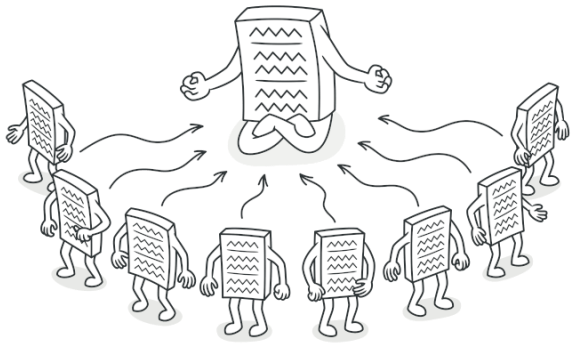
Singleton

Intenção

Singleton é um padrão de design criacional que permite garantir que uma classe tenha apenas uma instância, ao mesmo tempo em que fornece um ponto de acesso global a essa instância.



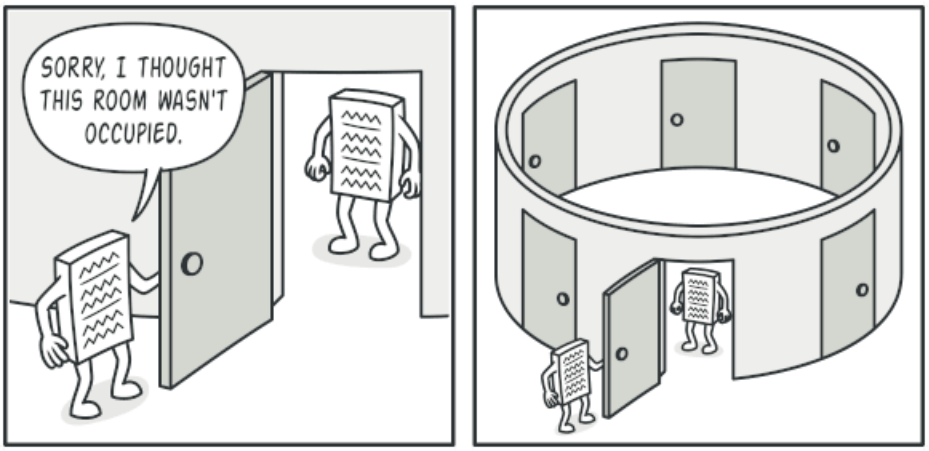
Problema

O padrão Singleton resolve dois problemas ao mesmo tempo quanto ao *Princípio da Responsabilidade Única* :

1. **Garanta que uma classe tenha apenas uma única instância**. Por que alguém iria querer controlar quantas instâncias uma classe tem? O motivo mais comum para isso é controlar o acesso a algum recurso compartilhado — por exemplo, um banco de dados ou um arquivo.

   É assim que funciona: imagine que você criou um objeto, mas depois de um tempo decidiu criar um novo. Em vez de receber um objeto novo, você receberá o que já criou.

   Observe que esse comportamento é impossível de implementar com um construtor regular, pois uma chamada de construtor **deve** sempre retornar um novo objeto por design.



Os clientes podem nem perceber que estão trabalhando com o mesmo objeto o tempo todo.

2. **Forneça um ponto de acesso global para essa instância**. Lembra daquelas variáveis globais que você (tudo bem, eu) usou para armazenar alguns objetos essenciais? Embora sejam muito úteis, elas também são muito inseguras, pois qualquer código pode potencialmente sobrescrever o conteúdo dessas variáveis e travar o aplicativo.

   Assim como uma variável global, o padrão Singleton permite que você acesse algum objeto de qualquer lugar no programa. No entanto, ele também protege essa instância de ser sobrescrita por outro código.

   Há outro lado desse problema: você não quer que o código que resolve o problema nº 1 fique espalhado por todo o seu programa. É muito melhor tê-lo dentro de uma classe, especialmente se o resto do seu código já depende dele.

Hoje em dia, o padrão Singleton se tornou tão popular que as pessoas podem chamar algo de *singleton* mesmo que resolva apenas um dos problemas listados.

Solução

Todas as implementações do Singleton têm estas duas etapas em comum:

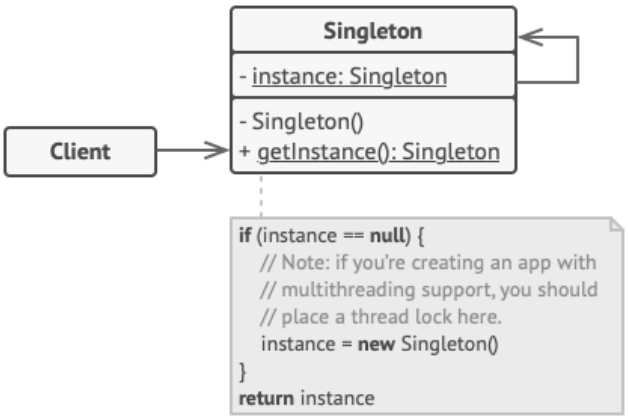
- Torne o construtor padrão privado, para evitar que outros objetos usem o `new`operador com a classe Singleton.
- Crie um método de criação estático que atue como um construtor. Por baixo dos panos, esse método chama o construtor privado para criar um objeto e salvá-lo em um campo estático. Todas as chamadas seguintes para esse método retornam o objeto em cache.

Se seu código tiver acesso à classe Singleton, então ele é capaz de chamar o método estático do Singleton. Então, sempre que esse método for chamado, o mesmo objeto sempre será retornado.

Analogia do mundo real

O governo é um excelente exemplo do padrão Singleton. Um país pode ter apenas um governo oficial. Independentemente das identidades pessoais dos indivíduos que formam governos, o título, “O Governo de X”, é um ponto global de acesso que identifica o grupo de pessoas no comando.

Estrutura



1. A classe **Singleton** declara o método estático `getInstance`que retorna a mesma instância de sua própria classe.

   O construtor do Singleton deve ser escondido do código do cliente. Chamar o `getInstance`método deve ser a única maneira de obter o objeto Singleton.

Pseudocódigo

Neste exemplo, a classe de conexão do banco de dados atua como um **Singleton** . Esta classe não tem um construtor público, então a única maneira de obter seu objeto é chamar o `getInstance`método. Este método armazena em cache o primeiro objeto criado e o retorna em todas as chamadas subsequentes.

```
// A classe Database define o método `getInstance` que permite que
// os clientes acessem a mesma instância de uma conexão de banco de dados
// em todo o programa.
class Database is
    // O campo para armazenar a instância singleton deve ser declarado estático.
    private static field instance: Database

    // O construtor do singleton deve ser sempre privado para
    // evitar chamadas diretas de construção com o operador `new`.
    private constructor Database() is
        // Alguns códigos de inicialização, como a conexão atual ao banco de dados
        // ...

    // O método estático que controla o acesso à instância singleton.
    public static method getInstance() is
        if (Database.instance == null) then
            acquireThreadLock() and then

                // Garante que a instância ainda não foi
                // inicializada por outro thread enquanto este
                // estava esperando a liberação do bloqueio.

                if (Database.instance == null) then
                    Database.instance = new Database()
        return Database.instance

    // Finalmente, qualquer singleton deve definir alguma lógica de negócios
    // que pode ser executado em sua instância.
    public method query(sql) is
        // Por exemplo, todas as consultas de banco de dados de um aplicativo passam
        // por esse método. Portanto, você pode colocar
        // lógica de limitação ou cache aqui.
        // ...

class Application is
    method main() is
        Database foo = Database.getInstance()
        foo.query("SELECT ...")
        // ...
        Database bar = Database.getInstance()
        bar.query("SELECT ...")
        // A variável `bar` conterá o mesmo objeto que a variável `foo`.
```

Aplicabilidade

Use o padrão Singleton quando uma classe em seu programa deve ter apenas uma única instância disponível para todos os clientes; por exemplo, um único objeto de banco de dados compartilhado por diferentes partes do programa.
O padrão Singleton desabilita todos os outros meios de criar objetos de uma classe, exceto o método de criação especial. Este método cria um novo objeto ou retorna um existente se ele já tiver sido criado.

Use o padrão Singleton quando precisar de controle mais rigoroso sobre variáveis globais.
Diferentemente de variáveis globais, o padrão Singleton garante que haja apenas uma instância de uma classe. Nada, exceto a própria classe Singleton, pode substituir a instância em cache.

Note que você sempre pode ajustar essa limitação e permitir a criação de qualquer número de instâncias Singleton. O único pedaço de código que precisa mudar é o corpo do `getInstance`método.

Como implementar

1. Adicione um campo estático privado à classe para armazenar a instância singleton.
2. Declare um método de criação estático público para obter a instância singleton.
3. Implemente "lazy initialization" dentro do método estático. Ele deve criar um novo objeto em sua primeira chamada e colocá-lo no campo estático. O método deve sempre retornar essa instância em todas as chamadas subsequentes.
4. Torne o construtor da classe privado. O método estático da classe ainda poderá chamar o construtor, mas não os outros objetos.
5. Revise o código do cliente e substitua todas as chamadas diretas ao construtor do *singleton* por chamadas ao seu método de criação estático.

Prós

- Você pode ter certeza de que uma classe tem apenas uma única instância.
- Você ganha um ponto de acesso global para essa instância.
- O objeto singleton é inicializado somente quando é solicitado pela primeira vez.

Contras

- Viola o *Princípio da Responsabilidade Única* . O padrão resolve dois problemas ao mesmo tempo.
- O padrão Singleton pode mascarar um design ruim, por exemplo, quando os componentes do programa sabem muito uns sobre os outros.
- O padrão requer tratamento especial em um ambiente multithread para que vários threads não criem um objeto singleton várias vezes.
- Pode ser difícil testar a unidade do código cliente do Singleton porque muitas estruturas de teste dependem de herança ao produzir objetos simulados. Como o construtor da classe singleton é privado e substituir métodos estáticos é impossível na maioria das linguagens, você precisará pensar em uma maneira criativa de simular o singleton. Ou simplesmente não escreva os testes. Ou não use o padrão Singleton.

Relações com outros padrões

- Uma classe **Facade pode frequentemente ser transformada em um Singleton,** já que um único objeto de fachada é suficiente na maioria dos casos.
- **Flyweight** seria parecido com **Singleton** se você de alguma forma conseguisse reduzir todos os estados compartilhados dos objetos para apenas um objeto flyweight. Mas há duas diferenças fundamentais entre esses padrões:
    1. Deve haver apenas uma instância Singleton, enquanto uma classe *Flyweight* pode ter várias instâncias com diferentes estados intrínsecos.
    2. O objeto *Singleton* pode ser mutável. Objetos Flyweight são imutáveis.
- **Resumo Fábricas** , **Construtores** e **Protótipos** podem ser implementados como **Singletons** .

Exemplos de código-fonte são encontrados em inúmeras linguagens de programação

C# C++ Go Java PHP Python Ruby Rust Swift TypeScript

Referências

Refactoring Guru. Disponível em <https://refactoring.guru/pt-br/design-patterns/singleton> Acesso em set de 2024.

GAMMA, Erich et al. **Padrões de projeto: elementos reutilizáveis de software orientado a objetos**. Porto Alegre, RS: Bookman, 2000.

LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo**. 3. ed. Porto Alegre, RS: Bookman, 2007.