Design Patterns – Singleton - Lorenzon

Design Patterns

Padrões de Projeto são soluções reutilizáveis para problemas comuns que surgem repetidamente no desenvolvimento de software orientado a objetos.

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.

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.

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.

Uma analogia a um algoritmo é que ele seria 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: você pode ver o resultado e suas funcionalidades, mas a ordem exata de implementação depende de você.

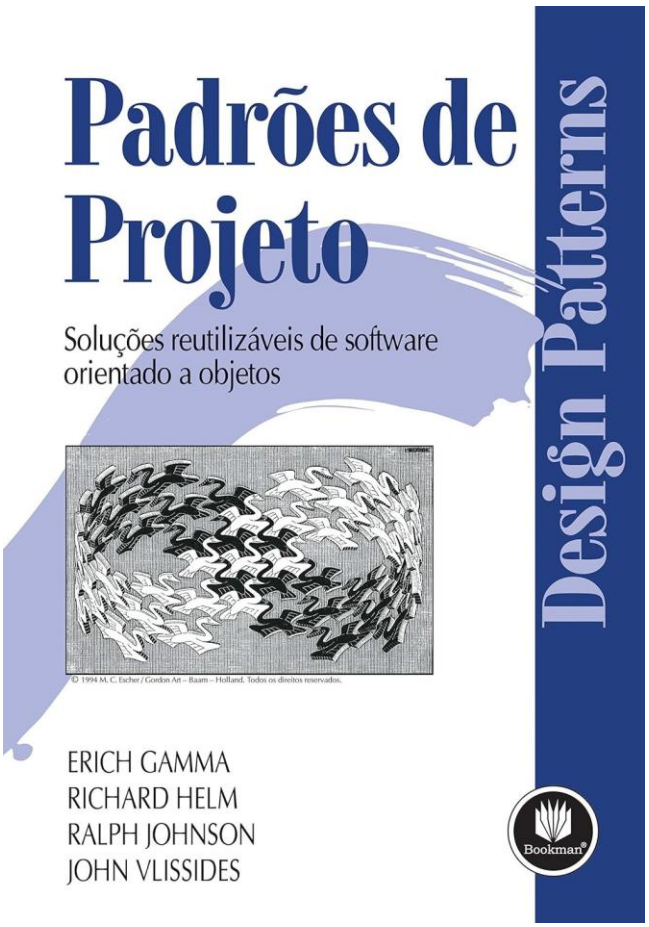
Assim como arquitetos usam plantas recorrentes para resolver certos desafios estruturais (escadas, portas, janelas), programadores usam padrões para resolver desafios de organização de código.

Origem e História

Popularizados pelo livro "Design Patterns: Elements of Reusable Object-Oriented Software" (1994), conhecido como Gang of Four (GoF).

Os padrões de projeto não são conceitos obscuros e sofisticados—bem o contrário. Os padrões são soluções típicas para problemas comuns em projetos orientados a objetos. Quando uma solução é repetida de novo e de novo em vários projetos, alguém vai eventualmente colocar um nome para ela e descrever a solução em detalhe. É basicamente assim que um padrão é descoberto.

O conceito de padrões foi primeiramente descrito por Christopher Alexander em 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.



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 projeto orientado a objetos e se tornou um best-seller rapidamente. Devido a seu longo título, as pessoas começaram a chamá-lo simplesmente de "o livro da Gangue dos Quatro (Gang of Four)" que logo foi simplificado para o "livro GoF".

Desde então, dúzias de outros padrões orientados a objetos foram descobertos. A "abordagem por padrões" se tornou muito popular em outros campos de programação, então muitos desses padrões agora existem fora do projeto orientado a objetos também.

Classificação dos Padrões (GoF)

Os padrões criacionais 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.

Criacionais (5): como criar objetos de forma flexível.

(**Singleton**, **Factory Method**, Abstract Factory, **Builder**, Prototype)

Estruturais(7): como organizar classes e objetos para formar estruturas maiores.

(**Adapter**, Bridge, Composite, Decorator, **Facade**, Flyweight, Proxy)

Comportamentais (11): como objetos interagem e se comunicam.

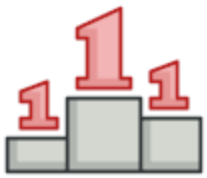
(Chain of Responsibility, Command, Interpreter, **Iterator**, Mediator, Memento, **Observer**, State, **Strategy**, Template Method, Visitor)

Por que devo aprender padrões?

A verdade é que você pode conseguir trabalhar como um programador por muitos anos sem saber sobre um único padrão. Muitas pessoas fazem exatamente isso. Ainda assim, contudo, você estará implementando alguns padrões mesmo sem saber. Então, por que gastar tempo para aprender sobre eles?

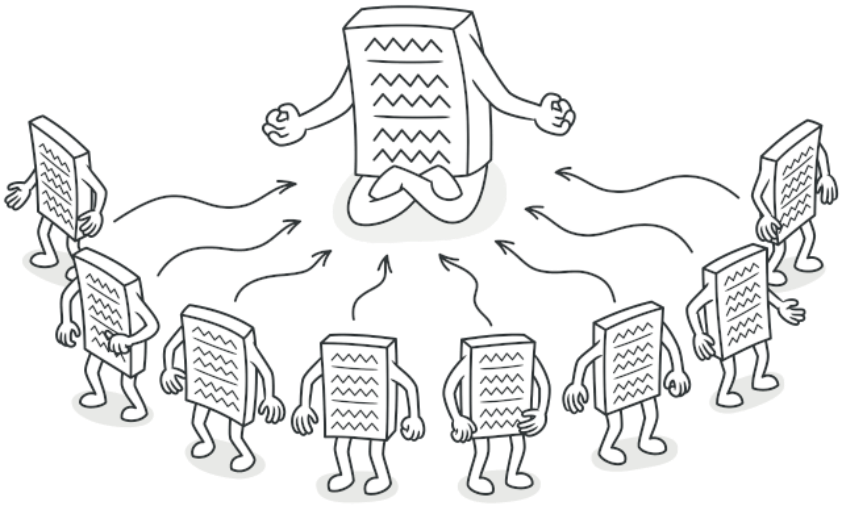
Os padrões de projeto são um kit de ferramentas para soluções tentadas e testadas para problemas comuns em projeto de software. Mesmo que você nunca tenha encontrado esses problemas, saber sobre os padrões é ainda muito útil porque eles ensinam como resolver vários problemas usando princípios de projeto orientado a objetos;

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.



Singleton

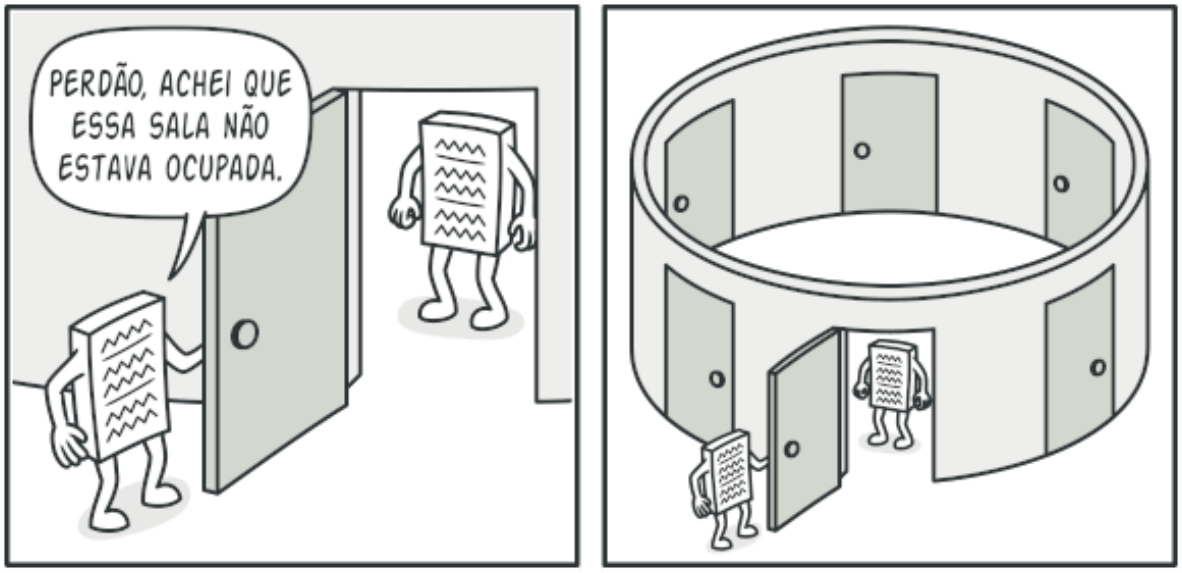
O Singleton é um padrão de projeto criacional que permite a você garantir que uma classe tenha apenas uma instância, enquanto provê um ponto de acesso global para essa instância.



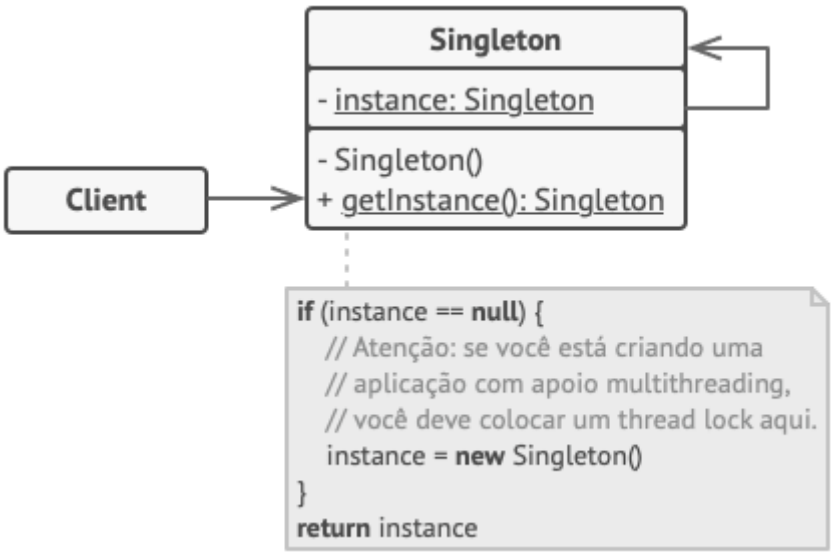
Como implementar

1. Adicione um campo privado estático na classe para o armazenamento da instância singleton;
2. Declare um método de criação público estático para obter a instância singleton;
3. Implemente a "inicialização preguiçosa" dentro do método estático. Ela deve criar um novo objeto na sua primeira chamada e colocá-lo no campo estático. O método deve sempre retornar aquela instância em todas as chamadas subsequentes;
4. Faça o construtor da classe ser privado. O método estático da classe vai ainda ser capaz de chamar o construtor, mas não os demais objetos;
5. Vá para o código cliente e substitua todas as chamadas diretas para o construtor do singleton com chamadas para seu método de criação estático.

```java
public class Configuracao {

    // 1. Atributo estatico que guarda a unica instância
    private static Configuracao instancia;

    // 2. 4. Construtor privado impede criacao direta
    private Configuracao() {
        System.out.println("Configuração iniciada.");
    }

    // 3. Metodo publico que retorna a instancia unica
    public static Configuracao getInstancia() {
        if (instancia == null) {
            instancia = new Configuracao();
        }
        return instancia;
    }
}
```



```
5.
public class App {
    //declarando a referência
    private static Configuracao c1;
    public static void main(String[] args) {
        //não tem construtor, mas método estático que resolve o padrão
        c1 = Configuracao.getInstancia();
        Configuracao c2 = Configuracao.getInstancia();

        //mesma instância
        System.out.println(c1.toString() + " == " + c2.toString());
    }
}
```



Todas as implementações do Singleton tem esses dois passos em comum:

- Fazer o construtor padrão privado, para prevenir que outros objetos usem o operador new com a classe singleton;
- Criar um método estático de criação que age como um construtor. Esse método chama o construtor privado por debaixo dos panos para criar um objeto e o salva em um campo estático. Todas as chamadas seguintes para esse método retornam o objeto em cache;
- Se o seu código tem acesso à classe singleton, então ele será capaz de chamar o método estático da singleton. Então sempre que aquele método é chamado, o mesmo objeto é retornado.

Quando Utilizar

Utilize o padrão Singleton quando uma classe em seu programa deve ter apenas uma instância disponível para todos seus clientes; por exemplo, um objeto de base de dados único compartilhado por diferentes partes do programa.

O padrão Singleton desabilita todos os outros meios de criar objetos de uma classe exceto pelo método especial de criação. Esse método tanto cria um novo objeto ou retorna um objeto existente se ele já tenha sido criado.

Utilize o padrão Singleton quando você precisa de um controle mais estrito sobre as variáveis globais. Ao contrário das variáveis globais, o padrão Singleton garante que há apenas uma instância de uma classe. Nada, a não ser a própria classe singleton, pode substituir a instância salva em cache.

Observe que você sempre pode ajustar essa limitação e permitir a criação de qualquer número de instâncias singleton. O único pedaço de código que requer mudanças é o corpo do método getInstance.

Prós

- Você pode ter certeza que uma classe só terá uma única instância;
- Você ganha um ponto de acesso global para aquela instância;
- O objeto singleton é inicializado somente quando for pedido pela primeira vez.

Contras

- Viola o princípio de responsabilidade única. O padrão resolve dois problemas de uma só vez;
- O padrão Singleton pode mascarar um design ruim, por exemplo, quando os componentes do programa sabem muito sobre cada um;
- O padrão requer tratamento especial em um ambiente multithreaded para que múltiplas threads não possam criar um objeto singleton várias vezes;
- Pode ser difícil realizar testes unitários do código cliente do Singleton porque muitos frameworks de teste dependem de herança quando produzem objetos simulados. Já que o construtor da classe singleton é privado e sobrescrever métodos estáticos é impossível na maioria das linguagem, você terá que pensar em uma maneira criativa de simular o singleton. Ou apenas não escreva os testes. Ou não use o padrão Singleton.

Existem muitos exemplos de Singleton nas bibliotecas principais do Java:

java.lang.Runtime#getRuntime(), **java.awt.Desktop#getDesktop()** e

java.lang.System#getSecurityManager()