TÓPICO 02: INTRODUÇÃO A DESIGN PATTERNS

Design Patterns - Professor Ramon Venson - SATC 2026.1

Roteiro

- O que são Padrões de Design
- Origem
- Importância
- Críticas / cuidados
- Categorias (GoF)
- Outros tipos de padrões

O que são Design Patterns?

- Soluções **reutilizáveis** para problemas **recorrentes**
- Baseados em **boas práticas**
- Descrevem **estrutura** e **intenção**, não código pronto
- Ajudam a deixar o código:
  - mais flexível
  - mais modular
  - mais fácil de manter

Padrão ≠ receita pronta

- Em geral **não é** copiar e colar código
- É um **modelo** / **guia** para resolver um tipo de problema
- Precisa ser **adaptado** ao contexto

Exemplo de ideia de padrão

```
// Estratégia diferente para cálculo de frete
public interface CalculoFrete {
  double calcular(double peso);
}

public class FreteSedex implements CalculoFrete {
  public double calcular(double peso) { return peso * 10; }
}

public class FretePac implements CalculoFrete {
  public double calcular(double peso) { return peso * 5; }
}
```

- A ideia (Strategy) pode ser aplicada em vários contextos, não só frete

Origem (1/2)

- Conceito de “padrão” veio da **Arquitetura**
- Christopher Alexander – 1977 – *A Pattern Language*
  - Padrões para projetar cidades, prédios, ambientes

Origem (2/2)

- Adaptado para software nos anos 90
- “Gang of Four” (GoF):
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Livro (1994):
  Design Patterns: Elements of Reusable Object-Oriented Software
- Catálogo clássico com **23 padrões**

Importância

Padrões ajudam a:

- Reutilizar soluções testadas
- Melhorar a **comunicação** entre desenvolvedores
- Organizar melhor o código
- Facilitar manutenção e evolução
- Reduzir riscos de “reinventar a roda”

Comunicação com padrões

- “Vamos usar um **Strategy** aqui”
- “Isso parece um **Observer**”

=> Uma frase já transmite:

- o **problema**
- a **ideia de solução**
- as **consequências** esperadas

Críticas (1/4) – Sintoma de problema

- Às vezes o padrão “nasce” de uma **limitação da linguagem**
- Linguagens modernas tornam alguns padrões:
  - triviais
  - desnecessários

> Ex.: Strategy pode virar apenas “passar função como parâmetro” em linguagens funcionais.

Críticas (2/4) – Overengineering

- Uso exagerado de padrões
- Problemas:
  - muito código para pouco ganho
  - excesso de classes / interfaces
  - mais difícil de entender

Regra prática:

> Comece simples. Introduza um padrão quando houver um **motivo claro**.

Críticas (3/4) – Não são bala de prata

Padrões **não**:

- consertam modelo de domínio ruim
- resolvem decisões arquiteturais equivocadas
- substituem entendimento do problema

São **ferramentas**, não objetivo final.

Críticas (4/4) – Jargão

- Muito nome: Factory, Visitor, Decorator, Proxy...
- Pode:
  - excluir quem é iniciante
  - gerar mal-entendidos se o time não conhece os termos

Dica:

- Use o nome do padrão
- Mas **explique a intenção**: “resolver X fazendo Y”.

Categorias GoF

- **Criacionais**
- **Estruturais**
- **Comportamentais**

Cada categoria agrupa padrões com **propósitos parecidos**.

Padrões Criacionais

- Problema: **como criar objetos**?
- Evitar:
  - `new` espalhado
  - acoplamento à implementação

Exemplos de padrões criacionais

- Singleton
- Factory Method
- Abstract Factory
- Builder
- Prototype

Exemplo simples: Factory Method

```
public interface Transporte {
  void entregar();
}

public class Caminhao implements Transporte {
  public void entregar() { System.out.println("Entrega por caminhão"); }
}

public abstract class Logistica {
  public void planejarEntrega() {
    Transporte t = criarTransporte();
    t.entregar();
  }

  protected abstract Transporte criarTransporte();
}
```

Padrões Estruturais

- Foco: **como organizar classes/objetos**
- Ajudam a montar estruturas maiores de forma flexível

Exemplos de padrões estruturais

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Exemplo simples – Adapter

```
// API antiga
class TomadaAntiga {
  void ligarDoisPinos() { ... }
}

// Nova interface esperada
interface TomadaTresPinos {
  void ligarTresPinos();
}

// Adapter
class Adaptador implements TomadaTresPinos {
  private TomadaAntiga antiga;
  public Adaptador(TomadaAntiga antiga) { this.antiga = antiga; }
  public void ligarTresPinos() { antiga.ligarDoisPinos(); }
}
```

Padrões Comportamentais

- Foco: **comunicação** entre objetos
- Como responsabilidades são distribuídas

Exemplos de padrões comportamentais

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Observer
- Strategy
- Template Method
- Visitor

Exemplo simples – Observer

```
interface Observador {
  void atualizar();
}

class Sujeito {
  private List<Observador> obs = new ArrayList<>();
  void adicionar(Observador o) { obs.add(o); }
  void notificar() {
    for (Observador o : obs) o.atualizar();
  }
}
```

- Vários objetos reagem a um evento sem o sujeito saber detalhes deles

Outros tipos de padrões

Além dos GoF:

- **Arquiteturais**
  - MVC, MVP, MVVM
  - Microservices / Event-Driven Architecture
- **Integração**
  - Enterprise Integration Patterns
- **Teste**
  - Test Automation Patterns

Resumo

- Padrões de design = **vocabulário + soluções**
- Use para:
  - comunicar melhor
  - organizar melhor o código
- Evite:
  - aplicar padrão “porque é bonito”
  - complexidade desnecessária

> Entenda o problema primeiro.
> Depois veja se um padrão ajuda.