**Alunos: Lucas, Rogerio, Thiago**

Singleton:

assegura que somente um objeto de uma determinada classe seja criado em todo o projeto;

resolve dois problemas de uma só vez, violando o princípio de responsabilidade única:

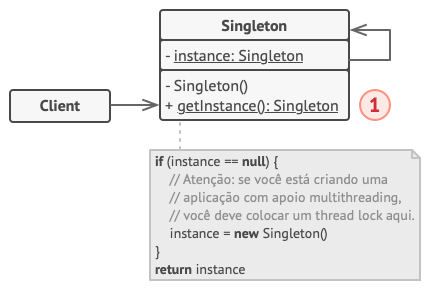
P1.Garantir que uma classe tenha apenas uma única instância, para controlar o acesso a algum recurso compartilhado exemplo, uma base de dados ou um arquivo.

P2. Fornece um ponto de acesso global para aquela instância. o padrão Singleton permite que você acesse algum objeto de qualquer lugar no programa. Fornece um ponto de acesso global para aquela instância.

Solução

* 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.

## **Estrutura**



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.

## **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 tiver 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.

**Prototype**

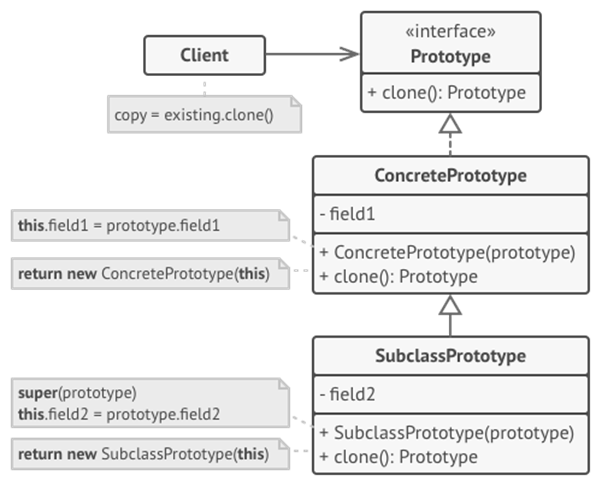
O Prototype é um padrão que permite copiar objetos existentes sem fazer seu código ficar dependente de suas classes.

Copiar um objeto “do lado de fora” [nem sempre](https://refactoring.guru/cargo-cult) é possível. Há ainda mais um problema com a abordagem direta. Uma vez que você precisa saber a classe do objeto para criar uma cópia, seu código se torna dependente daquela classe, pode ocorrer também de você só saber a interface que o objeto segue, mas não sua classe concreta, quando, por exemplo, um parâmetro em um método aceita quaisquer objetos que seguem uma interface.

O padrão Prototype delega o processo de clonagem para o próprio objeto que está sendo clonado. O padrão declara uma interface comum para todos os objetos que suportam clonagem. Essa interface permite que você clone um objeto sem acoplar seu código à classe daquele objeto.

A implementação do método clonar é muito parecida em todas as classes. O método cria um objeto da classe atual e carrega todos os valores de campo para do antigo objeto para o novo. Um objeto que suporta clonagem é chamado de um *protótipo*. Quando seus objetos têm dúzias de campos e centenas de possíveis configurações, cloná-los pode servir como uma alternativa à subclasses.

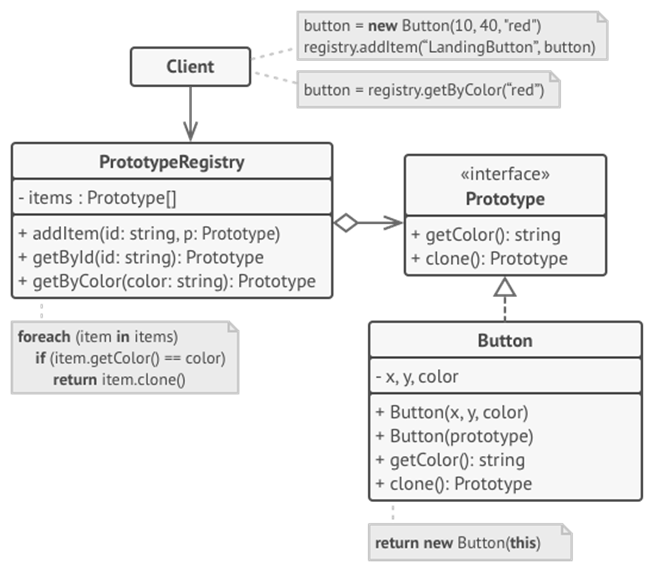
**Implementação básica**

1. A interface Protótipo declara os métodos de clonagem. Na maioria dos casos é apenas um método clonar.

2. A classe Protótipo Concreta implementa o método de clonagem. Além de copiar os dados do objeto original para o clone, esse método também pode lidar com alguns casos específicos do processo de clonagem relacionados a clonar objetos ligados, desfazendo dependências recursivas, etc.

3. O Cliente pode produzir uma cópia de qualquer objeto que segue a interface do protótipo

**Implementação do registro do protótipo**



O Registro do Protótipo fornece uma maneira fácil de acessar protótipos de uso frequente.

Ele salva um conjunto de objetos pré construídos que estão prontos para serem copiados.

Deve ser utilizado o padrão Prototype quando seu código não dependera de classes concretas de objetos que você precisa copiar.

Isso acontece muito quando o código funciona com objetos passados para você por um código de terceiros através de alguma interface. As classes concretas desses objetos são desconhecidas, e você não pode depender delas mesmo que quisesse.

O padrão Prototype fornece o código cliente com uma interface geral para trabalhar com todos os objetos que suportam clonagem. Essa interface faz o código do cliente ser independente das classes concretas dos objetos que ele clona. O padrão deve ser utilizado quando você precisa reduzir o número de subclasses que somente diferem na forma que inicializam seus respectivos objetos.

Alguém pode ter criado essas subclasses para ser capaz de criar objetos com uma configuração específica. O padrão Prototype permite que você use um conjunto de objetos pré construídos, configurados de diversas formas, como protótipos, portanto ao invés de instanciar uma subclasse que coincide com alguma configuração, o cliente pode simplesmente procurar por um protótipo apropriado e cloná-lo.

**Implementação**

1. Crie uma interface protótipo e declare o método *clonar* nela. Ou apenas adicione o método para todas as classes de uma hierarquia de classes existente, se você tiver uma.
2. Uma classe protótipo deve definir o construtor alternativo que aceita um objeto daquela classe como um argumento. O construtor deve copiar os valores de todos os campos definidos na classe do objeto passado para a nova instância recém criada. Se você está mudando uma subclasse, você deve chamar o construtor da classe pai para permitir que a superclasse lide com a clonagem de seus campos privados. Se a sua linguagem de programação não suporta sobrecarregamento de métodos, você pode definir um método especial para copiar os dados do objeto.
3. O método de clonagem geralmente consiste em apenas uma linha: executando um operador *new* com a versão protótipo do construtor. Observe que toda classe deve explicitamente sobrescrever o método de clonagem and usar sua própria classe junto com o operador *new*. Do contrário, o método de clonagem pode produzir um objeto da classe superior.
4. Opcionalmente, crie um registro protótipo centralizado para armazenar um catálogo de protótipos usados com frequência. Você pode implementar o registro como uma nova classe factory ou colocá-lo na classe protótipo base com um método estático para recuperar o protótipo. Esse método deve procurar por um protótipo baseado em critérios de busca que o código cliente passou para o método. O critério pode ser tanto uma string ou um complexo conjunto de parâmetros de busca. Após o protótipo apropriado ser encontrado, o registro deve cloná-lo e retornar a cópia para o cliente. Por fim, substitua as chamadas diretas para os construtores das subclasses com chamadas para o método factory do registro do protótipo.

**Vantagens e desvantagens**

· Você pode clonar objetos sem acoplá-los a suas classes concretas.

· Você pode se livrar de códigos de inicialização repetidos em troca de clonar protótipos pré-construídos.

· Você pode produzir objetos complexos mais convenientemente.

· Você tem uma alternativa para herança quando lidar com configurações pré determinadas para objetos complexos.

· Clonar objetos complexos que têm referências circulares pode ser bem complicado.