

Modelo de Arquivo para as atividades em vídeo e GitHub em
PDF

| | |
|--------------------------------------|------------------------|
| Nome: | Timóteo Alves de Souza |
| Disciplina: | Padrões de Projetos |
| Turma: | 5º Período |
| Atividade referente a aula: | Trabalho Final |
| | |
| Linka da Atividade do Vídeo Parte 1: | |
| Linka da Atividade do Vídeo Parte 2: | |
| Linka da Atividade do Vídeo Parte 3: | |
| Linka da Atividade do Vídeo Parte 4: | |
| Linka da Atividade do Vídeo Parte 5: | |
| | |
| Link do GitHub: | |

Salvar o arquivo em PDF antes de enviar para o Moodle.

Relatório: Padrões de Projeto MVC e DAO

1. Introdução

Os padrões de projeto são soluções generalistas para problemas recorrentes no desenvolvimento de software. Eles permitem reutilizar experiências e boas práticas, tornando o código mais estruturado, legível e de fácil manutenção. Neste relatório, abordaremos dois dos padrões mais utilizados no desenvolvimento de software: o **MVC (Model-View-Controller)** e o **DAO (Data Access Object)**.

Ambos os padrões possuem finalidades específicas e são empregados em cenários diferentes, mas compartilham o objetivo de modularizar o código, separando responsabilidades e facilitando futuras manutenções.

2. Padrão MVC (Model-View-Controller)

Definição

O padrão **MVC** é utilizado para separar as preocupações em aplicações com interfaces de usuário, dividindo a aplicação em três componentes principais:

Model: Responsável pela lógica de negócios e manipulação de dados.

View: Exibe a interface gráfica e a apresentação dos dados.

Controller: Atua como intermediário, capturando as entradas do usuário e atualizando o Model e a View.

Usabilidade

O MVC é frequentemente utilizado em aplicações onde há interação entre o usuário e os dados, como em sistemas web, desktop ou mobile. A separação em camadas permite um desenvolvimento mais organizado e facilita a manutenção, já que a lógica de negócios, a apresentação de dados e o controle de fluxo ficam independentes entre si.

Vantagens

Manutenção facilitada: Separação clara entre as camadas facilita localizar e corrigir erros.

Reutilização: O mesmo Model pode ser utilizado por diferentes Views, facilitando o reaproveitamento.

Escalabilidade: A divisão em três partes torna o sistema mais escalável.

Desvantagens

Complexidade: Para sistemas simples, o MVC pode aumentar a complexidade, uma vez que há a necessidade de criar e gerenciar várias classes.

Curva de aprendizado: Pode ser desafiador para iniciantes compreender a interação entre as camadas.

Exemplo de MVC (Cadastro de Usuários)

O exemplo implementa um sistema de cadastro de usuários, onde o **Model** armazena os dados do usuário, a **View** exibe as informações e o **Controller** faz a comunicação entre os dois.

Código:

Usuario.java

```
Usuario.java X
MVC > Usuario.java > ...
1  public class Usuario {
2      private String nome;
3      private String email;
4
5      public Usuario(String nome, String email) {
6          this.nome = nome;
7          this.email = email;
8      }
9
10     public String getNome() {
11         return nome;
12     }
13
14     public void setNome(String nome) {
15         this.nome = nome;
16     }
17
18     public String getEmail() {
19         return email;
20     }
21
22     public void setEmail(String email) {
23         this.email = email;
24     }
25 }
26
```

UsuarioView.java

```
UsuarioView.java X
MVC > UsuarioView.java > ...
1  import java.util.Scanner;
2
3  public class UsuarioView {
4      private Scanner scanner = new Scanner(System.in);
5
6      public String getInput(String prompt) {
7          System.out.println(prompt);
8          return scanner.nextLine();
9      }
10
11     public void displayUsuarioInfo(String nome, String email) {
12         System.out.println("Nome: " + nome);
13         System.out.println("Email: " + email);
14     }
15 }
16
```

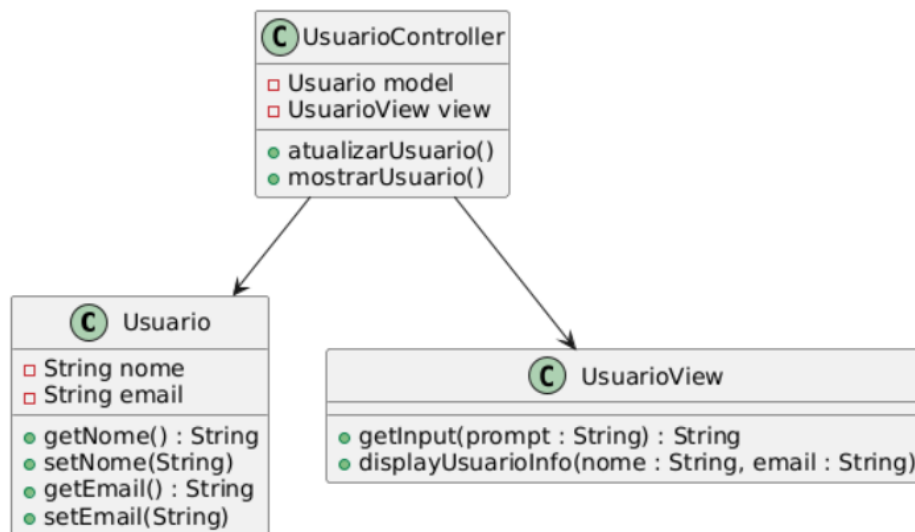
UsuarioController.java

```
UsuarioController.java X
MVC > UsuarioController.java > ...
1  public class UsuarioController {
2      private Usuario model;
3      private UsuarioView view;
4
5      public UsuarioController(Usuario model, UsuarioView view) {
6          this.model = model;
7          this.view = view;
8      }
9
10     public void atualizarUsuario() {
11         String nome = view.getInput(prompt:"Digite o nome do usuário:");
12         String email = view.getInput(prompt:"Digite o email do usuário:");
13         model.setNome(nome);
14         model.setEmail(email);
15     }
16
17     public void mostrarUsuario() {
18         view.displayUsuarioInfo(model.getNome(), model.getEmail());
19     }
20 }
21
```

Main.java

```
Main.java X
MVC > Main.java > ...
1  public class Main {
2      public static void main(String[] args) {
3          Usuario model = new Usuario(nome:"Desconhecido", email:"email@exemplo.com");
4          UsuarioView view = new UsuarioView();
5          UsuarioController controller = new UsuarioController(model, view);
6
7          // Atualizando dados do usuário
8          controller.atualizarUsuario();
9
10         // Mostrando informações atualizadas
11         controller.mostrarUsuario();
12     }
13 }
14
```

UML do Exemplo MVC:



3. Padrão DAO (Data Access Object)

Definição

O **DAO (Data Access Object)** é um padrão de projeto que abstrai e encapsula o acesso aos dados de uma fonte (como um banco de dados). Ele oferece uma interface simples para realizar operações CRUD (Create, Read, Update, Delete) sem expor os detalhes de implementação de como os dados são persistidos.

Usabilidade

O padrão DAO é amplamente utilizado em sistemas que necessitam acessar e manipular dados em um banco de dados ou outra fonte de dados persistente. Ele separa a lógica de acesso aos dados da lógica de negócios, facilitando a manutenção e possíveis trocas de banco de dados ou tecnologias de persistência no futuro.

Vantagens

Abstração: Esconde os detalhes de implementação do banco de dados.

Facilidade de manutenção: Mudanças no banco de dados não afetam a lógica de negócios.

Reutilização: O DAO pode ser reaproveitado por diferentes partes do sistema, desde que a interface de acesso seja mantida.

Desvantagens

Sobrecarga: Em sistemas simples, a criação de um DAO pode adicionar complexidade desnecessária.

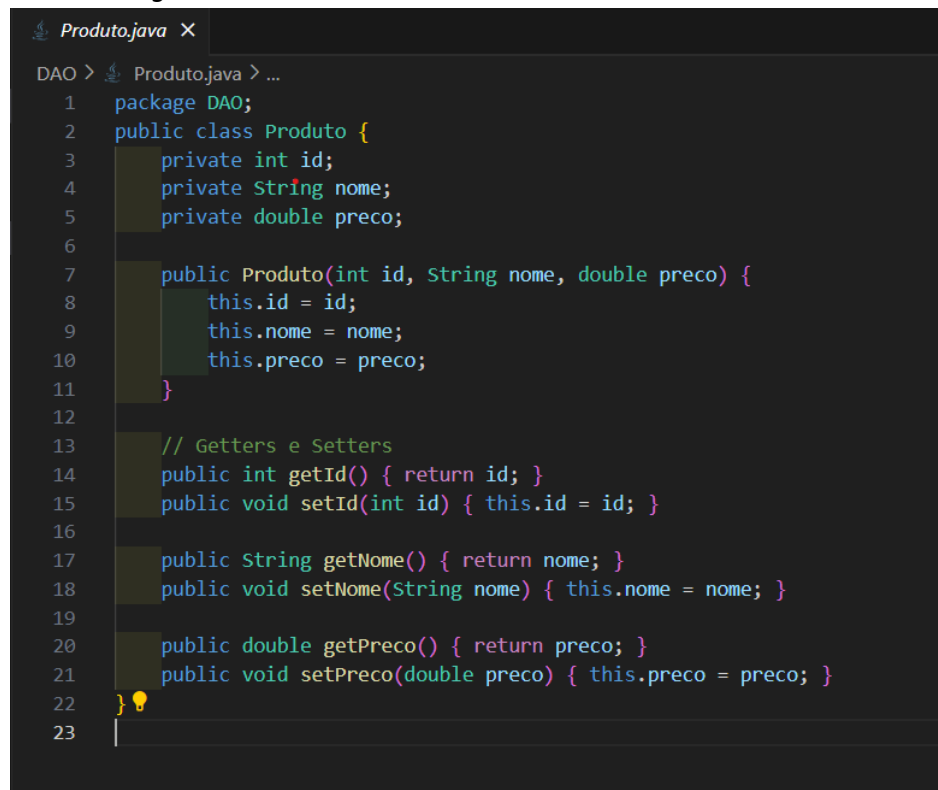
Possível repetição de código: Dependendo da implementação, pode haver código redundante em DAOs diferentes.

Exemplo de DAO (Gerenciamento de Produtos)

Neste exemplo, implementamos um sistema de gerenciamento de produtos utilizando o padrão DAO. A interface **ProdutoDAO** define as operações que podem ser realizadas, e a classe **ProdutoDAOImpl** implementa essas operações.

Código

Produto.java

A screenshot of an IDE window titled 'Produto.java'. The code is for a class named 'Produto' in the 'DAO' package. It has three private attributes: 'id' (int), 'nome' (String), and 'preco' (double). There is a constructor 'Produto(int id, String nome, double preco)' that initializes these attributes. There are also getter and setter methods for each attribute. The code is as follows:

```
DAO > Produto.java > ...
1  package DAO;
2  public class Produto {
3      private int id;
4      private String nome;
5      private double preco;
6
7      public Produto(int id, String nome, double preco) {
8          this.id = id;
9          this.nome = nome;
10         this.preco = preco;
11     }
12
13     // Getters e Setters
14     public int getId() { return id; }
15     public void setId(int id) { this.id = id; }
16
17     public String getNome() { return nome; }
18     public void setNome(String nome) { this.nome = nome; }
19
20     public double getPreco() { return preco; }
21     public void setPreco(double preco) { this.preco = preco; }
22 }
23
```

ProdutoDAO.java

```
ProdutoDAO.java X
DAO > ProdutoDAO.java > ...
1 package DAO;
2 import java.util.List;
3
4 public interface ProdutoDAO {
5     void adicionarProduto(Produto produto);
6     Produto buscarProduto(int id);
7     List<Produto> listarProdutos();
8     void atualizarProduto(Produto produto);
9     void deletarProduto(int id);
10 }
11
```

ProdutoDAOIMPL.java

```
ProdutoDAOImpl.java X
DAO > ProdutoDAOImpl.java > ProdutoDAOImpl > atualizarProduto(Produto)
1 package DAO;
2 import java.util.ArrayList;
3 import java.util.List;
4
5 public class ProdutoDAOImpl implements ProdutoDAO {
6     private List<Produto> produtos = new ArrayList<>();
7
8     @Override
9     public void adicionarProduto(Produto produto) {
10         produtos.add(produto);
11     }
12
13     @Override
14     public Produto buscarProduto(int id) {
15         for (Produto p : produtos) {
16             if (p.getId() == id) {
17                 return p;
18             }
19         }
20         return null;
21     }
22
23     @Override
24     public List<Produto> listarProdutos() {
25         return produtos;
26     }
27
28     @Override
29     public void atualizarProduto(Produto produto) {
30         for (Produto p : produtos) {
31             if (p.getId() == produto.getId()) {
32                 p.setNome(produto.getNome());
33                 p.setPreco(produto.getPreco());
34             }
35         }
36     }
37
38     @Override
39     public void deletarProduto(int id) {
40         produtos.removeIf(p -> p.getId() == id);
41     }
42 }
43
```

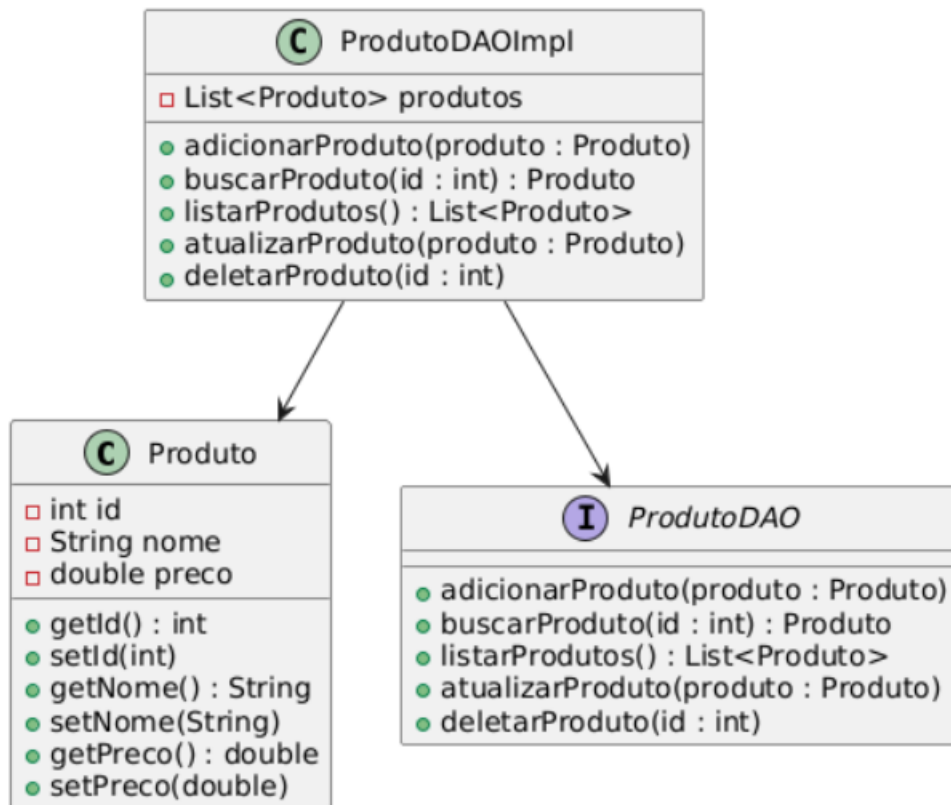
Main.java

```

Main.java X
DAO > Main.java > ...
1 package DAO;
2 public class Main {
3     public static void main(String[] args) {
4         ProdutoDAO produtoDAO = new ProdutoDAOImpl();
5
6         Produto produto1 = new Produto(id:1, nome:"Laptop", preco:2500.00);
7         Produto produto2 = new Produto(id:2, nome:"Smartphone", preco:1500.00);
8
9         // Adicionando produtos
10        produtoDAO.adicionarProduto(produto1);
11        produtoDAO.adicionarProduto(produto2);
12
13        // Listando produtos
14        System.out.println("Produtos cadastrados:");
15        for (Produto p : produtoDAO.listarProdutos()) {
16            System.out.println(p.getNome() + " - R$ " + p.getPreco());
17        }
18    }
19 }
20

```

UML do Exemplo DAO:



4. Conclusão

Os padrões MVC e DAO são ferramentas poderosas para estruturar e organizar aplicações. O **MVC** é especialmente útil em sistemas que possuem interação com o usuário, permitindo uma separação clara entre a lógica de negócios e a interface. Já o **DAO** é essencial em aplicações que requerem acesso a dados persistentes, pois abstrai o mecanismo de persistência, facilitando mudanças e manutenções no banco de dados.

A escolha de um padrão adequado depende das necessidades do sistema. No caso de sistemas com interface, o **MVC** é uma excelente opção, enquanto que o **DAO** é essencial para lidar com persistência de dados, promovendo uma arquitetura modular e escalável.

Referências

- Gamma, Erich, et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1994.
- Fowler, Martin. **Patterns of Enterprise Application Architecture**. Addison-Wesley, 2002.