
FIAP

Domain Driven Design
using Java

AGENDA

1. Padrões Arquiteturais em Java
2. Exercícios

Introdução

- Padrões abordados:
  - Model, View Controller (MVC)

Padrão MVC

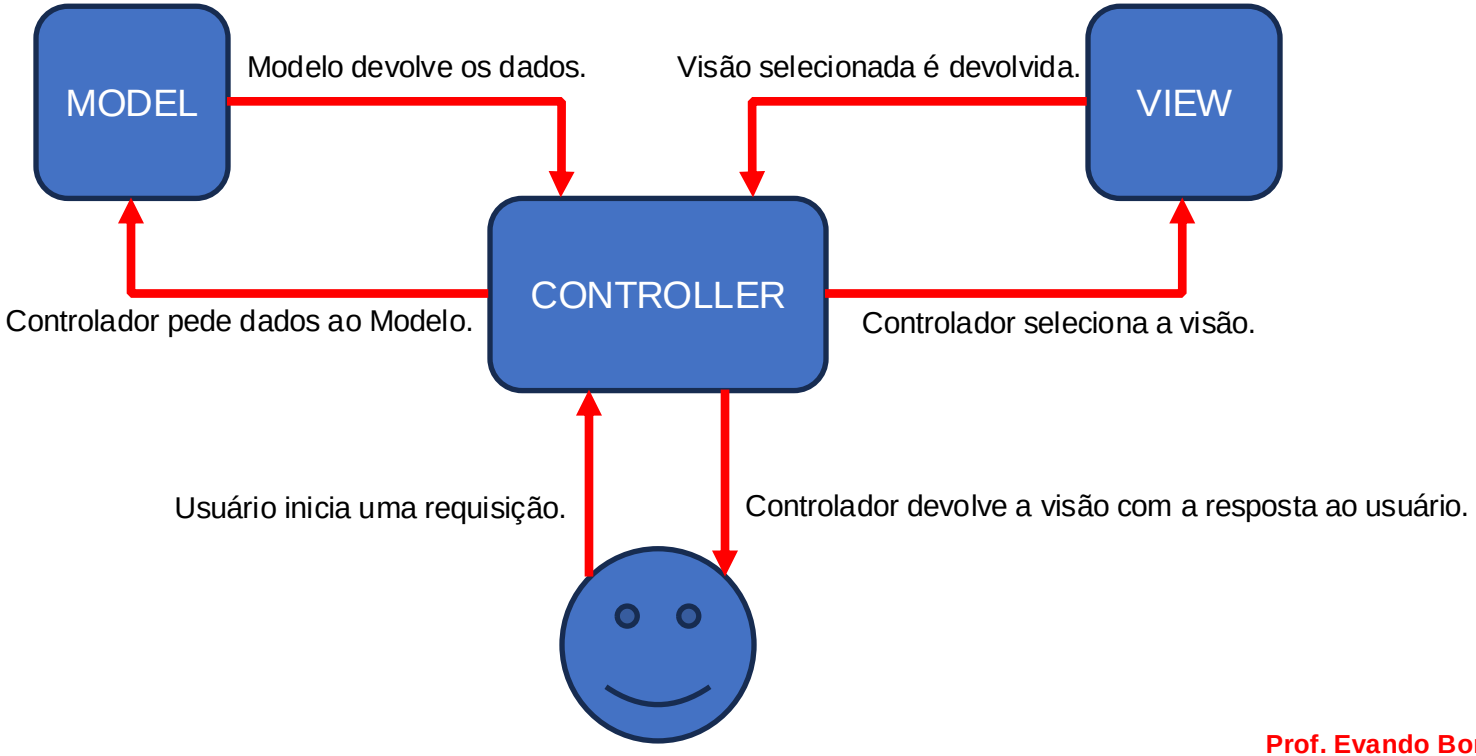
- O que é o Padrão MVC?
  - Uma arquitetura que separa responsabilidades em três componentes: **Model**, **View** e **Controller**.
  - Amplamente utilizado para facilitar manutenção, escalabilidade e reutilização do código.

- Porque aprender MVC:
  - Ensina boas práticas para construir aplicações robustas.
  - É aplicável tanto em sistemas desktop quanto em aplicações web.

Componentes do MVC

- Model (Modelo):
  - Gerencia os dados e a lógica de negócios da aplicação.
  - Interage com o banco de dados ou outras fontes externas (ex.: APIs).
  - Não sabe como os dados serão exibidos.
- View (Visão):
  - Responsável pela interface com o usuário.
  - Exibe os dados recebidos do Model e coleta as interações do usuário.
  - Exemplo: Páginas HTML (aplicações web) ou componentes Swing/JavaFX (aplicações desktop).
- Controller (Controlador):
  - Atua como intermediário entre o Model e a View.
  - Recebe comandos da View, processa a lógica necessária e atualiza o Model.
  - Retorna dados para a View, garantindo que ela exiba as informações de forma adequada.

Estrutura MVC



Benefícios do MVC

- Separa responsabilidades:
  - Facilita a identificação e correção de problemas.

- Código mais limpo:
  - Promove organização e modularidade.

- Escalabilidade:
  - Permite expandir componentes sem interferir nos outros.

Exemplo MVC

- Entidade

```java
public class Carro {
    private String marca;
    private String modelo;
    private int ano;

    public Carro(String marca, String modelo, int ano) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }

    // Getters e Setters
    public String getMarca() { return marca; }
    public void setMarca(String marca) { this.marca = marca; }

    public String getModelo() { return modelo; }
    public void setModelo(String modelo) { this.modelo = modelo; }

    public int getAno() { return ano; }
    public void setAno(int ano) { this.ano = ano; }
}
```

Exemplo MVC

- Classe DAO

```java
import java.util.ArrayList;
import java.util.List;

public class CarroDAO {
    private List<Carro> carros = new ArrayList<>();

    public void salvarCarro(Carro carro) {
        carros.add(carro);
    }

    public List<Carro> listarCarros() {
        return carros;
    }
}
```

Exemplo MVC

- View – Interface gráfica com swing

```java
import javax.swing.*;
import java.awt.event.ActionListener;

public class CarroView extends JFrame {
    private JTextField marcaField = new JTextField(10);
    private JTextField modeloField = new JTextField(10);
    private JTextField anoField = new JTextField(4);
    private JButton salvarButton = new JButton("Salvar Carro");
    private JTextArea displayArea = new JTextArea(10, 30);

    public CarroView() {
        JPanel carroPanel = new JPanel();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400, 300);

        carroPanel.add(new JLabel("Marca: "));
        carroPanel.add(marcaField);
        carroPanel.add(new JLabel("Modelo: "));
        carroPanel.add(modeloField);
        carroPanel.add(new JLabel("Ano: "));
        carroPanel.add(anoField);
        carroPanel.add(salvarButton);
        carroPanel.add(new JScrollPane(displayArea));

        this.add(carroPanel);
    }
```

```java
    // Métodos para interagir com o Controller
    public String getMarca() { return marcaField.getText(); }
    public String getModelo() { return modeloField.getText(); }
    public int getAno() { return Integer.parseInt(anoField.getText()); }
    public void addSalvarCarroListener(ActionListener listener) {
        salvarButton.addActionListener(listener);
    }

    public void displayCarros(String carros) {
        displayArea.setText(carros);
    }

    public void displayErrorMessage(String errorMessage) {
        JOptionPane.showMessageDialog(this, errorMessage);
    }
}
```

Exemplo MVC - Controller

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CarroController {
    private CarroView carroView;
    private CarroDAO carroDAO;

    public CarroController(CarroView view, CarroDAO dao) {
        this.carroView = view;
        this.carroDAO = dao;

        this.carroView.addSalvarCarroListener(new SalvarCarroListener());
    }
```

```java
    class SalvarCarroListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                String marca = carroView.getMarca();
                String modelo = carroView.getModelo();
                int ano = carroView.getAno();

                Carro novoCarro = new Carro(marca, modelo, ano);
                carroDAO.salvarCarro(novoCarro);

                StringBuilder carrosText = new StringBuilder();
                for (Carro carro : carroDAO.listarCarros()) {
                    carrosText.append(carro.getMarca())
                              .append(" ")
                              .append(carro.getModelo())
                              .append(" - ")
                              .append(carro.getAno())
                              .append("\n");
                }

                carroView.displayCarros(carrosText.toString());
            } catch (Exception ex) {
                carroView.displayErrorMessage("Erro ao salvar carro!");
            }
        }
    }
}
```

Exercícios

1 - Criação de Cadastro:

- Crie uma aplicação MVC para cadastrar **produtos.**
- **Model:** Classe Produto com atributos nome, preco, e quantidade.
- **View:** Interface gráfica para entrada de dados do produto.
- **Controller:** Processa os dados e os exibe na interface.

2 - Adaptação do **DAO**:

- Modifique o DAO para permitir **exclusão** de produtos.
- Adicione um botão na View para excluir itens com base no nome.

3 - Melhorando a **View:**

- Adicione validação nos campos de entrada.
- Exiba uma mensagem de erro se o campo nome estiver vazio ou preço for negativo.


FIAP