



#R4E

Software Developer

# Design Patterns

Conceitos



# Conteúdo

- Conceito de Design Pattern
- História dos Design Patterns
- Pertinência de Uso
- Críticas aos Design Patterns
- Classificação de Patterns
- Cuidados e Boas Práticas

# O que é um Design Pattern

- **Design Patterns** (Padrões de Projeto) são soluções típicas para problemas comuns em projetos de software. São como projetos de obras pré-fabricadas que podemos adaptar para resolver um problema de projeto recorrente no código.



# O que é um Design Pattern

- Não é possível simplesmente encontrar um **Design Pattern** e copiá-lo para o programa, como se faz com funções e bibliotecas. O **Design Pattern** não é um pedaço de código específico, mas um conceito geral para resolver um problema em particular. Podemos seguir os detalhes do padrão e implementar uma solução que se adeque às realidades do projeto em questão.



# O que é um Design Pattern

- Os padrões são frequentemente confundidos com algoritmos, porque ambos os conceitos descrevem soluções típicas para alguns problemas conhecidos. Enquanto um **algoritmo** define **sempre** um **conjunto claro de ações** para atingir uma meta, um **padrão** é mais uma **descrição de alto nível** de uma solução. O código do mesmo padrão aplicado para dois programas distintos pode ser bem diferente.



# O que é um Design Pattern

- A analogia de um algoritmo é uma receita de comida: ambos têm etapas claras para chegar a um objetivo. Por outro lado, um padrão é mais como uma planta de obras: podemos ver o resultado e as suas funcionalidades, mas a ordem exata de implementação depende de nós.



# Em que consiste um Design Pattern

- A maioria dos padrões são descritos formalmente para que as pessoas possam reproduzi-los em diferentes contextos. Algumas seções que são geralmente presentes na descrição de um padrão:
  - O **Propósito** do padrão descreve brevemente o problema e a solução.
  - A **Motivação** explica a fundo o problema e a solução que o padrão torna possível.
  - As **Estruturas** de classes mostram cada parte do padrão e como se relacionam.
  - **Exemplos de código** numa das linguagens de programação populares tornam mais fácil compreender a ideia por trás do padrão.
- Alguns catálogos de padrão listam outros detalhes úteis, tais como a **aplicabilidade** do padrão, **etapas de implementação**, e **relações com outros padrões**.

# História de Design Patterns

- Quem inventou os **Design Patterns**? Boa pergunta, mas não muito precisa. Os design patterns não são conceitos obscuros e sofisticados — bem pelo contrário. Os padrões são **soluções típicas** para **problemas comuns** em projetos orientados a objetos.
- Quando uma solução é repetida uma vez e outra e outra... em vários projetos, alguém vai, eventualmente, nomeá-la e descrever a solução em detalhe. É assim que um padrão é descoberto.





# História de Design Patterns

- O conceito de padrões foi primeiramente descrito por Christopher Alexander no livro Uma Linguagem de Padrões. O livro descreve uma “linguagem” para o projeto de um ambiente urbano. As unidades dessa linguagem são os padrões. Eles podem descrever quão alto as janelas devem estar, quantos andares um prédio deve ter, quão largas as áreas verdes de um bairro devem ser, e assim em diante.



# História de Design Patterns

- A ideia foi seguida por quatro autores: Erich Gamma, John Vlissides, Ralph Johnson, e Richard Helm. Em 1994, eles publicaram Padrões de Projeto — Soluções Reutilizáveis de Software Orientado a Objetos, no qual eles aplicaram o conceito de padrões de projeto para programação. O livro mostrava 23 padrões que resolviam vários problemas de projetos orientado a objetos e tornou-se rapidamente um best-seller. Devido ao longo título, as pessoas começaram a chamá-lo simplesmente de “o livro da Gangue dos Quatro (Gang of Four)” que foi simplificado para o “livro GoF”.



# Porquê aprender Design Patterns?

- A verdade é que conseguimos trabalhar como um programador, por muitos anos, sem saber sobre um único padrão. Muitas pessoas fazem exatamente isso. Ainda assim, estaremos a implementar alguns padrões mesmo sem saber. Então, porque gastar tempo a aprender sobre eles?



# Porquê aprender Design Patterns?

- Os **Design Patterns** são um kit de ferramentas para soluções tentadas e testadas para problemas comuns em projetos de software. Mesmo que nunca tenhamos encontrado esses problemas, saber sobre os padrões é muito útil porque eles ensinam como resolver vários problemas usando princípios de projetod orientado a objetos.



# Porquê aprender Design Patterns?

- Os Design Patterns definem uma linguagem comum que a equipa pode usar para comunicar de forma mais eficiente. Podemos dizer, “Oh, é só usar um **Singleton** para isso,” e toda a gente vai entender a ideia por trás da sugestão. Não é preciso explicar o que é um **singleton** se toda a gente reconhecer o padrão.



# Críticas aos Design Patterns

"Desenrascanços" para uma linguagem de programação fraca

- Geralmente a necessidade de padrões surge quando as pessoas escolhem uma linguagem de programação ou uma tecnologia que tem uma deficiência no nível de abstração. Neste caso, os padrões se transformam em desenrascanços que dão à linguagem os superpoderes necessários.
- Por exemplo, o padrão Strategy pode ser implementado com uma simples função anônima (lambda) na maioria das linguagens de programação modernas.



# Críticas aos Design Patterns

## Soluções ineficientes

- Os padrões tentam sistematizar abordagens que já são amplamente usadas. Essa unificação é vista por muitos como um dogma (elefante na sala) e eles implementam os padrões “direto ao ponto”, sem adaptá-los ao contexto dos seus projetos.



# Críticas aos Design Patterns

## Uso Injustificável


*Se tudo que tenho é um martelo, tudo parece um prego.*

- Esse é o problema que assombra muitos novatos que acabaram de se familiarizar com os padrões. Tendo aprendido sobre eles, tentam aplicá-los em tudo, até mesmo em situações onde um código mais simples seria suficiente.





# Classificação de Design Patterns

- 
- Design Patterns diferem em complexidade, nível de detalhe, e escala de aplicabilidade ao sistema inteiro que está a ser desenvolvido.
  - Analogia com a construção de uma estrada: podemos fazer uma intersecção mais segura instalando sinais de trânsito ou construindo nós de vários níveis com passagens subterrâneas para pedestres.

# Classificação de Design Patterns

- Os padrões mais básicos e de baixo nível são frequentemente chamados idiomáticos. Eles aplicam-se, geralmente, apenas a uma única linguagem de programação.
- Os padrões mais universais e de alto nível são os padrões arquitetônicos. Os programadores podem implementar esses padrões em praticamente qualquer linguagem. Ao contrário de outros padrões, podem ser usados para fazer o projeto da arquitetura de uma aplicação na sua integridade.



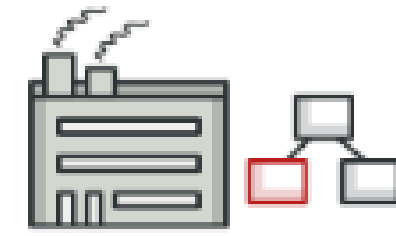
# Classificação de Design Patterns

- Além disso, todos os padrões podem ser categorizados pelo seu propósito ou intenção. Temos três grupos principais de padrões:
  - Os **padrões criacionais (Creational Patterns)** fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização de código.
  - Os **padrões estruturais (Structural Patterns)** explicam como instanciar objetos e classes em estruturas maiores, enquanto ainda mantêm as estruturas flexíveis e eficientes.
  - Os **padrões comportamentais (Behavioral Patterns)** cuidam da comunicação eficiente e da sinalização de responsabilidades entre objetos.

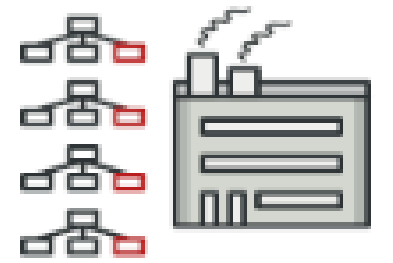
# Creational Patterns

- Estes padrões fornecem vários mecanismos de criação de objetos, que aumentam a flexibilidade e reutilização de código já existente.

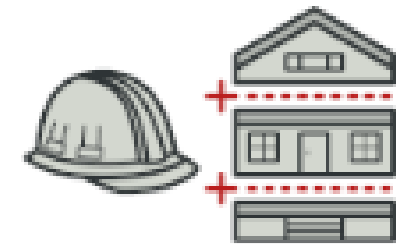
- Factory Method
- Abstract Method
- Builder
- Prototype
- Singleton



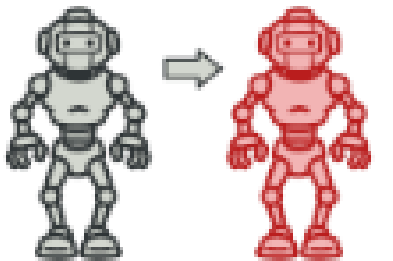
**Factory Method**



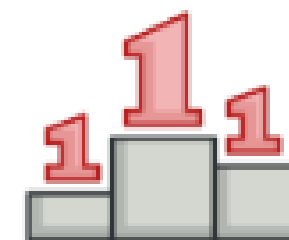
**Abstract Factory**



**Builder**



**Prototype**

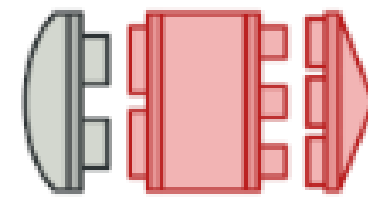


**Singleton**

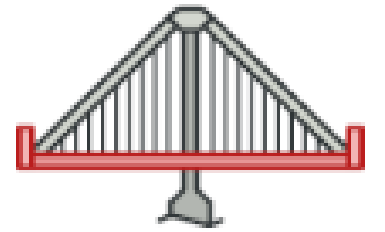
# Structural Patterns

- Estes padrões explicam como montar objetos e classes em estruturas maiores mas mantendo essas estruturas flexíveis e eficientes.

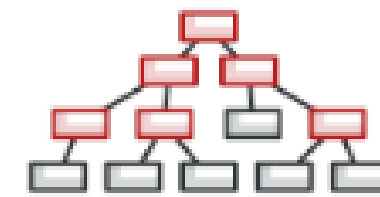
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy



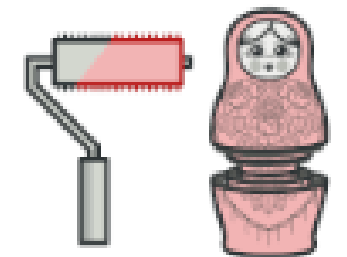
**Adapter**



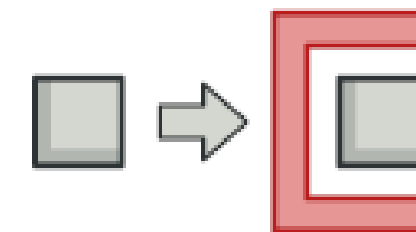
**Bridge**



**Composite**



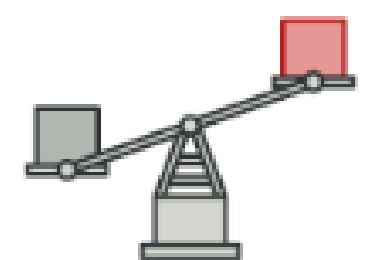
**Decorator**



**Proxy**



**Facade**

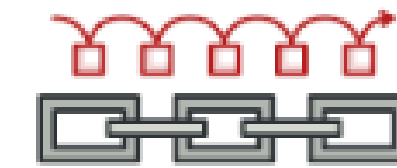


**Flyweight**

# Behavioral Patterns

Estes padrões são focados em algoritmos e a designação de responsabilidades entre objetos.

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor



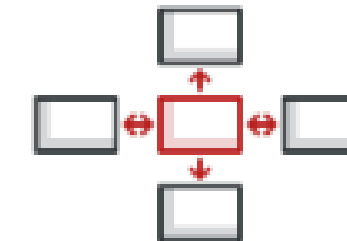
**Chain of  
Responsibility**



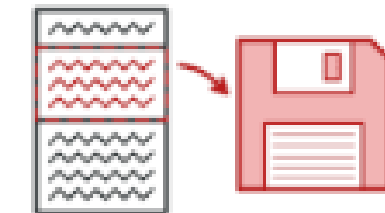
**Command**



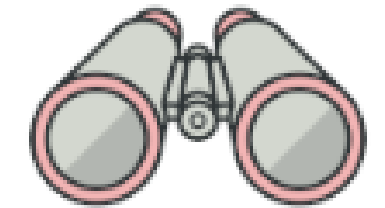
**Iterator**



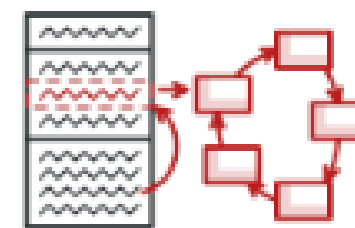
**Mediator**



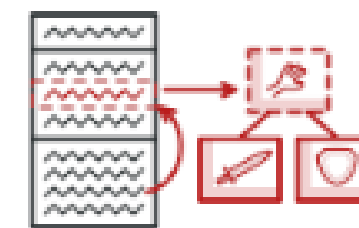
**Memento**



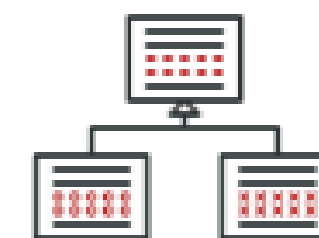
**Observer**



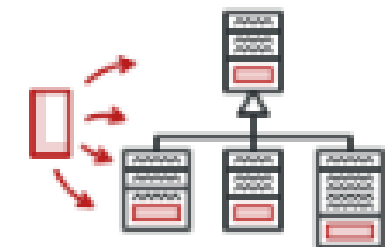
**State**



**Strategy**



**Template  
Method**



**Visitor**



#R4E

Software Developer

# Design Patterns

Conceitos

