refactoring.guru

# Singleton

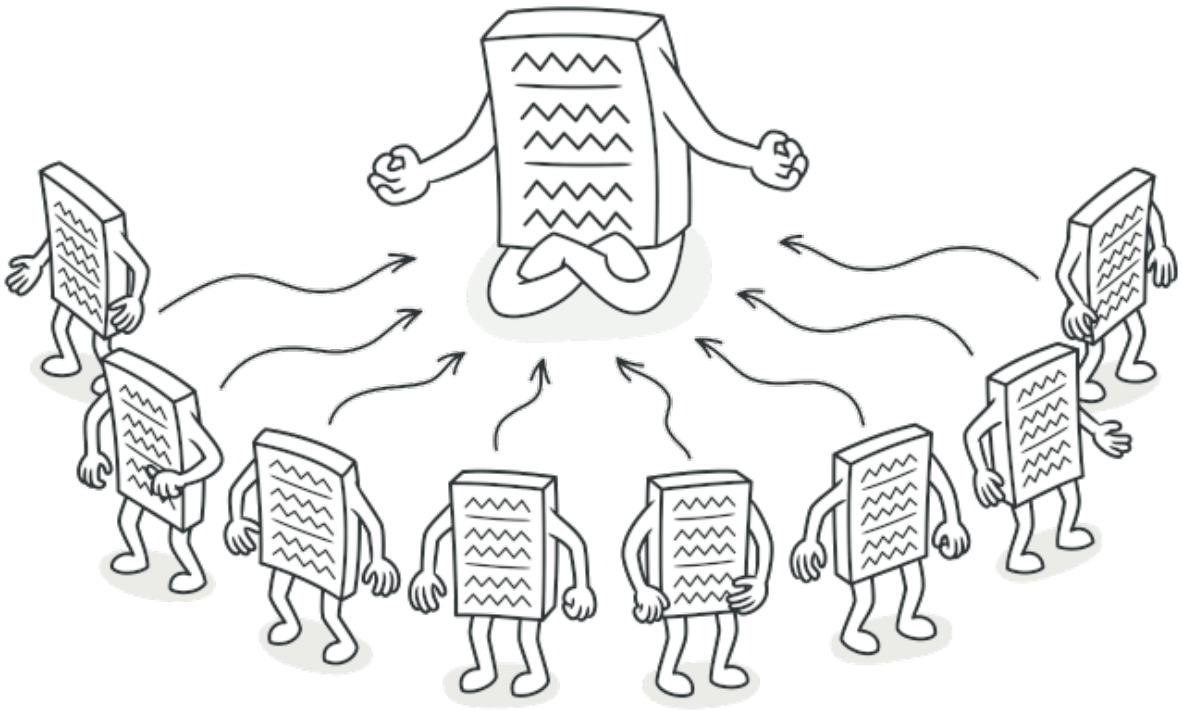
9–12 minutes

---

Também conhecido como: Carta única

## Propósito

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.



## Problema

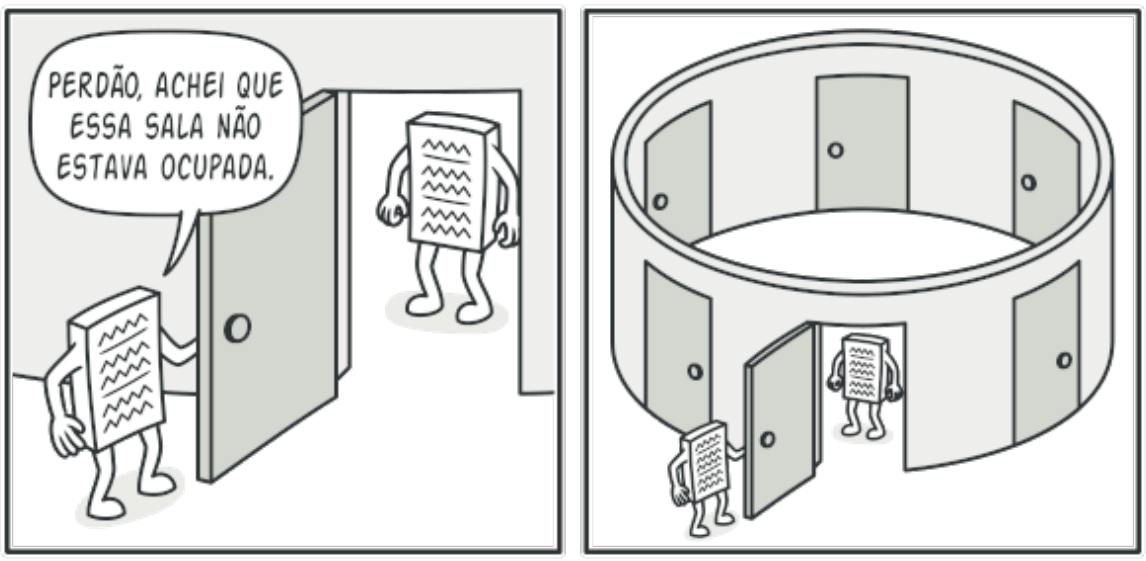
O padrão Singleton resolve dois problemas de uma só vez,

violando o *princípio de responsabilidade única*:

1. **Garantir que uma classe tenha apenas uma única instância**. Por que alguém iria querer 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.

   Funciona assim: imagine que você criou um objeto, mas depois de um tempo você decidiu criar um novo. Ao invés de receber um objeto fresco, você obterá um que já foi criado.

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




   Clientes podem não se dar conta que estão lidando com o mesmo objeto a todo momento.

2. **Fornece um ponto de acesso global para aquela instância**. Se lembra daquelas variáveis globais que você (tá bom, eu) usou para guardar alguns objetos essenciais? Embora sejam muito úteis, elas também são muito inseguras uma vez que qualquer

código pode potencialmente sobrescrever os conteúdos daquelas variáveis e quebrar a aplicação.

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

Há outro lado para esse problema: você não quer que o código que resolve o problema #1 fique espalhado por todo seu programa. É muito melhor tê-lo dentro de uma classe, especialmente se o resto do seu código já depende dela.

Hoje em dia, o padrão Singleton se tornou 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:

- 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.

## Analogia com o mundo real

O governo é um excelente exemplo de um 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 de acesso global que identifica o grupo de pessoas no command.

## Estrutura

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

## Pseudocódigo

Neste exemplo, a classe de conexão com a base de dados age como um **Singleton**. Essa classe não tem um construtor público, então a única maneira de obter seu objeto é chamando o método `getInstance`. Esse método coloca o primeiro objeto criado em cache e o retorna em todas as chamadas subsequentes.

```
// A classe Database define o método `getInstance` que permite
// clientes acessar a mesma instância de uma conexão a base de
// dados através do programa.
class Database is
    // O campo para armazenar a instância singleton deve ser
    // declarado como estático.
    private static field instance: Database
```

```
// O construtor do singleton devem sempre ser privado para
// prevenir chamadas diretas de construção com o operador
// `new`.
private constructor Database() is
    // Algum código de inicialização, tal como uma conexão
    // com um servidor de base de dados.


// O método estático que controla acesso à instância do
// singleton
public static method getInstance() is
    if (Database.instance == null) then
        acquireThreadLock() and then
            // Certifique que a instância ainda não foi
            // inicializada por outra thread enquanto está
            // estiver esperando pela liberação do `lock`.
            if (Database.instance == null) then
                Database.instance = new Database()
    return Database.instance

// Finalmente, qualquer singleton deve definir alguma lógica
// de negócio que deve ser executada em sua instância.
public method query(sql) is
    // Por exemplo, todas as solicitações à base de dados de
    // uma aplicação passam por esse método. Portanto, você
    // pode colocar a lógica de throttling 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` vai conter o mesmo objeto que a
    // variável `foo`.
```

## Aplicabilidade

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`.

## 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.

## Prós e contras

- 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.
- 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.

## Relações com outros padrões

- Uma classe fachada pode frequentemente ser transformada em uma singleton já que um único objeto fachada é suficiente na maioria dos casos.
- O Flyweight seria parecido com o Singleton se você, de algum modo, reduzisse todos os estados de objetos compartilhados para apenas um objeto flyweight. Mas há duas mudanças fundamentais entre esses padrões:

1. Deve haver apenas uma única instância singleton, enquanto que uma classe *flyweight* pode ter múltiplas instâncias com diferentes estados intrínsecos.
2. O objeto *singleton* pode ser mutável. Objetos flyweight são imutáveis.

- As Fábricas Abstratas, Construtores, e Protótipos podem todos ser implementados como Singletons.