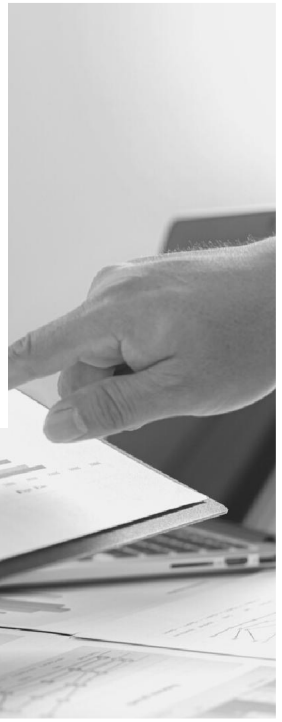
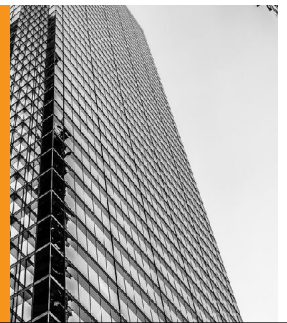
Desenvolver e organizar interface gráfica para aplicações desktop

Arquitetura de Software

Rede Fecomércio RS de Educação





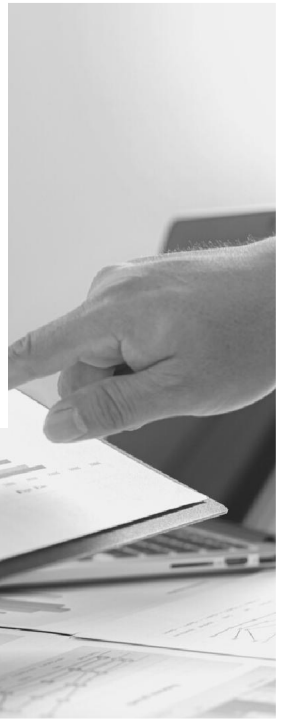



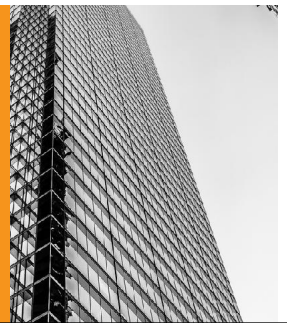

Arquitetura de Software












Rede Fecomércio RS de Educação

O que é Arquitetura de Software?

É a forma como o sistema é organizado:

- Como os arquivos se relacionam
- Onde cada responsabilidade fica
- Como as partes se comunicam

Arquitetura = estrutura interna do código

O Problema de NÃO ter Arquitetura

Quando não existe arquitetura:

- Código misturado (tela + regra + menu)
- Arquivos gigantes
- Dificuldade de entender
- Erros ao alterar algo simples

“Funciona hoje, quebra amanhã”

Exemplo de Código Sem Arquitetura

```
// Tudo no JFrame
menu.add(item);
item.addActionListener(e -> {
    JOptionPane.showMessageDialog(null,
"Mensagem");
});
```

- Tela cria menu
- Tela controla lógica
- Difícil de manter

O que a Arquitetura Resolve?

- Código organizado
- Cada classe tem uma função
- Facilita manutenção
- Facilita crescimento do sistema
- Pensado para o futuro
- Pensado para trabalho em equipe

Arquitetura de Software aplicada ao Java Swing









Rede Fecomércio RS de Educação

Como organizar um sistema desktop Java corretamente

Java Swing não é só interface gráfica

- A tela não roda em loop
- O sistema espera eventos
- Cada ação dispara um método

Arquitetura ajuda a organizar esses eventos

Problema Comum em Projetos Swing

Em projetos sem arquitetura, acontece:

- Código do menu dentro da tela
- Regras misturadas com interface
- Muitos ActionListener no mesmo arquivo
- Classe com centenas de linhas
- Difícil de entender
- Difícil de manter

Conceito-chave: Separação de Responsabilidades

Princípio da Responsabilidade Única

Cada parte do sistema deve ter uma função clara:

Tela → exibir interface

Menu → navegação

Mensagens → comunicação com usuário

“Quem faz tudo, faz tudo mal”

Conceito-chave: Separação de Responsabilidades

Arquitetura MVC - MODEL - VIEW - CONTROLLER

MVC é um padrão de arquitetura que organiza o sistema em três responsabilidades principais, evitando código bagunçado e difícil de manter.

MVC não é obrigatório, apenas uma boa prática profissional.

Arquitetura MVC
MODEL (Modelo)

Responsável pelos dados e regras de negócio.

- Representa as informações do sistema
- Contém validações
- Não conhece a interface gráfica

Exemplos:

- Cliente
- Produto
- Funcionário
- Venda

```
public class Cliente {
    private String nome;
    private String cpf;
}
```

Arquitetura MVC
VIEW (Visão / Interface)

Responsável por mostrar a interface ao usuário.

- Usa JFrame, JDialog, JPanel
- Mostra dados
- Captura ações do usuário
- Não contém regra de negócio

```
public class TelaCliente extends JFrame {
    // componentes visuais
}
```

Arquitetura MVC
Controller (Controle)

Responsável por ligar a View ao Model.

- Recebe ações da View
- Processa regras
- Atualiza o Model
- Decide o que a View deve mostrar

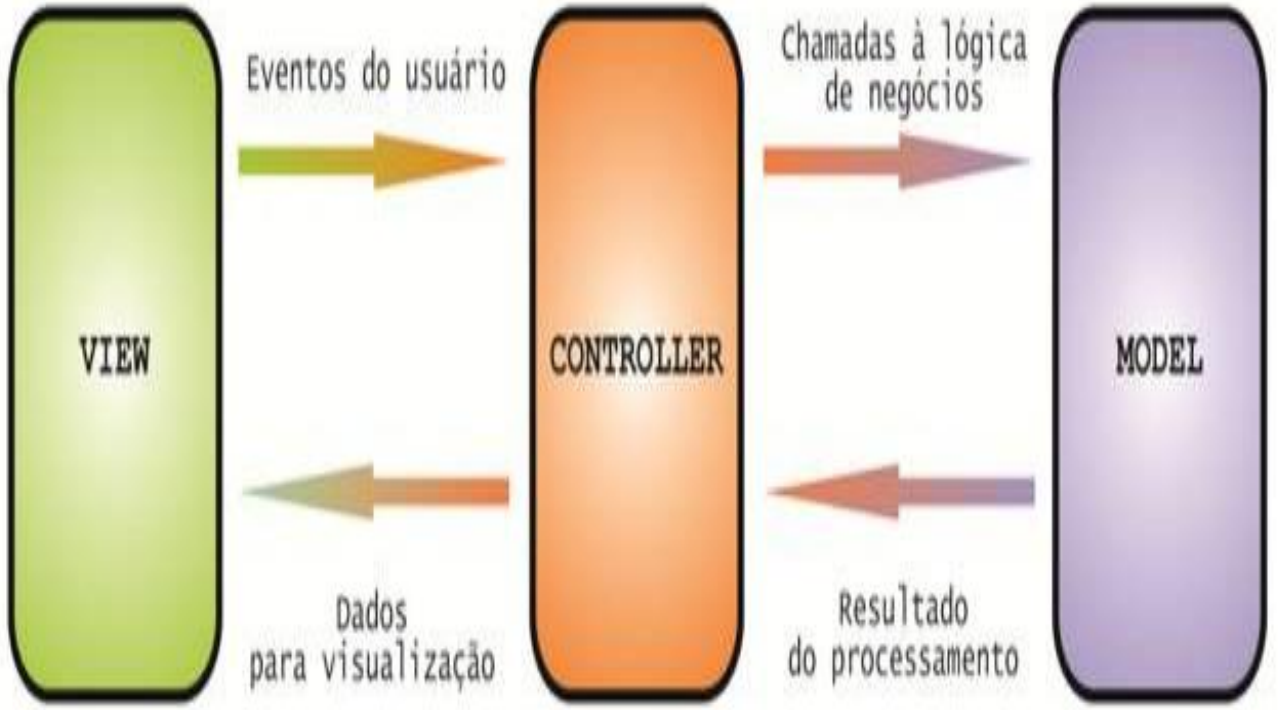
```
public class ClienteController {
    public void salvar(Cliente c) {
        // validação e lógica
    }
}
```

Estrutura Arquitetural (MVC)

Fluxo básico:

Essa estrutura de pacotes é organizada e segue uma lógica funcional para projetos Swing.

app/ → inicialização
view/ → telas (JFrame / JDialog)
menu/ → JMenuBar e JMenuItem
util/ → mensagens e funções comuns



Arquitetura Ideal no Swing

app/ (Inicialização)

Este pacote deve conter apenas a classe principal que configura o visual e inicia o sistema.

Normalmente usa-se o FlatLaf (Look and Feel) para dar uma aparência moderna e profissional (estilo VS Code ou IntelliJ) ao seu Java Swing.

Exemplo: Main.java (contém o public static void main).

Arquitetura Ideal no Swing

view/ (Telas)

Aqui ficam as janelas do sistema.

JFrame: Para as janelas principais (Ex: LoginView.java, MainView.java).

JDialog: Para janelas pop-up ou de cadastro que exigem foco (Ex: UsuarioDialog.java).

Organização: Se o projeto crescer, crie subpastas por módulo (ex: view.vendas, view.relatorios).

Arquitetura Ideal no Swing

menu/ (Componentes de Navegação)

Em vez de colocar todo o código do menu dentro da MainView, você isola a lógica aqui.

Estratégia: Crie uma classe que estende JMenuBar. Isso permite que você altere o menu sem poluir o código da tela principal.

Exemplo: MenuPrincipal.java.

Arquitetura Ideal no Swing

util/ (Utilitários e Helpers)

Funções que você repete em vários lugares.

Mensagens: Uma classe Mensagem.java com métodos estáticos como Mensagem.erro("Texto") ou Mensagem.alerta("Texto") usando JOptionPane.

Validadores: Métodos para validar CPF, e-mail ou formatar datas.

Conexão: Se não usar um pacote dao, a classe de conexão com banco de dados costuma ficar aqui.