Missão Prática | Nível 3 | Mundo 3

Bernardo Oliveira Ramos -202307212041

Polo Silva Lobo
Desenvolvedor Full stack – mundo 3 período 2024.3 – Semestre Letivo

Objetivo da Prática

Descreva nessa seção qual o objetivo da sua prática. Todos os Relatórios de Práticas deverão ser confeccionados em arquivo no formato PDF, com a **Logo da Universidade**, **nome do Campus**, **nome do Curso**, **nome da Disciplina**, **número da Turma**, **semestre letivo**. Além disso, o projeto deve ser armazenado em um repositório no GIT e o respectivo endereço deve constar na documentação e essa documentação deve estar no no GIT. O código deve estar versionado no GIT de forma organizada.

Lembre-se que a organização contará pontos.

Esse template é um modelo a ser seguido. O aluno pode optar por seguir outro modelo, **desde que atenda a todas as etapas disponíveis na Missão Prática**. O documento final deve estar em pdf.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 1º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e

Classe Pessoa

```
package cadastrobd.model;

public class Pessoa {
   protected int id;
   protected String nome, logradouro, cidade, estado, telefone, 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 + ", Nome: " + nome);
    }
}
```

Classe PessoaFisica

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    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;
    }

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

Classe PessoaJuridica

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    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;
    }

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

```
    }
}
```

Criação de Classes Utilitárias (Pacote cadastrobd.model.util)

Classe ConectorBD

```
package cadastrobd.model.util;

import java.sql.*;

public class ConectorBD {
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(

"jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate
=true;",
                "user", "password");
    }

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

    public static void close(AutoCloseable obj) {
        try {
            if (obj != null) obj.close();
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

5. Implementação do DAO (Pacote cadastrobd.model)

Classe PessoaFisicaDAO

```
package cadastrobd.model;

import cadastrobd.model.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 * FROM PessoaFisica WHERE id = ?";
        PreparedStatement stmt = ConectorBD.getPrepared(sql);
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            return 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"));
        }
```

```
        return null;
    }

    public List<PessoaFisica> getPessoas() throws SQLException {
        List<PessoaFisica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM PessoaFisica";
        ResultSet rs = ConectorBD.getPrepared(sql).executeQuery();

        while (rs.next()) {
            pessoas.add(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")));
        }
        return pessoas;
    }
}
```

Aqui está um passo a passo detalhado para seguir o procedimento que você descreveu. Esse projeto envolve a criação de um sistema simples em Java usando NetBeans e SQL Server. Vamos organizar o processo por etapas para facilitar a implementação.

---

1. Criação do Projeto e Configuração do Driver JDBC

1. Abra o **NetBeans** e crie um novo projeto:
    - **Nome**: CadastroBD
    - **Tipo**: Aplicativo Java Padrão (modelo Ant).
2. Adicione o driver JDBC ao projeto:

- Baixe o **Microsoft JDBC Driver** neste link.
- Extraia o arquivo .zip e encontre o mssql-jdbc-12.2.0.jre11.jar.

3. No NetBeans:
   - Clique com o botão direito em **Libraries** (Bibliotecas).
   - Escolha **Add JAR/Folder** e selecione o arquivo .jar do passo anterior.
   - Verifique se o driver foi reconhecido corretamente.

---

2. Configuração da Conexão com o Banco de Dados

1. Na aba **Serviços → Banco de Dados**, clique com o botão direito em **Drivers** e selecione **Novo Driver**.
2. Defina o **driver SQL Server**.
3. Configure a conexão:
   - **JDBC URL**:

bash

Copiar código

jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate=true;

   - **Database**: loja
   - **User**: <seu usuário>
   - **Password**: <sua senha>
   - Marque **Lembrar senha**.
4. Teste a conexão e finalize.

---

3. Criação das Classes Model (Pacote cadastrobd.model)

Classe Pessoa

java

Copiar código

```
package cadastrobd.model;


public class Pessoa {
    protected int id;
    protected String nome, logradouro, cidade, estado, telefone, 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 + ", Nome: " + nome);
    }
}
```

Classe PessoaFisica

java

Copiar código

```
package cadastrobd.model;


public class PessoaFisica extends Pessoa {
    private String cpf;


    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;
    }


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

Classe PessoaJuridica

java

Copiar código

```
package cadastrobd.model;


public class PessoaJuridica extends Pessoa {
    private String cnpj;
```

```
    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;
    }

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

---

4. Criação de Classes Utilitárias (Pacote cadastrobd.model.util)

Classe ConectorBD

java

Copiar código

```
package cadastrobd.model.util;

import java.sql.*;

public class ConectorBD {
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(
```

```
"jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate
=true;",
            "user", "password");
    }

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

    public static void close(AutoCloseable obj) {
        try {
            if (obj != null) obj.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

---

5. Implementação do DAO (Pacote cadastrobd.model)

Classe PessoaFisicaDAO

java

Copiar código

```
package cadastrobd.model;

import cadastrobd.model.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 * FROM PessoaFisica WHERE id = ?";
        PreparedStatement stmt = ConectorBD.getPrepared(sql);
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            return 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"));
        }
        return null;
    }

    public List<PessoaFisica> getPessoas() throws SQLException {
        List<PessoaFisica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM PessoaFisica";
        ResultSet rs = ConectorBD.getPrepared(sql).executeQuery();

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

```
            pessoas.add(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")));
        }
        return pessoas;
    }
}
```

Conclusão:

a) **Qual a importância dos componentes de middleware, como o JDBC?** Olha, o middleware é meio que um "tradutor" que ajuda sistemas diferentes a se comunicarem. No caso do JDBC, ele serve como uma ponte entre o Java e o banco de dados. Sem ele, seria muito mais complicado trocar informações, porque cada banco tem sua própria linguagem e jeito de fazer as coisas. O JDBC deixa tudo mais padronizado e facilita o envio de consultas e a manipulação dos dados no banco, sem a gente se preocupar com os detalhes técnicos de cada sistema de banco que usamos.Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?

b) **Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?**
Então, a principal diferença é que o Statement é mais simples e direto, mas não é muito seguro quando falamos de SQL Injection (um tipo de ataque no banco). Já o PreparedStatement é um pouco mais chato de configurar, mas ele compensa porque é mais seguro e eficiente. Ele pré-compila a query e permite reutilizar o mesmo comando várias vezes com dados diferentes, o que acaba sendo melhor quando você precisa executar muitas operações parecidas.Como a herança é

refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

c) **Como o padrão DAO melhora a manutenibilidade do software?**
O padrão DAO (Data Access Object) separa bem as responsabilidades no código. Basicamente, ele cria uma camada específica para lidar com as interações com o banco de dados. Isso facilita a vida porque, se um dia você precisar trocar o banco ou mudar a forma como acessa os dados, só precisa mexer nessa parte. Além disso, o resto do código não precisa saber como o banco funciona, o que deixa tudo mais organizado e mais fácil de dar manutenção.

d) **Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?**
Ah, essa parte é interessante porque bancos relacionais não foram feitos pensando diretamente em herança, então a gente precisa adaptar. Existem algumas maneiras de fazer isso. Uma é criar uma tabela pra cada classe (tipo, uma pra classe-mãe e outras pras filhas) e usar chaves estrangeiras pra conectar tudo. Outra forma é juntar tudo numa tabela só e colocar uma coluna que indica qual tipo de objeto cada linha representa. Tem também o jeito de usar tabelas separadas só para as subclasses, mas cada abordagem tem seus prós e contras dependendo do que você quer priorizar: simplicidade, performance ou facilidade de manutenção.

2º Procedimento | Alimentando a Base

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 2º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e

//Código Atualizado da Classe CadastroBDTeste

```
package cadastrobd;

import cadastrobd.model.*;
```

```
import cadastrobd.model.util.ConectorBD;
import java.sql.SQLException;
import java.util.Scanner;

public class CadastroBDTeste {

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

        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 pelo ID");
            System.out.println("5 - Exibir Todos");
            System.out.println("0 - Sair");
            System.out.print("Escolha uma opção: ");
            opcao = scanner.nextInt();
            scanner.nextLine(); // Limpa o buffer do scanner

            try {
```

```
            switch (opcao) {
                case 1 -> incluir(scanner, pfDAO, pjDAO);
                case 2 -> alterar(scanner, pfDAO, pjDAO);
                case 3 -> excluir(scanner, pfDAO, pjDAO);
                case 4 -> exibirPorId(scanner, pfDAO, pjDAO);
                case 5 -> exibirTodos(scanner, pfDAO, pjDAO);
                case 0 -> System.out.println("Encerrando o sistema...");
                default -> System.out.println("Opção inválida! Tente novamente.");
            }
        } catch (SQLException e) {
            System.err.println("Erro: " + e.getMessage());
        }

    } while (opcao != 0);

    scanner.close();
}

    private static void incluir(Scanner scanner, PessoaFisicaDAO pfDAO,
PessoaJuridicaDAO pjDAO) throws SQLException {
    System.out.println("Incluir Pessoa Física (1) ou Jurídica (2)?");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpa o buffer

    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();

        if (tipo == 1) {
            System.out.print("CPF: ");
            String cpf = scanner.nextLine();
                PessoaFisica pf = new PessoaFisica(0, nome, logradouro, cidade, estado, telefone, email, cpf);
            pfDAO.incluir(pf);
            System.out.println("Pessoa Física incluída com sucesso!");
        } else {
            System.out.print("CNPJ: ");
            String cnpj = scanner.nextLine();
              PessoaJuridica pj = new PessoaJuridica(0, nome, logradouro, cidade, estado, telefone, email, cnpj);
            pjDAO.incluir(pj);
            System.out.println("Pessoa Jurídica incluída com sucesso!");
        }
```

```
    }

    private static void alterar(Scanner scanner, PessoaFisicaDAO pfDAO,
PessoaJuridicaDAO pjDAO) throws SQLException {
        System.out.println("Alterar Pessoa Física (1) ou Jurídica (2)?");
        int tipo = scanner.nextInt();
        scanner.nextLine();

        System.out.print("ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        if (tipo == 1) {
            PessoaFisica pf = pfDAO.getPessoa(id);
            if (pf != null) {
                System.out.println("Dados atuais:");
                pf.exibir();
                System.out.println("Digite os novos dados:");
                incluir(scanner, pfDAO, pjDAO);
                System.out.println("Pessoa Física alterada com sucesso!");
            } else {
                System.out.println("Pessoa Física não encontrada.");
            }
        } else {
            PessoaJuridica pj = pjDAO.getPessoa(id);
            if (pj != null) {
```

```
            System.out.println("Dados atuais:");
            pj.exibir();
            System.out.println("Digite os novos dados:");
            incluir(scanner, pfDAO, pjDAO);
            System.out.println("Pessoa Jurídica alterada com sucesso!");
        } else {
            System.out.println("Pessoa Jurídica não encontrada.");
        }
    }
}

    private static void excluir(Scanner scanner, PessoaFisicaDAO pfDAO,
PessoaJuridicaDAO pjDAO) throws SQLException {
    System.out.println("Excluir Pessoa Física (1) ou Jurídica (2)?");
    int tipo = scanner.nextInt();
    scanner.nextLine();

    System.out.print("ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();

    if (tipo == 1) {
        pfDAO.excluir(id);
        System.out.println("Pessoa Física excluída com sucesso!");
    } else {
        pjDAO.excluir(id);
```

```
            System.out.println("Pessoa Jurídica excluída com sucesso!");
        }
    }

    private static void exibirPorId(Scanner scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO pjDAO) throws SQLException {
        System.out.println("Exibir Pessoa Física (1) ou Jurídica (2)?");
        int tipo = scanner.nextInt();
        scanner.nextLine();

        System.out.print("ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        if (tipo == 1) {
            PessoaFisica pf = pfDAO.getPessoa(id);
            if (pf != null) {
                pf.exibir();
            } else {
                System.out.println("Pessoa Física não encontrada.");
            }
        } else {
            PessoaJuridica pj = pjDAO.getPessoa(id);
            if (pj != null) {
                pj.exibir();
            } else {
```

```
                System.out.println("Pessoa Jurídica não encontrada.");
            }
        }
    }

    private static void exibirTodos(Scanner scanner, PessoaFisicaDAO pfDAO, PessoaJuridicaDAO pjDAO) throws SQLException {
        System.out.println("Exibir todas as Pessoas Físicas (1) ou Jurídicas (2)?");
        int tipo = scanner.nextInt();
        scanner.nextLine();

        if (tipo == 1) {
            for (PessoaFisica pf : pfDAO.getPessoas()) {
                pf.exibir();
            }
        } else {
            for (PessoaJuridica pj : pjDAO.getPessoas()) {
                pj.exibir();
            }
        }
    }
}
```

Conclusão:

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

Então, a persistência em arquivo é bem direta: você salva os dados num formato como .txt, .csv ou até JSON. É simples e não precisa de um servidor ou configuração complexa, mas pode ser meio limitado se você tiver muitos dados ou precisar fazer consultas rápidas e organizadas. Já o banco de dados é mais robusto, com várias vantagens como segurança, controle de transações e facilidade para buscar e filtrar dados com SQL. A diferença principal é que os bancos são feitos para lidar com muitos dados de forma eficiente e organizada, enquanto os arquivos são uma solução mais simples e direta.Como o uso de operador *lambda* simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

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

Com os lambdas, o código fica bem mais enxuto. Antes, a gente tinha que criar uma classe anônima ou escrever um monte de código para iterar sobre uma lista e imprimir valores. Agora, com o lambda, você pode fazer algo assim:

```
minhaLista.forEach(item -> System.out.println(item));
```

Isso deixa o código mais limpo e fácil de entender. Além disso, o lambda ajuda a focar mais no *"o que fazer"* e menos no *"como fazer"*, simplificando bastante a escrita.

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

Então, o método main é estático, o que significa que ele pertence à classe, e não a uma instância dela. Quando você quer chamar outro método direto do main sem criar um objeto, esse outro método também precisa ser static, senão o compilador vai reclamar. Isso acontece porque métodos não estáticos precisam de um objeto para existir, e o main não cria um objeto automaticamente para você. Por isso, marcar como static é meio que dizer: “Esse método pode rodar sozinho, sem depender de uma instância.”

Conclusão

A Missão Prática de criação de um aplicativo Java com persistência em SQL Server via JDBC proporcionou uma experiência valiosa ao conectar teoria e prática. A implementação do padrão DAO e o uso do mapeamento objeto-relacional destacaram a importância de separar responsabilidades e manter o código organizado e escalável. Entre os principais desafios, estão a configuração do JDBC e a integração com o banco, que exigiram atenção aos detalhes. No geral, a atividade reforçou conceitos essenciais de persistência de dados, padrões de design e habilidades de depuração, além de evidenciar a relevância da prática para consolidar o aprendizado em programação orientada a objetos e backend.