

Design Patterns

UNIVERSIDADE DO VALE DO SAPUCAÍ – UNIVAS – POUSO ALEGRE/MG
PROFESSOR: RAFFAEL CARVALHO

Introdução ao Design Patterns

O que é um padrão de projeto?

- Padrões de projeto são soluções típicas para problemas comuns em projeto de software.
- Eles são como plantas de obra pré fabricadas que você pode customizar para resolver um problema de projeto recorrente em seu código.

O que é um padrão de projeto?

- Você não pode apenas encontrar um padrão e copiá-lo para dentro do seu programa, como você faz com funções e bibliotecas que encontra por aí.
- O padrão não é um pedaço de código específico, mas um conceito geral para resolver um problema em particular.
- Você pode seguir os detalhes do padrão e implementar uma solução que se adeque às realidades do seu próprio programa.

O que é um padrão de projeto?

- Os padrões são frequentemente confundidos com algoritmos, porque ambos os conceitos descrevem soluções típicas para alguns problemas conhecidos.
- Enquanto um algoritmo sempre define 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.

Do que consiste um padrão?

Uma descrição de um padrão geralmente apresenta:

- 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** em uma das linguagens de programação populares tornam mais fácil compreender a ideia por trás do padrão.

Classificação dos padrões

- Os **padrões de criação** fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização de código.
- Os **padrões estruturais** explicam como montar objetos e classes em estruturas maiores, enquanto ainda mantém as estruturas flexíveis e eficientes.
- Os **padrões comportamentais** cuidam da comunicação eficiente e da assinalação de responsabilidades entre objetos.

História dos padrões

- O conceito de padrões foi primeiramente descrito por Christopher Alexander em ***Uma Linguagem de Padrões***.
- *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**. O livro mostrava 23 padrões que resolviam vários problemas de projeto orientado a objetos e se tornou um best-seller rapidamente.*

Por que devo aprender padrões?

- *Os padrões de projeto são um kit de ferramentas para soluções tentadas e testadas para problemas comuns em projeto de software.*
- *Os padrões de projeto definem uma linguagem comum que você e seus colegas podem usar para se comunicar mais eficientemente. Você pode dizer, “Oh, é só usar um Singleton para isso” e todo mundo vai entender a ideia por trás da sua sugestão. Não é preciso explicar o que um singleton é se você conhece o padrão e seu nome.*

Princípios de Projetos de Software

Características de um bom projeto

Reutilização de código

- **Objetivo:** economizar tempo e custo no desenvolvimento ao aproveitar código existente.
- **Desafios:** código antigo pode ser difícil de adaptar por:
 - Acoplamento forte entre componentes.
 - Dependência de classes concretas.
 - Uso de operações codificadas diretamente.

Características de um bom projeto

- **Solução:** *usar padrões de projeto para aumentar a flexibilidade.*
- **Níveis de reutilização:**
 - **Classes** (baixo nível): *como bibliotecas ou containers.*
 - **Padrões de projeto** (nível intermediário): *descrevem relações entre classes.*
 - **Frameworks** (alto nível): *agrupam decisões de projeto e oferecem extensibilidade via herança e callbacks.*

Princípios universais de projeto

Encapsule o que varia

- ***Ideia central:** separar o que muda daquilo que permanece constante.*
- ***Analogia:** compartimentos de navio - se um falhar, o restante se mantém flutuando.*

Encapsule o que varia

Aplicação prática:

- **Nível de método:**
 - *Evitar misturar lógica de cálculo de impostos com o cálculo total do pedido.*
 - *Extrair a lógica de impostos para um método ou classe específica.*
- **Nível de classe:**
 - *Evitar adicionar muitas responsabilidades a uma única classe.*
 - *Criar classes especializadas para funcionalidades específicas (ex: classe separada para cálculo de imposto).*

Programa para uma interface, não uma implementação

Conceito: *dependa de abstrações (interfaces ou classes abstratas), não de implementações concretas.*

Benefícios:

- *Maior flexibilidade.*
- *Menor acoplamento.*
- *Facilidade para substituir ou estender funcionalidades.*

Programa para uma interface, não uma implementação

Passos recomendados:

- 1. Identificar o que uma classe precisa de outra.*
- 2. Criar uma interface com esses métodos.*
- 3. Fazer a classe concreta implementar essa interface.*
- 4. A classe dependente deve usar a interface, não a classe diretamente.*

Programa para uma interface, não uma implementação

Exemplo prático:

- *Criar uma interface Empregado comum a Desenvolvedor, Tester, etc.*
- *A classe Empresa trabalha com a interface, podendo usar qualquer tipo de empregado.*

Prefira composição sobre herança

Herança reutiliza código, mas pode trazer problemas como:

- *Subclasses obrigadas a implementar métodos desnecessários.*
- *Quebra de encapsulamento.*
- *Acoplamento forte.*
- *Hierarquias complexas e difíceis de manter.*

Prefira composição sobre herança

Composição:

- *representa relação "tem um" (ex: um carro tem um motor).*
- *Permite que objetos deleguem comportamentos a outros.*
- *Facilita mudanças em tempo de execução (ex: trocar o motor de um carro).*
- *Também se aplica à agregação (ex: um carro tem um motorista, mas não controla seu ciclo de vida).*

Exemplo prático:

- *Ao invés de criar classes CaminhaoEletricoComAutopiloto, etc., crie componentes como Motor, Navegacao e combine-os com composição.*

Exercício

1. *Com base no material estudado, escreva uma resenha que explore os conceitos apresentados e reflita sobre sua importância no desenvolvimento de sistemas de qualidade.*

Bibliografia Básica

- PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional. 8. ed. São Paulo: McCraw Hill – Artmedia, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML
- Guia do Usuário. Rio de Janeiro: Campus, 2012.
- JOHNSON, Ralph; VLISSIDES, John; HELM, Richard; GAMMA, Erich. Padrões de Projeto. São Paulo: Bookman, 2008.