#R4E
Software Developer

Design Patterns

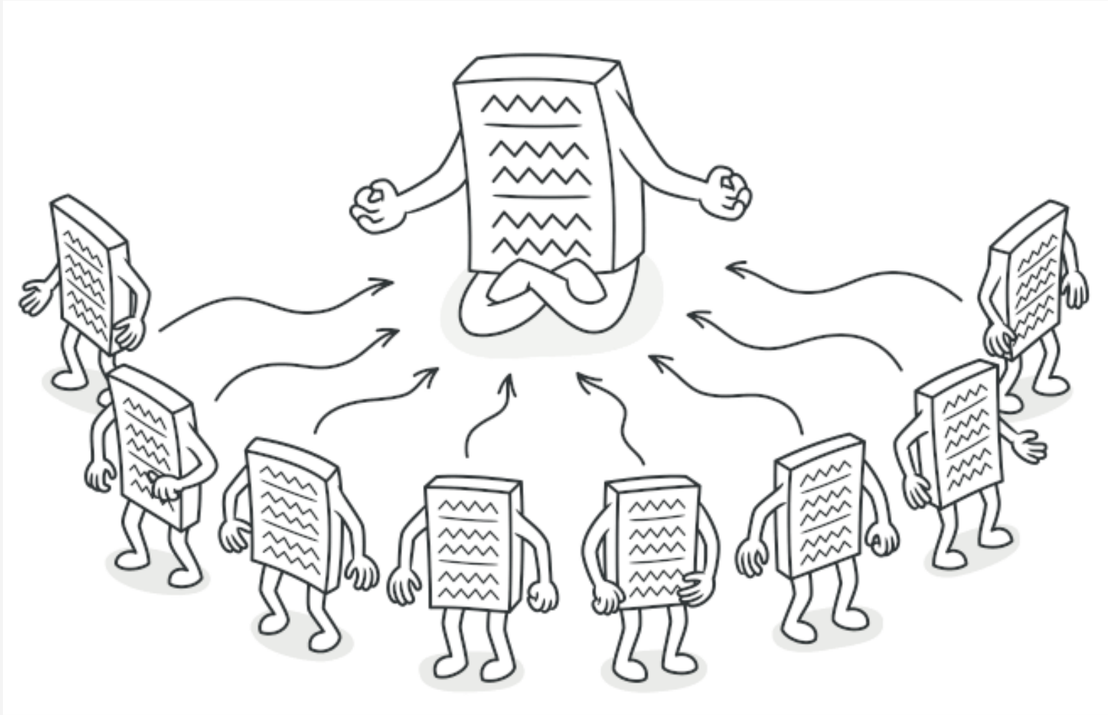
Singleton

Conteúdo

- Propósito
- Problema
- Solução
- Analogias
- Estrutura
- Aplicabilidade
- Como Implementar
- Prós e Contras
- Relações c/ Outros Padrões

Propósito

- O **Singleton** é um padrão de projeto criacional (Creational Pattern) que garante que uma classe tenha apenas uma instância, enquanto provê um ponto de acesso global para essa instância.



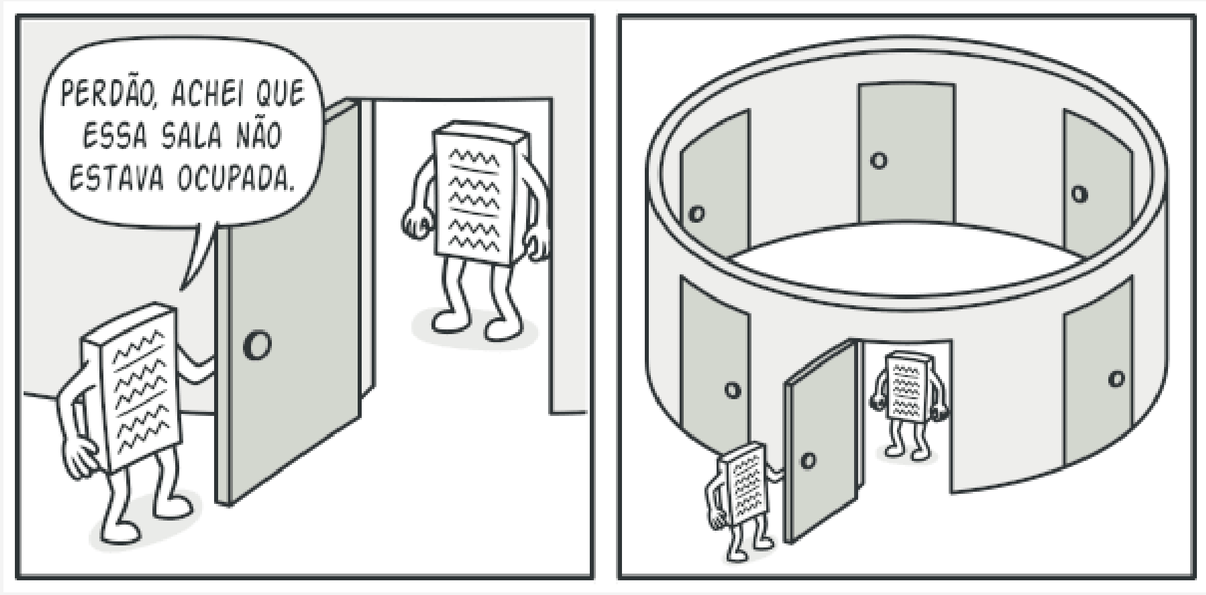
Problema

O padrão **Singleton** resolve dois problemas de uma só vez, violando o princípio de *responsabilidade única*:

- 1. Garantir que uma classe tenha **apenas uma única instância**. Porquè controlar quantas instâncias uma classe tem? A razão mais comum para isso é para controlar o acesso a algum recurso compartilhado—por exemplo, uma base de dados ou um arquivo.
- Funcionamento: imagine que criou um objeto, mas passado algum tempo decidiu criar um novo. Ao contrário de receber um objeto criado de fresco, recebe um que já foi criado.

Problema

- Esse comportamento é impossível implementar com um construtor regular uma vez que uma invocação do construtor retorna sempre um novo objeto, por defeito.



- Clientes podem nem dar conta que estão a lidar sempre com o mesmo objeto.

Problema

- Fornece um ponto de acesso global para aquela instância. Podemos utilizar, para isso, variáveis globais... Embora sejam inseguras, pois a qualquer momento podem ser acedidas e modificadas.
- Assim como uma variável global, o padrão Singleton permite aceder um objeto a partir de qualquer lugar no programa. Contudo, ele também **protege** aquela instância de ser **sobrescrita** por outro código.

Problema

- Há outro lado para esse problema: não é pretendido que o código que resolve o problema #1 fique espalhado pelo programa todo. É muito melhor tê-lo dentro de uma classe, especialmente se o resto do código já depende dela.
- Hoje em dia, o padrão Singleton tornou-se tão popular que as pessoas podem chamar algo de singleton mesmo se ele resolve apenas um dos problemas listados.

Solução

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

- Implementar 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, nos bastidores, para criar um objeto e salva num campo estático. Todas as invocações seguintes desse método retornam o objeto em cache.

Se o código tem acesso à classe singleton, então ele é capaz de chamar o método estático da singleton. Assim, sempre que aquele método é chamado, o mesmo objeto é retornado.

Analogia

Uma analogia comum para o padrão de design Singleton é a de um cofre. Imagina que tens um cofre numa sala, onde guardas um tesouro valioso. Queres ter a certeza de que apenas uma pessoa pode aceder ao tesouro de cada vez, para evitar roubos ou conflitos. Neste caso, o cofre é o objeto que possui um estado único (o tesouro) e o acesso a ele precisa de ser controlado. O padrão de design Singleton permite que apenas uma instância desse objeto seja criada, garantindo que todos os acessos ao tesouro sejam feitos através dessa única instância.

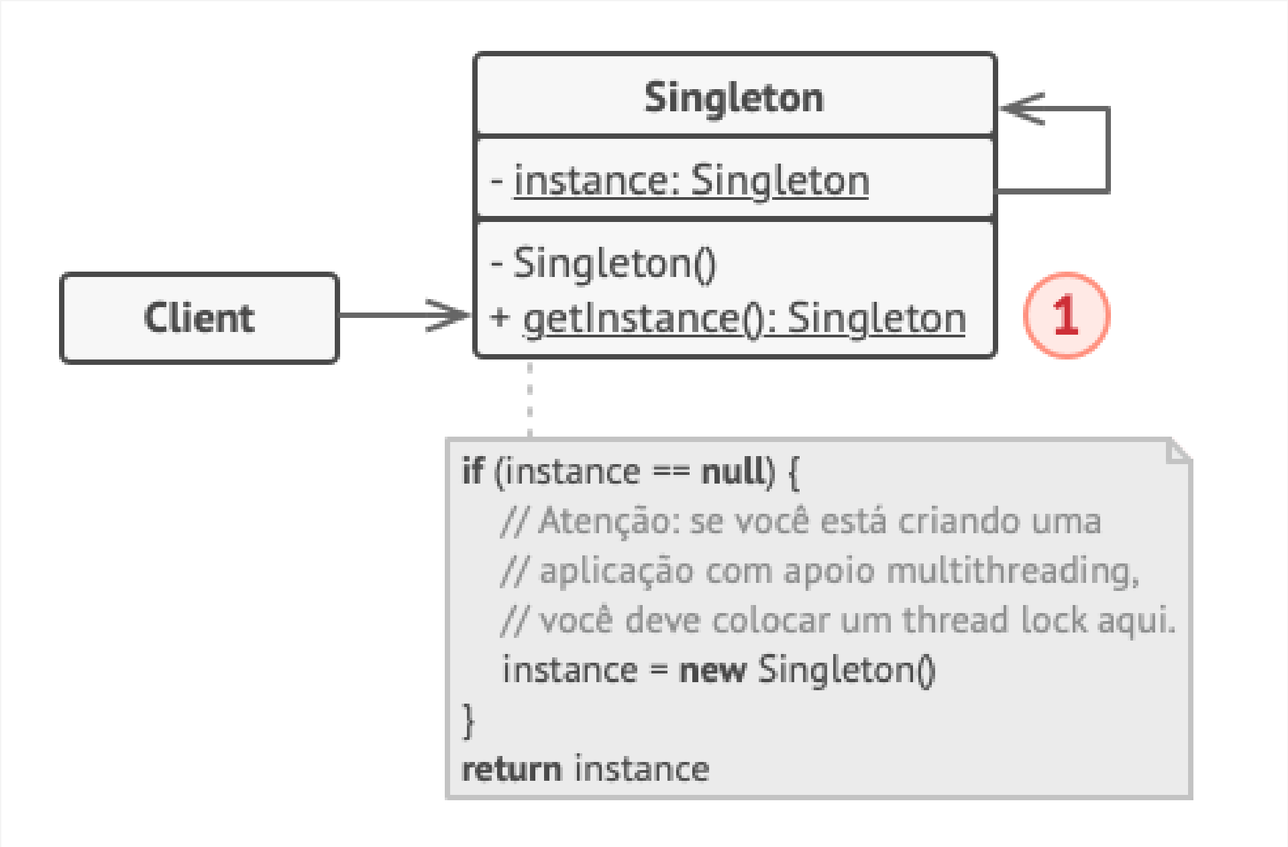
Analogia

Assim como no padrão Singleton, uma vez que uma pessoa está dentro da sala com o cofre, ela tem acesso exclusivo ao tesouro. Se outra pessoa tentar entrar na sala, ela não terá permissão para entrar até que a primeira pessoa saia e liberte o acesso.
Da mesma forma, quando usas o padrão Singleton num programa, garantes que apenas uma instância do objeto seja criada e que todas as partes do código utilizem essa mesma instância. Isso é útil em situações em que precisas de partilhar um recurso único entre diferentes partes do sistema, como uma conexão com uma base de dados ou configurações globais.

Estrutura

- A classe Singleton declara o método estático getInstance que retorna a mesma instância da sua própria classe.

- O construtor da singleton deve ser escondido do código cliente. Chamando o método getInstance deve ser o único modo de obter o objeto singleton.



Aplicabilidade

- Utilize o padrão Singleton quando uma classe no programa deve ter apenas uma instância disponível para todos seus clientes, por exemplo, um objeto de base de dados único partilhado 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.

Aplicabilidade

- **Utilize o padrão Singleton quando precisa de um controlo mais restrito 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 podemos sempre 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.

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 invocação e colocá-lo no campo estático. O método deve sempre retornar aquela instância em todas as chamadas subsequentes.
4. Faça do construtor da classe privado. O método estático da classe vai 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.

Prós

- Pode ter certeza que uma classe só terá uma única instância.
- 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 mau design, por exemplo, quando os componentes do programa sabem muito sobre cada um.
- O padrão requer tratamento especial num 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, é necessário pensar numa maneira criativa de simular o singleton. Ou não escreva os testes. Ou não use o padrão Singleton.

Relações c/ Outros Padrões

- Uma classe fachada pode frequentemente ser transformada numa singleton já que um único objeto fachada é suficiente na maioria dos casos.
- O Flyweight seria parecido com o Singleton se conseguissemos, de algum modo, reduzir todos os estados de objetos compartilhados para apenas um objeto flyweight. Mas há duas mudanças fundamentais entre esses padrões:
  - Deve haver apenas uma única instância singleton, enquanto que uma classe flyweight pode ter múltiplas instâncias com diferentes estados intrínsecos.
  - O objeto singleton pode ser mutável. Objetos flyweight são imutáveis.
- As Fábricas Abstratas, Construtores, e Protótipos podem todos ser implementados como Singletons.

#R4E
Software Developer

Design Patterns

Singleton