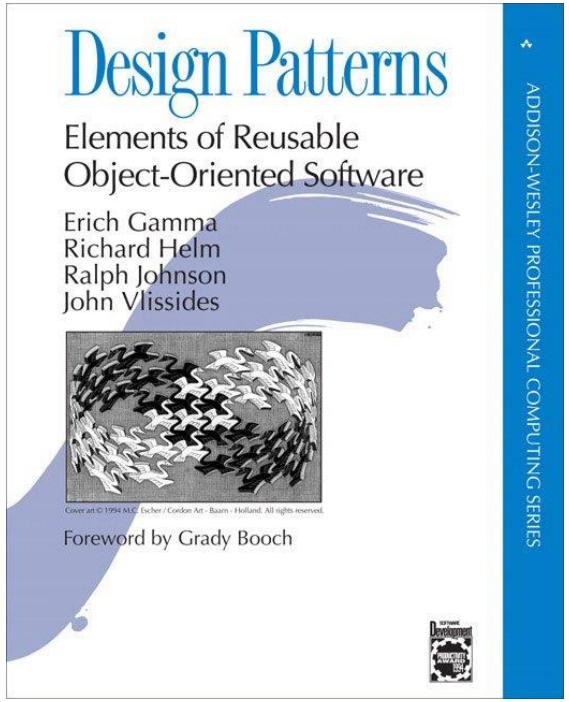
ARQUITETURA DE SOFTWARE

Introdução aos Padrões de Projeto

Geucimar Briatore
geucimar@up.edu.br

Atualizado em 10/2022

Livro de referência Gamma et al.



1994 (Gang of Four)

- Conceitos arquiteturais de Christopher Alexander, 1977;
- Foram catalogados 23 padrões;
- Gang of Four, GoF.

Exemplos dos padrões

https://refactoring.guru/pt-br/design-patterns/what-is-pattern

O que são padrões de projeto?

- **Padrões de projeto (design patterns)** são estruturas de classes ou soluções predefinidas de programação para problemas recorrentes na arquitetura e desenvolvimento de software;
- *"Cada padrão descreve um problema que ocorre repetidamente em nosso ambiente e, em seguida, define a solução adequada para esse problema, de uma maneira que você pode usar essa solução um milhão de vezes, sem nunca ter de fazê-la da mesma maneira duas vezes."* (Cristopher Alexander, 1977. Engenheiro e Arquiteto da Construção Civil);
- Os padrões promovem a reutilização de código e baixo acoplamento;
- O padrão não é um bloco de código, mas um conceito geral para resolver um problema em particular. Assim, não é possível encontrar um padrão e copiá-lo para dentro do software.

Padrões 1-6 (de 23)

1. **Abstract Factory** (Fábrica Abstrata): fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem expor suas classes concretas;
2. **Adapter** (Adaptador): converte a interface de uma classe em outra interface esperada pelos clientes. Permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis;
3. **Bridge** (Ponte): separa uma abstração da sua implementação, de modo que as duas possam variar independentemente;
4. **Builder** (Construtor): separa a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações;
5. **Chain of Responsibility** (Cadeia de Responsabilidade): evita o acoplamento do remetente de uma solicitação ao seu destinatário, dando a mais de um objeto a chance de tratar a solicitação. Encadeia os objetos receptores e passa a solicitação ao longo da cadeia até que um objeto a trate;
6. **Command** (Comando): encapsula uma solicitação como um objeto, desta forma permitindo que você parametrize clientes com diferentes solicitações, enfileire ou registre (log) solicitações e suporte operações que podem ser desfeitas;

Padrões 7-12 (de 23)

7. **Composite** (Composto): compõe objetos em estrutura de árvore para representar hierarquias do tipo partes-todo. Permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme;
8. **Decorator** (Decorador): atribui responsabilidades adicionais a um objeto dinamicamente. Fornecem uma alternativa flexível a subclasses para extensão da funcionalidade;
9. **Façade** (Fachada): fornece uma interface unificada para um conjunto de interfaces em um subsistema. Define uma interface de nível mais alto que torna o subsistema mais fácil de usar;
10. **Factory Method** (Método Fábrica): define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada. Permite a uma classe delegar a instanciação às subclasses;
11. **Flyweight** (Peso Mosca): usa compartilhamento para suportar grandes quantidades de objetos, de granularidade fina, de maneira eficiente;
12. **Interpreter** (Interpretador): dada uma linguagem, define uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças nessa linguagem;

Padrões 13-18 (de 23)

13. **Iterator** (Iterador): fornece uma maneira de acessar seqüencialmente os elementos de uma agregação de objetos sem expor sua representação subjacente;
14. **Mediator** (Mediador): define um objeto que encapsula a forma como um conjunto de objetos interage. Promove o acoplamento fraco ao evitar que os objetos se refiram explicitamente uns aos outros, permitindo que você varie suas interações independentemente;
15. **Memento** (Recordação): sem violar o encapsulamento, captura e externaliza um estado interno de um objeto, de modo que o mesmo possa posteriormente ser restaurado para este estado;
16. **Observer** (Observador): define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados;
17. **Prototype** (Protótipo): especifica os tipos de objetos a serem criados usando uma instância prototípica e criar novos objetos copiando esse protótipo;
18. **Proxy** (Procurador): fornece um objeto representante, ou um marcador de outro objeto, para controlar o acesso ao mesmo;

Padrões 19-23 (de 23)

19. **Singleton** (Objeto Único): garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela;
20. **State** (Estado): permite que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe;
21. **Strategy** (Estratégia): define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. Permite que o algoritmo varie independentemente dos clientes que o utilizam;
22. **Template Method** (Método Modelo): define o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses. Permite que as subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura;
23. **Visitor** (Visitante): representa uma operação a ser executada sobre os elementos da estrutura de um objeto. Permite que você defina uma nova operação sem mudar as classes dos elementos sobre os quais opera.

Divisões de classificação dos padrões

- **Padrões criacionais** se preocupam com o processo de criação de objetos aumentando a flexibilidade e a reutilização de código;
- **Padrões estruturais** ajudam na composição de classes ou de objetos, enquanto mantém as estruturas flexíveis e eficientes;
- **Padrões comportamentais** cuidam das interações e das responsabilidades dos objetos.

Classificação dos padrões

		Propósito		
		1. Criacional	***2. Estrutural***	***3. Comportamental***
Escopo	**Classe**	*Factory Method*	*Class Adapter*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Object Adapter* *Bridge* *Composite* *Decorator* *Facade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*