Padrões de Projeto

Design Patterns

Por Luiz Otávio Miranda

Fonte: Livro Padrões de Projeto - Soluções reutilizáveis de software orientado a objetos - GoF

O que são padrões de projeto

- São **soluções para problemas conhecidos** na arquitetura de software que foram utilizados e testados no passado e continuam relevantes nos dias atuais
- Foram catalogados e popularizados pelo livro "***Padrões de projeto - Soluções reutilizáveis de software orientado a objetos***" (*os padrões da "GoF", de 1994 - por Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides*)
- Inicialmente, foram criados com foco na **POO**. Atualmente, são **soluções universais** que funcionam em qualquer linguagem de programação, mesmo as que não suportam **POO** em sua totalidade (como *JavaScript*, *Python*, entre outras)
- São divididos em 3 categorias: **de criação**, que visam abstrair o processo de como objetos são criados na aplicação; **estruturais**, que lidam com a composição de classes e objetos; **comportamentais**, que caracterizam como as classes e objetos interagem e distribuem responsabilidades na aplicação
- Podem ser combinados para criar uma aplicação completa

Benefícios e problemas

O que é bom:

- Você não precisa reinventar a roda
- Padrões universais facilitam o entendimento do seu projeto
- Evita refatoração desnecessária
- Ajuda na reutilização de código (conceito **DRY** - Don't repeat yourself)
- Abstrai e nomeia partes particulares do projeto
- Ajuda na aplicação dos princípios do design orientado a objetos (**SOLID**)
- Facilitam a criação de testes unitários

O que é ruim:

- Alguns padrões podem ser complexos até que você os compreenda
- Muito código para atingir um objetivo simples
- Podem trazer otimizações prematuras para o seu código
- Se usados incorretamente, podem atrapalhar ao invés de ajudar

Padrões de projeto que vamos estudar

De criação

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

Estrutural

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

Comportamental

- Interpreter
- Template method
- Chain of responsibility
- Iterator
- Command
- Mediator
- Memento
- Observer
- State
- Strategy
- Visitor

Princípios do design orientado a objetos (SOLID)

- **Single Responsibility Principle** (Princípio da responsabilidade única) - *uma classe deve ter apenas um motivo para mudar* (evite conjunções aditivas: e, bem como, também...). *Este princípio está intimamente ligado com outro, conhecido como* ***Separation of concerns***
- **Open/closed principle** (Princípio do aberto/fechado) - *classes ou objetos e métodos devem estar abertos para extensão, mas fechados para modificações*
- **Liskov substitution principle** (Princípio da substituição de Liskov) - *classes derivadas devem ser capazes de substituir totalmente classes-bases*
- **Interface segregation principle** (Princípio da segregação de Interface) - *os clientes não devem ser forçados a depender de interfaces que não utilizam*
- **Dependency inversion principle** (Princípio da inversão de dependência) - *módulos de alto nível não devem ser dependentes do módulos de baixo nível; ambos devem depender de abstrações. Detalhes devem depender das abstrações, não o inverso*