refactoring.guru

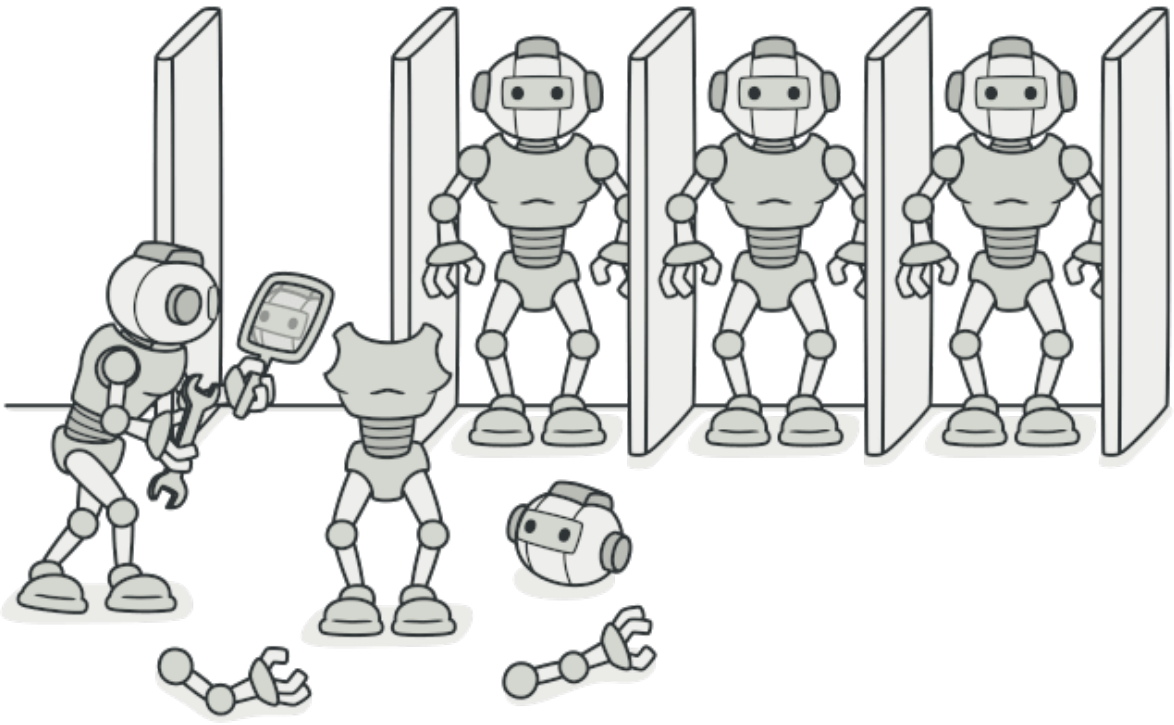
Prototype

12–16 minutes

Também conhecido como: Protótipo, Clone

Propósito

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



Problema

Digamos que você tenha um objeto, e você quer criar uma cópia

exata dele. Como você o faria? Primeiro, você tem que criar um novo objeto da mesma classe. Então você terá que ir por todos os campos do objeto original e copiar seus valores para o novo objeto.

Legal! Mas tem uma pegadinha. Nem todos os objetos podem ser copiados dessa forma porque alguns campos de objeto podem ser privados e não serão visíveis fora do próprio objeto.

Copiando um objeto “do lado de fora” nem sempre 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. Se a dependência adicional não te assusta, tem ainda outra pegadinha. Algumas vezes você só sabe 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.

Solução

O padrão Prototype delega o processo de clonagem para o próprio objeto que está sendo clonado. O padrão declara um

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. Geralmente, tal interface contém apenas um único método `clonar`.

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. Você pode até mesmo copiar campos privados porque a maioria das linguagens de programação permite objetos acessar campos privados de outros objetos que pertençam a mesma classe.

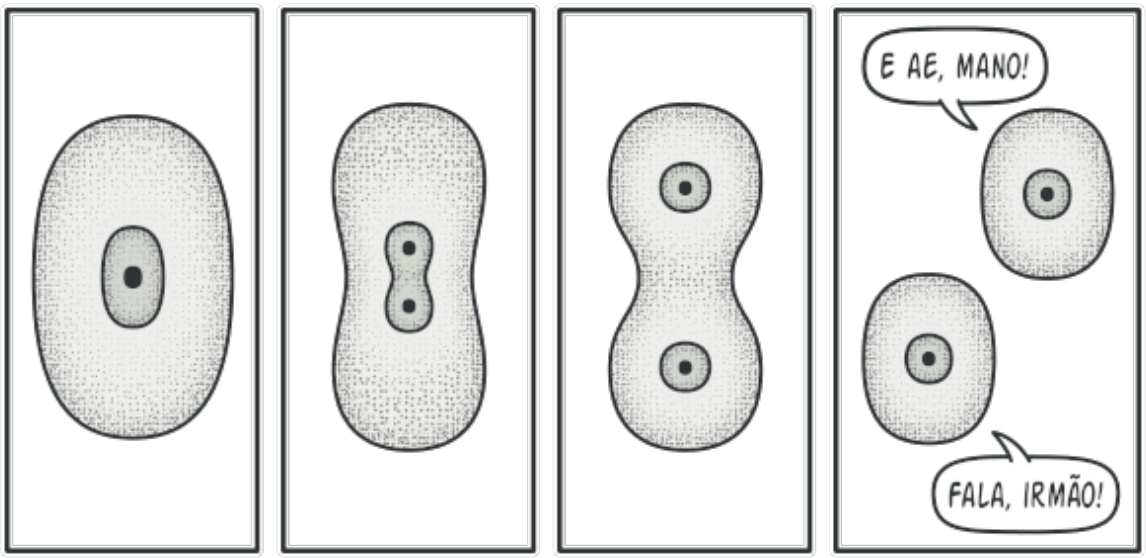
Um objeto que suporta clonagem é chamado de um *protótipo*. Quando seus objetos têm dúzias de campos e centenas de possíveis configurações, cloná-los pode servir como uma alternativa à subclasses.

Pré construir protótipos pode ser uma alternativa às subclasses.

Funciona assim: você cria um conjunto de objetos, configurados de diversas formas. Quando você precisa um objeto parecido com o que você configurou, você apenas clona um protótipo ao invés de construir um novo objeto a partir do nada.

Analogia com o mundo real

Na vida real, os protótipos são usados para fazer diversos testes antes de se começar uma produção em massa de um produto. Contudo, nesse caso, os protótipos não participam de qualquer produção, ao invés disso fazem um papel passivo.



A divisão de uma célula.

Já que protótipos industriais não se copiam por conta própria, uma analogia ao padrão é o processo de divisão celular chamado mitose (biologia, lembra?). Após a divisão mitótica, um par de células idênticas são formadas. A célula original age como um protótipo e tem um papel ativo na criação da cópia.

Estrutura

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.

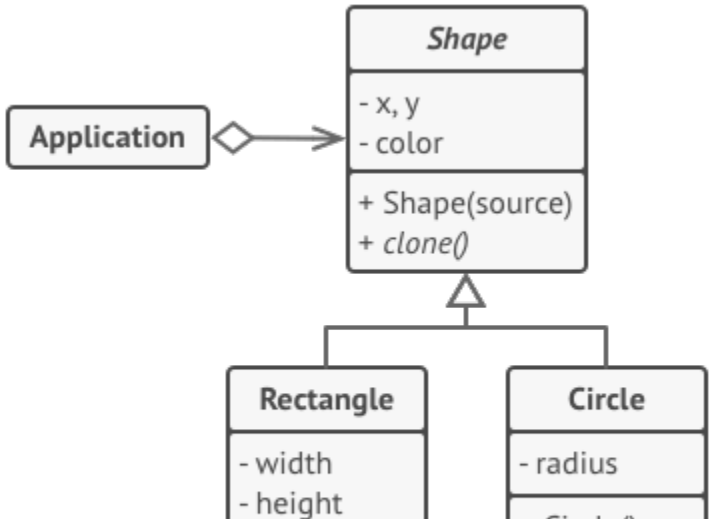
3. O **Cliente** pode produzir uma cópia de qualquer objeto que segue a interface do protótipo.

Implementação do registro do protótipo

1. 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. O registro de protótipo mais simples é um hashmap `nome → protótipo`. Contudo, se você precisa de um melhor critério de busca que apenas um nome simples, você pode construir uma versão muito mais robusta do registro.

Pseudocódigo

Neste exemplo, o padrão **Prototype** permite que você produza cópias exatas de objetos geométricos, sem acoplamento com o código das classes deles.



Clonando um conjunto de objetos que pertencem à uma hierarquia de classe.

Todas as classes de formas seguem a mesma interface, que fornece um método de clonagem. Uma subclasse pode chamar o método de clonagem superior antes de copiar seus próprios valores de campo para o objeto resultante.

```
// Protótipo base.
abstract class Shape is
    field X: int
    field Y: int
    field color: string

    // Um construtor normal.
    constructor Shape() is


    // O construtor do protótipo. Um objeto novo é inicializado
    // com valores do objeto existente.
    constructor Shape(source: Shape) is
        this()
        this.X = source.X
        this.Y = source.Y
        this.color = source.color

    // A operação de clonagem retorna uma das subclasses Shape.
    abstract method clone():Shape
```

```
// Protótipo concreto. O método de clonagem cria um novo objeto
// e passa ele ao construtor. Até o construtor terminar, ele tem
// uma referência ao clone fresco. Portanto, ninguém tem acesso
// ao clone parcialmente construído. Isso faz com que o clone
// resultante seja consistente.
class Rectangle extends Shape is
    field width: int
    field height: int

    constructor Rectangle(source: Rectangle) is
        // Uma chamada para o construtor pai é necessária para
        // copiar campos privados definidos na classe pai.
        super(source)
        this.width = source.width
        this.height = source.height

    method clone():Shape is
        return new Rectangle(this)


class Circle extends Shape is
    field radius: int

    constructor Circle(source: Circle) is
        super(source)
        this.radius = source.radius

    method clone():Shape is
```

```
        return new Circle(this)


// Em algum lugar dentro do código cliente.
class Application is
    field shapes: array of Shape

    constructor Application() is
        Circle circle = new Circle()
        circle.X = 10
        circle.Y = 10
        circle.radius = 20
        shapes.add(circle)

        Circle anotherCircle = circle.clone()
        shapes.add(anotherCircle)
        // A variável `anotherCircle` contém uma cópia exata do
        // objeto `circle`.

        Rectangle rectangle = new Rectangle()
        rectangle.width = 10
        rectangle.height = 20
        shapes.add(rectangle)

    method businessLogic() is
        // O protótipo arrasa porque permite que você produza
        // uma cópia de um objeto sem saber coisa alguma sobre
        // seu tipo.
        Array shapesCopy = new Array of Shapes.
```

```
// Por exemplo, nós não sabemos os elementos exatos no
// vetor shapes. Tudo que sabemos é que eles são todos
// shapes. Mas graças ao polimorfismo, quando nós
// chamamos o método `clone` em um shape, o programa
// checa sua classe real e executa o método de clonagem
// apropriado definido naquela classe. É por isso que
// obtemos clones apropriados ao invés de um conjunto de
// objetos Shape simples.
foreach (s in shapes) do
    shapesCopy.add(s.clone())

// O vetor `shapesCopy` contém cópias exatas dos filhos
// do vetor `shape`.
```

Aplicabilidade

Utilize o padrão Prototype quando seu código não deve depender de classes concretas de objetos que você precisa copiar.

Isso acontece muito quando seu código funciona com objetos passados para você de 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.

Utilize o padrão 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.

Ao invés de instanciar uma subclasse que coincide com alguma configuração, o cliente pode simplesmente procurar por um protótipo apropriado e cloná-lo.

Como implementar

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. O construtor é um local mais conveniente para se fazer isso porque ele entrega o objeto resultante logo depois que você chamar o operador `new`.
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.

Prós e contras

- 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.

Relações com outros padrões

- Muitos projetos começam usando o Factory Method (menos complicado e mais customizável através de subclasses) e evoluem para o Abstract Factory, Prototype, ou Builder (mais flexíveis, mas mais complicados).
- Classes Abstract Factory são quase sempre baseadas em um conjunto de métodos fábrica, mas você também pode usar o Prototype para compor métodos dessas classes.
- O Prototype pode ajudar quando você precisa salvar cópias de comandos no histórico.
- Projetos que fazem um uso pesado de Composite e do Decorator podem se beneficiar com frequência do uso do Prototype. Aplicando o padrão permite que você clone estruturas complexas ao invés de reconstruí-las do zero.
- O Prototype não é baseado em heranças, então ele não tem os inconvenientes dela. Por outro lado, o *Prototype* precisa de uma inicialização complicada do objeto clonado. O Factory Method é baseado em herança mas não precisa de uma etapa de inicialização.
- Algumas vezes o Prototype pode ser uma alternativa mais simples a um Memento. Isso funciona se o objeto, o estado no qual você quer armazenar na história, é razoavelmente intuitivo e não tem ligações para recursos externos, ou as ligações são fáceis de se restabelecer.
- As Fábricas Abstratas, Construtores, e Protótipos podem todos ser implementados como Singletons.