Prototype

Fernando Lucas da Silva

Necessidade

Em algumas situações, é necessário criarmos objetos “cópias” de objetos já existentes.

Normalmente, isso se deve ao fato de que esse novo objeto terá **outro uso**, podendo inclusive ser **modificado**, mas o que ocorrer com ele **não deve refletir** no objeto original.

Isso significa, de fato, alocar uma **nova região** de memória para essa cópia (o famoso *new*), uma vez que o objeto original não deve ser referenciado, seja por composição, herança ou polimorfismo.

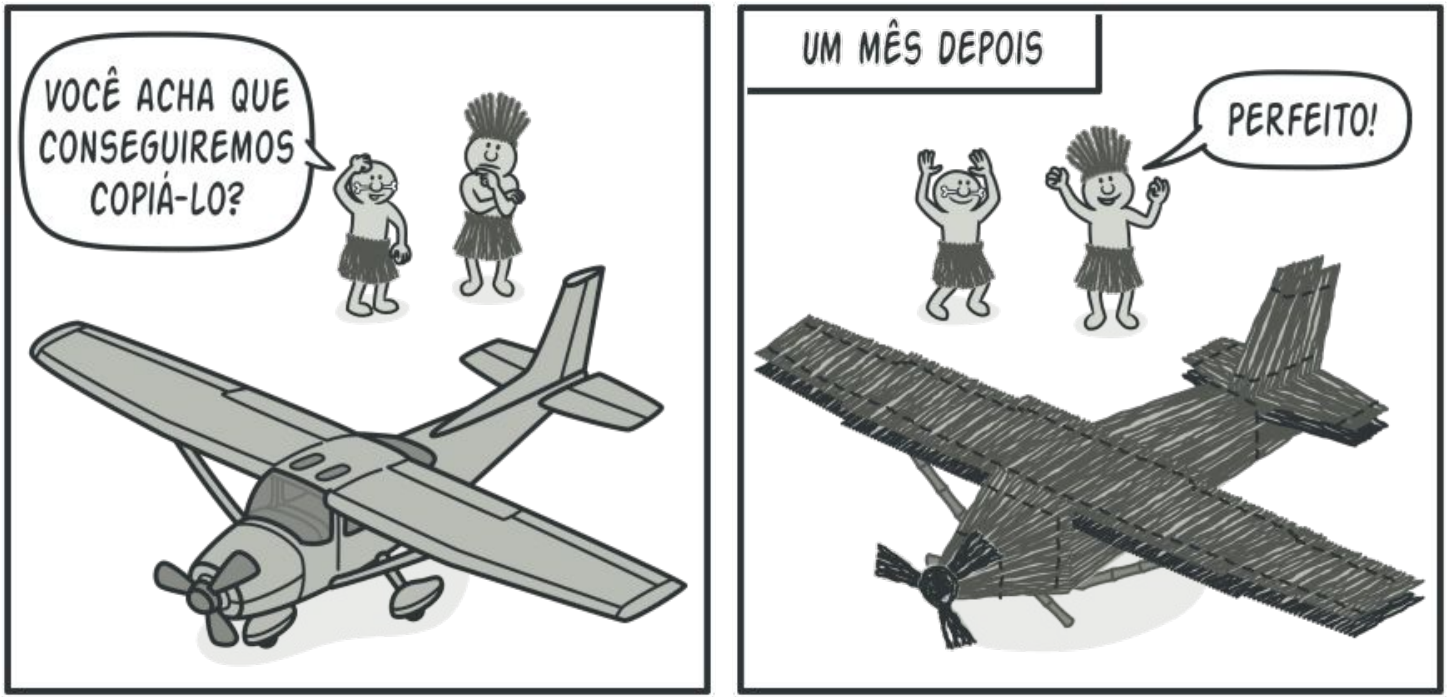
Problema

Existe uma forma simples e trivial de gerar cópias de um objeto, que é conhecida como **boilerplate**. Basicamente, consiste em relacionar campo a campo de um objeto para outro.

Porém, ela **não pode** ser usada em qualquer lugar do código. Além de, muitas vezes, sujar o código da lógica negocial, ainda pode (e provavelmente vai) esbarrar em **atributos privados** ou protegidos.

Quando estamos falando de criar uma **cópia fiel** de um objeto, os atributos privados **não podem ser ignorados**. Como faz pra resolver?

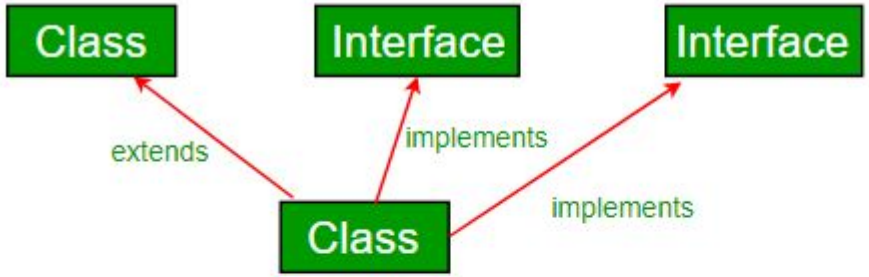
Exemplo do Refactoring Guru



Ainda tem mais problemas?

Seguindo a lógica de, por exemplo, criar um **método** para fazer cópias completas de uma classe A, geramos **acoplamento**. Ou seja, para copiar a classe B, é necessário escrever um novo método.

Além de que, pelo **polimorfismo**, às vezes não sabemos a classe **concreta** representada por aquele objeto. Ele pode estar sendo tratado como uma **interface** naquele ponto do código, ou até mesmo uma **classe mãe**. E aí?



Aí entra o Prototype

Para resolver o problema, temos o padrão **Prototype**. É um padrão de projeto **criacional** voltado para a clonagem de objetos.

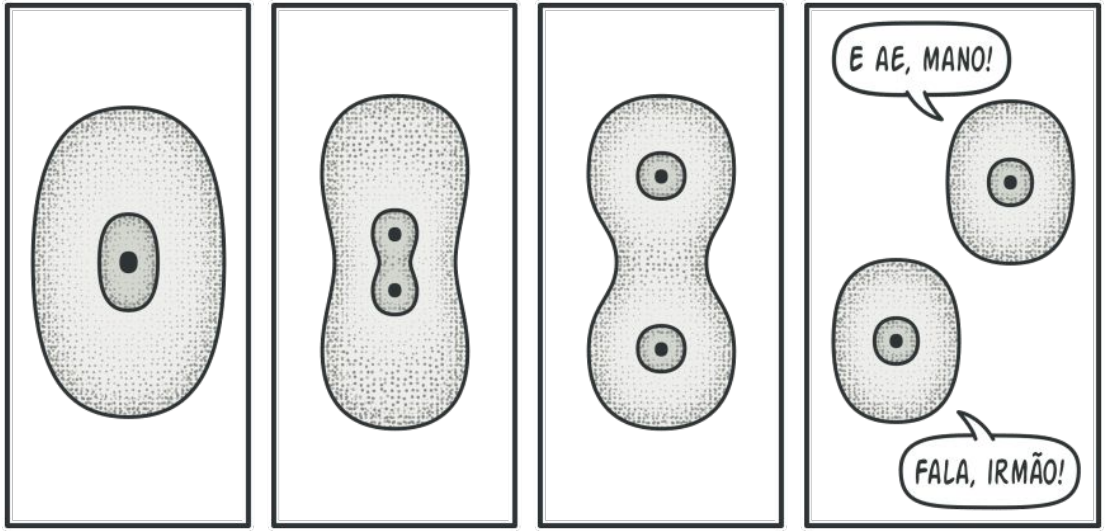
A ideia é delegar esse mapeamento a **cada classe**. Dessa forma, todas as classes "**clonáveis**" já saberão fazer isso.

Criamos uma interface (por exemplo, **Cloneable**), com o método **clone**(). Toda classe que **implementar** a interface, vai precisar desenvolver sua **lógica de cópia**, enxergando inclusive os atributos privados. Logo, estará seguindo o padrão Prototype e pode ser chamada de **protótipo**.

Como clonar, afinal?

Em um programa que usa Prototype, no qual podemos ter vários objetos **protótipos** por toda parte, é super fácil clonar.

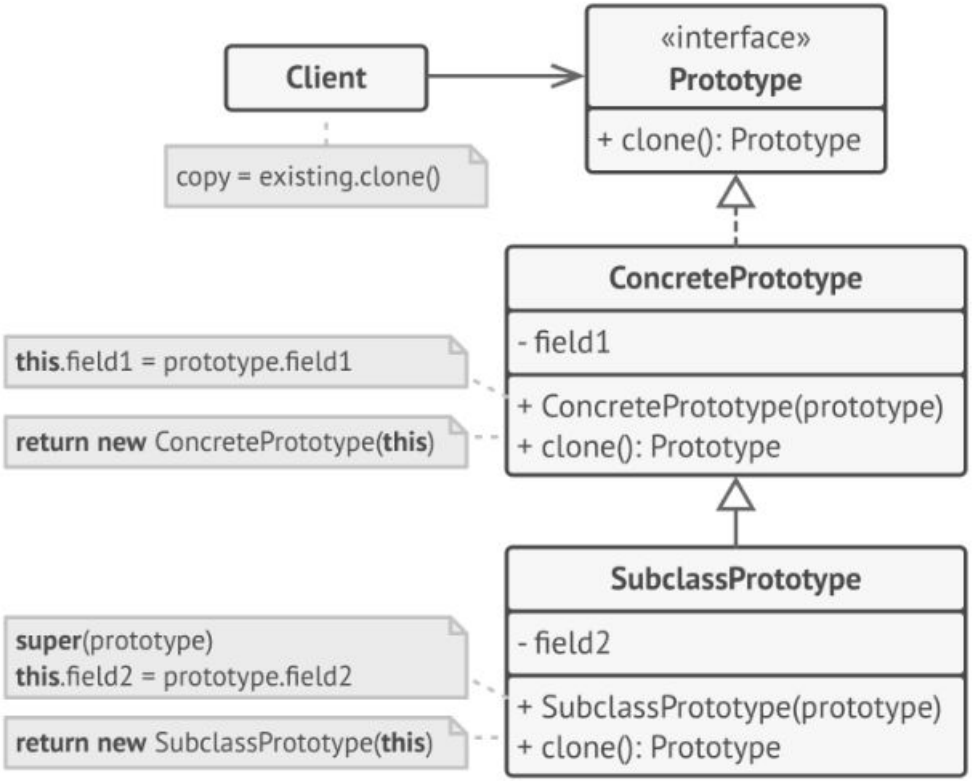
O objeto original terá aquele método **clone()**, que já vai ter toda lógica implementada e simplesmente vai retornar um **novo objeto**, porém **idêntico** a ele, como queremos.



Estrutura (implementação)

Nesse exemplo do Refactoring Guru, observamos:

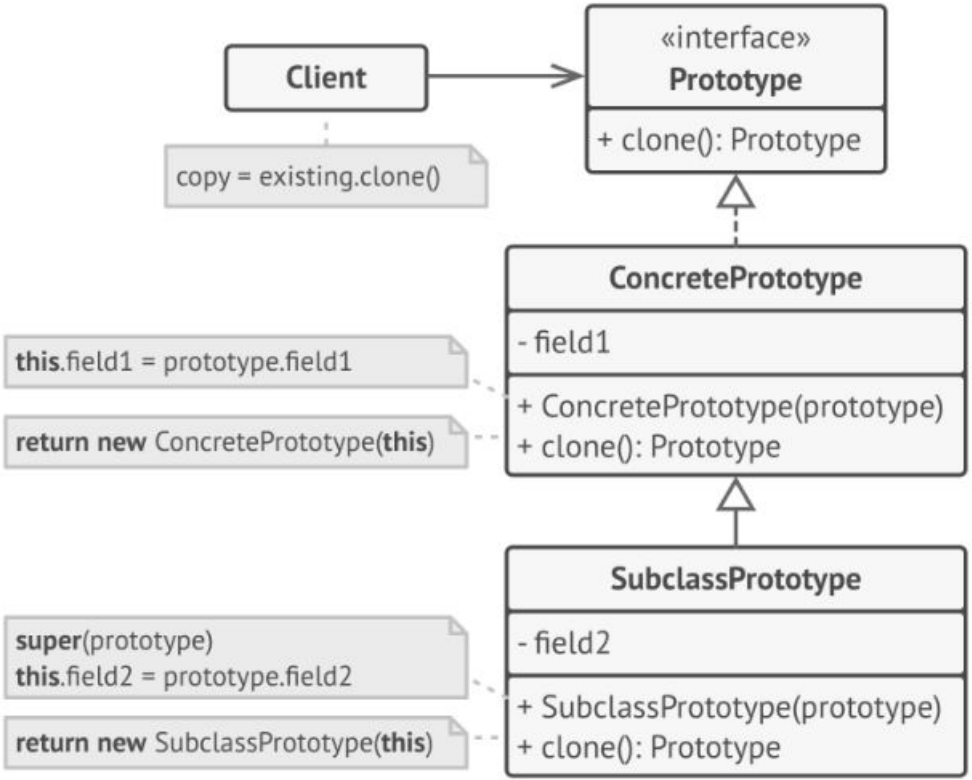
- Interface Prototype
- Classe concreta com um **construtor** que recebe um objeto de mesma classe
- Implementação do **clone()** chamando esse construtor, passando a si mesmo



Estrutura (herança)

A herança funciona de forma recursiva:

- Uma classe filha sobrescreve o método **clone()** da classe mãe
- Na nova lógica, primeiro chama o método da classe mãe, depois copia demais atributos



Vamos à prática!