro, cidade, estado, telefone,
email, cnpj);
    }
}
```