refactoring.guru

Memento

16–21 minutes

Também conhecido como: Lembrança, Retrato, Snapshot

Propósito

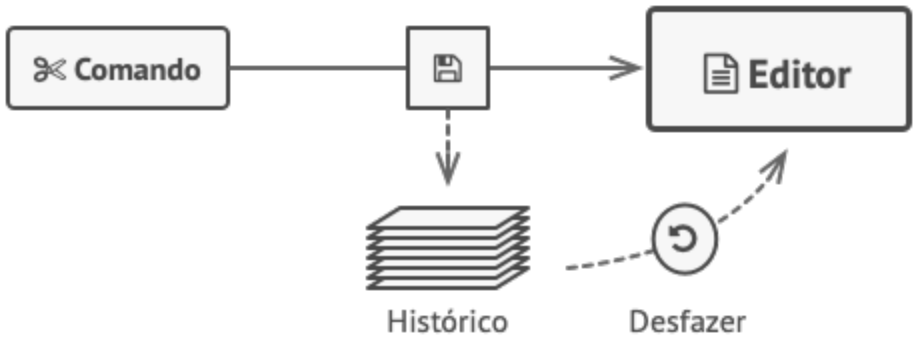
O **Memento** é um padrão de projeto comportamental que permite que você salve e restaure o estado anterior de um objeto sem revelar os detalhes de sua implementação.

Problema

Imagine que você está criando uma aplicação de editor de texto.

Além da simples edição de texto, seu editor pode formatar o texto, inserir imagens em linha, etc.

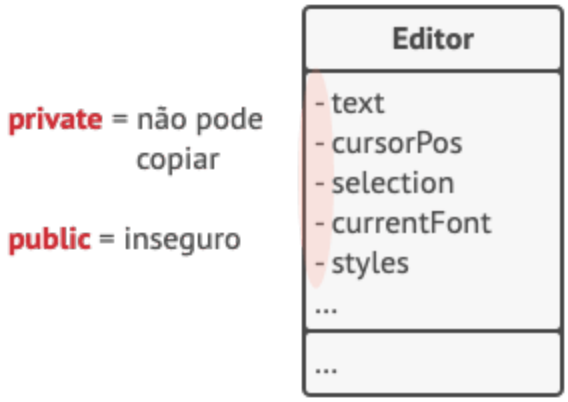
Em determinado momento você decide permitir que os usuários desfaçam quaisquer operações realizadas no texto. Essa funcionalidade tem se tornado tão comum nos últimos anos que, hoje em dia, as pessoas esperam que toda aplicação a tenha. Para a implementação você decide usar a abordagem direta. Antes de realizar qualquer operação, a aplicação grava o estado de todos os objetos e salva eles em algum armazenamento. Mais tarde, quando um usuário decide reverter a ação, a aplicação busca o último retrato do histórico e usa ele para restaurar o estado de todos os objetos.



Antes de executar uma operação, a aplicação salva um retrato do estado dos objetos, que pode mais tarde ser usada para restaurá-los a seu estado anterior.

Vamos pensar sobre esses retratos de estado. Como exatamente você produziria um? Você provavelmente teria que percorrer todos os campos de um objeto e copiar seus valores para o armazenamento. Contudo, isso só funcionaria se o objeto tiver poucas restrições de acesso a seu conteúdo. Infelizmente, a maioria dos objetos reais não deixa os outros espiarem dentro deles assim tão facilmente, escondendo todos os dados significativos em campos privados.

Ignore esse problema por enquanto e vamos assumir que nossos objetos comportam-se como hippies: preferindo relações abertas e mantendo seus estado público. Embora essa abordagem resolveria o problema imediato e permitiria que você produzisse retratos dos estados de seus objetos à vontade, ela ainda tem vários problemas graves. No futuro, você pode decidir refatorar algumas das classes do editor, ou adicionar e remover alguns campos. Parece fácil, mas isso também exigiria mudar as classes responsáveis por copiar o estado dos objetos afetados.



Como fazer uma cópia do estado privado de um objeto?

E tem mais. Vamos considerar os próprios “retratos” do estado do editor. Que dados ele contém? No mínimo dos mínimos ele terá o próprio texto, coordenadas do cursor, posição atual do scroll, etc. Para fazer um retrato você teria que coletar todos esses valores e colocá-los em algum tipo de contêiner.

O mais provável é que você vai armazenar muitos desses objetos contêineres dentro de alguma lista que representaria o histórico. Portanto os contêineres terminariam sendo objetos de uma classe. A classe não teria muitos métodos, mas vários campos que espelhariam o estado do editor. Para permitir que objetos sejam escritos e lidos no e do retrato, você teria que provavelmente tornar seus campos públicos. Isso iria expor todos

os estados do editor, privados ou não. Outras classes se tornariam dependentes de cada pequena mudança na classe do retrato, o que poderia acontecer dentro dos campos privados e métodos sem afetar as classes externas.

Parece que chegamos em um beco sem saída: você ou expõe todos os detalhes internos das classes tornando-as frágeis, ou restringe o acesso ao estado delas, tornando impossível produzir retratos. Existe alguma outra maneira de implementar o "desfazer"?

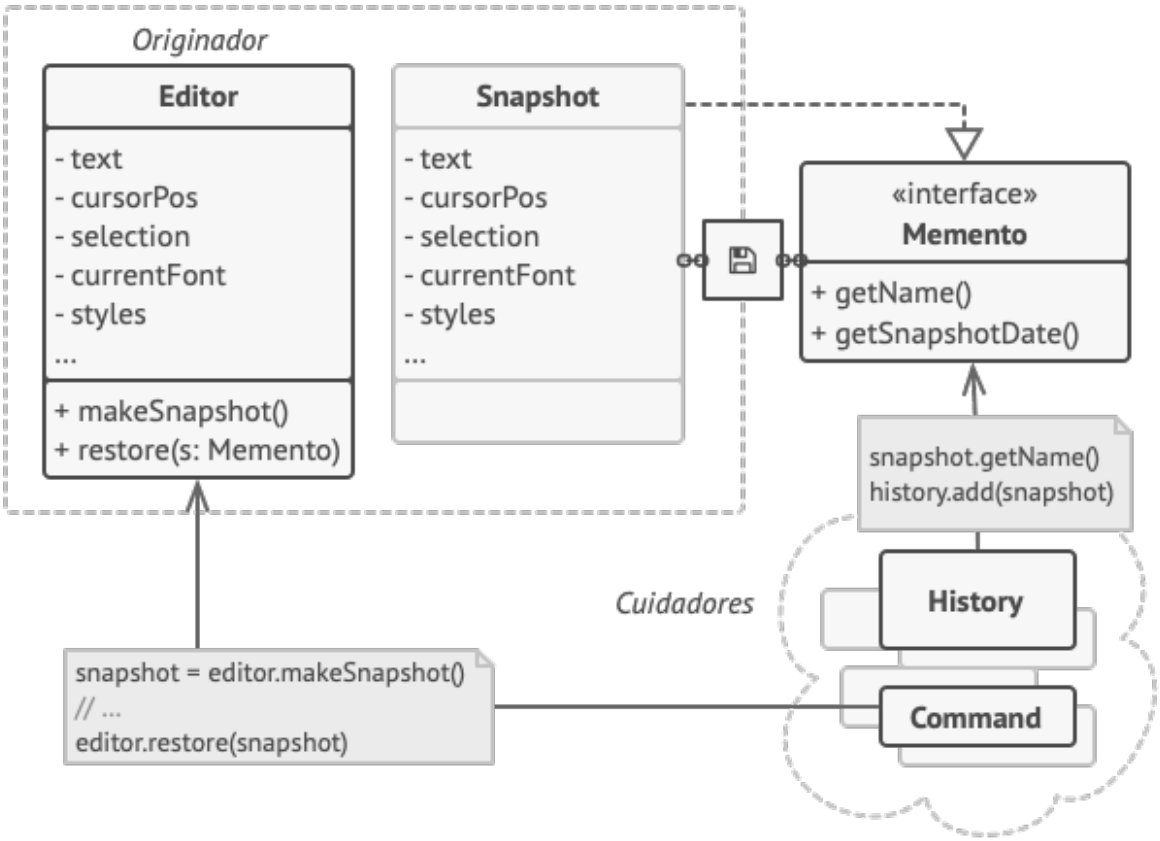
Solução

Todos os problemas que vivenciamos foram causados por um encapsulamento quebrado. Alguns objetos tentaram fazer mais do que podiam. Para coletar os dados necessários para fazer uma ação, eles invadiram o espaço privado de outros objetos ao invés de deixar esses objetos fazer a verdadeira ação.

O padrão Memento delega a criação dos retratos do estado para o próprio dono do estado, o objeto *originador*. Portanto, ao invés de outros objetos tentarem copiar o estado do editor “a partir do lado de fora”, a própria classe do editor pode fazer o retrato já que tem acesso total a seu próprio estado.

O padrão sugere armazenar a cópia do estado de um objeto em um objeto especial chamado *memento*. Os conteúdos de um memento não são acessíveis para qualquer outro objeto exceto aquele que o produziu. Outros objetos podem se comunicar com mementos usando uma interface limitada que pode permitir a recuperação dos metadados do retrato (data de criação, nome a operação efetuada, etc.), mas não ao estado do objeto original

contido no retrato.



O originador tem acesso total ao memento, enquanto que o cuidador pode acessar somente os metadados

Tal regra restritiva permite que você armazene mementos dentro de outros objetos, geralmente chamados de *cuidadores*. Uma vez que o cuidador trabalha com o memento apenas por meio de uma interface limitada, ele não será capaz de mexer com o estado armazenado dentro do memento. Ao mesmo tempo, o originador tem acesso total a todos os campos dentro do memento, permitindo que ele o restaure ao seu estado anterior à vontade.

Em nosso exemplo de editor de texto, nós podemos criar uma classe histórico separada para agir como a cuidadora. Uma pilha de mementos armazenada dentro da cuidadora irá crescer a cada vez que o editor estiver prestes a executar uma operação. Você pode até mesmo renderizar essa pilha dentro do UI da aplicação,

mostrando o histórico de operações realizadas anteriormente para um usuário.

Quando um usuário aciona o desfazer, o histórico pega o memento mais recente da pilha e o passa de volta para o editor, pedindo uma reversão. Já que o editor tem acesso total ao memento, ele muda seu próprio estado com os valores obtidos do memento.

Estrutura

Implementação baseada em classes aninhadas

A implementação clássica do padrão dependente do apoio para classes aninhadas, disponível em muitas linguagens de programação populares (tais como C++, C#, e Java).

1. A classe **Originadora** pode produzir retratos de seu próprio estado, bem como restaurar seu estado de retratos quando necessário.
2. O **Memento** é um objeto de valor que age como um retrato do estado da originadora. É uma prática comum fazer o memento imutável e passar os dados para ele apenas uma vez, através do construtor.
3. A **Cuidadora** sabe não só “quando” e “por quê” capturar o estado da originadora, mas também quando o estado deve ser restaurado.

   Uma cuidadora pode manter registros do histórico da originadora armazenando os mementos em um pilha. Quando a originadora precisa voltar atrás no histórico, a cuidadora busca o memento

mais do topo da pilha e o passa para o método de restauração da originadora.

4. Nessa implementação, a classe memento está aninhada dentro da originadora. Isso permite que a originadora acesse os campos e métodos do memento, mesmo que eles tenham sido declarados privados. Por outro lado, a cuidadora tem um acesso muito limitado aos campos do memento, que permite ela armazenar os mementos em uma pilha, mas não permite mexer com seu estado.

Implementação baseada em uma interface intermediária

Há uma implementação alternativa, adequada para linguagens de programação que não suportam classes aninhadas (sim, PHP, estou falando de você).

1. Na ausência de classes aninhadas, você pode restringir o acesso aos campos do memento ao estabelecer uma convenção para que cuidadoras possam trabalhar com um memento através apenas de uma interface intermediária explicitamente declarada, que só declararia os métodos relacionados aos metadados do memento.
2. Por outro lado, as originadoras podem trabalhar com um objeto memento diretamente, acessando campos e métodos declarados na classe memento. O lado ruim dessa abordagem é que você precisa declarar todos os membros do memento como públicos.

Implementação com um encapsulamento ainda mais estrito

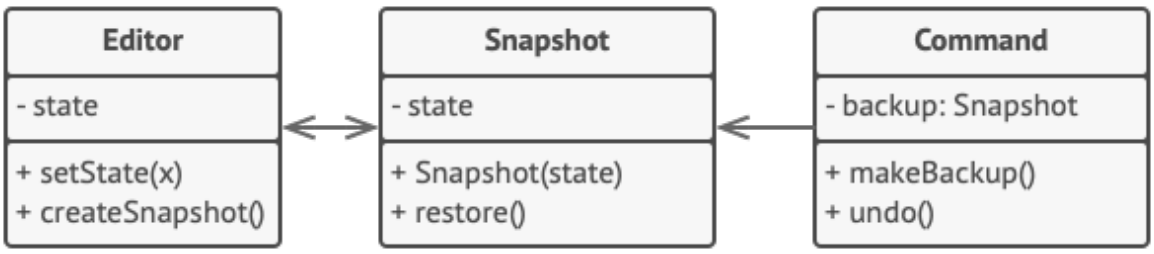
Há ainda outra implementação que é útil quando você não quer

deixar a mínima chance de outras classes acessarem o estado da originadora através do memento.

1. Essa implementação permite ter múltiplos tipos de originadoras e mementos. Cada originadora trabalha com uma classe memento correspondente. Nem as originadoras nem os mementos expõem seu estado para ninguém.
2. Cuidadoras são agora explicitamente restritas de mudar o estado armazenado nos mementos. Além disso, a classe cuidadora se torna independente da originadora porque o método de restauração agora está definido na classe memento.
3. Cada memento se torna ligado à originadora que o produziu. A originadora passa a si mesmo para o construtor do memento, junto com os valores de seu estado. Graças a relação íntima entre essas classes, um memento pode restaurar o estado da sua originadora, desde que esta última tenha definido os setters apropriados.

Pseudocódigo

Este exemplo usa o padrão Memento junto com o padrão Command para armazenar retratos do estado de um editor de texto complexo e restaurá-lo para um estado anterior desses retratos quando necessário



Salvando retratos do estado de um editor de texto.

Os objetos comando agem como cuidadores. Eles buscam o memento do editor antes de executar operações relacionadas aos comandos. Quando um usuário tenta desfazer o comando mais recente, o editor pode usar o memento armazenando naquele comando para reverter a si mesmo para o estado anterior.

A classe memento não declara qualquer campo público, getters, ou setters. Portanto nenhum objeto pode alterar seus conteúdos. Os mementos estão ligados ao objeto do editor que os criou. Isso permite que um memento restaure o estado do editor ligado a ele passando os dados via setters no objeto do editor. Já que mementos são ligados com objetos do editor específicos, você pode fazer sua aplicação suportar diversas janelas independentes do editor com uma pilha centralizada de desfazer.

```
// O originador tem alguns dados importantes que podem mudar
com
// o tempo. Ele também define um método para salvar seu estado
// dentro de um memento e outro método para restaurar o estado
// dele.
class Editor is
    private field text, curX, curY, selectionWidth

    method setText(text) is
        this.text = text

    method setCursor(x, y) is
        this.curX = x
        this.curY = y

    method setSelectionWidth(width) is
```

```
        this.selectionWidth = width

    // Salva o estado atual dentro de um memento.
    method createSnapshot():Snapshot is
        // O memento é um objeto imutável; é por isso que o
        // originador passa seu estado para os parâmetros do
        // construtor do memento.
        return new Snapshot(this, text, curX, curY, selectionWidth)

// A classe memento armazena o estado anterior do editor.
class Snapshot is
    private field editor: Editor
    private field text, curX, curY, selectionWidth

    constructor Snapshot(editor, text, curX, curY, selectionWidth) is
        this.editor = editor
        this.text = text
        this.curX = x
        this.curY = y
        this.selectionWidth = selectionWidth

    // Em algum momento, um estado anterior do editor pode ser
    // restaurado usando um objeto memento.
    method restore() is
        editor.setText(text)
        editor.setCursor(curX, curY)
        editor.setSelectionWidth(selectionWidth)

// Um objeto comando pode agir como cuidador. Neste caso, o
// comando obtém o memento antes que ele mude o estado do
```

```
// originador. Quando o undo(desfazer) é solicitado, ele
// restaura o estado do originador a partir de um memento.
class Command is
    private field backup: Snapshot

    method makeBackup() is
        backup = editor.createSnapshot()

    method undo() is
        if (backup != null)
            backup.restore()
```

Aplicabilidade

Utilize o padrão Memento quando você quer produzir retratos do estado de um objeto para ser capaz de restaurar um estado anterior do objeto.

O padrão Memento permite que você faça cópias completas do estado de um objeto, incluindo campos privados, e armazená-los separadamente do objeto. Embora a maioria das pessoas vão lembrar desse padrão graças ao caso “desfazer”, ele também é indispensável quando se está lidando com transações (isto é, se você precisa reverter uma operação quando se depara com um erro).

Utilize o padrão quando o acesso direto para os campos/getters /setters de um objeto viola seu encapsulamento.

O Memento faz o próprio objeto ser responsável por criar um retrato de seu estado. Nenhum outro objeto pode ler o retrato,

fazendo do estado original do objeto algo seguro e confiável.

Como implementar

1. Determine qual classe vai fazer o papel de originadora. É importante saber se o programa usa um objeto central deste tipo ou múltiplos objetos pequenos.
2. Crie a classe memento. Um por um, declare o conjunto dos campos que espelham os campos declarados dentro da classe originadora.
3. Faça a classe memento ser imutável. Um memento deve aceitar os dados apenas uma vez, através do construtor. A classe não deve ter setters.
4. Se a sua linguagem de programação suporta classes aninhadas, aninhe o memento dentro da originadora. Se não, extraia uma interface em branco da classe memento e faça todos os outros objetos usá-la para se referir ao memento. Você pode adicionar algumas operações de metadados para a interface, mas nada que exponha o estado da originadora.
5. Adicione um método para produção de mementos na classe originadora. A originadora deve passar seu estado para o memento através de um ou múltiplos argumentos do construtor do memento.

   O tipo de retorno do método deve ser o da interface que você extraiu na etapa anterior (assumindo que você extraiu alguma coisa). Por debaixo dos panos, o método de produção de memento deve funcionar diretamente com a classe memento.
6. Adicione um método para restaurar o estado da classe originadora

para sua classe. Ele deve aceitar o objeto memento como um argumento. Se você extraiu uma interface na etapa anterior, faça-a do tipo do parâmetro. Neste caso, você precisa converter o tipo do objeto que está vindo para a classe memento, uma vez que a originadora precisa de acesso total a aquele objeto.

7. A cuidadora, estando ela representando um objeto comando, um histórico, ou algo completamente diferente, deve saber quando pedir novos mementos da originadora, como armazená-los, e quando restaurar a originadora com um memento em particular.
8. O elo entre cuidadoras e originadoras deve ser movido para dentro da classe memento. Neste caso, cada memento deve se conectar com a originadora que criou ele. O método de restauração também deve ser movido para a classe memento. Contudo, isso tudo faria sentido somente se a classe memento estiver aninhada dentro da originadora ou a classe originadora fornece setters suficientes para sobrescrever seu estado.

Prós e contras

- Você pode produzir retratos do estado de um objeto sem violar seu encapsulamento.
- Você pode simplificar o código da originadora permitindo que a cuidadora mantenha o histórico do estado da originadora.
- A aplicação pode consumir muita RAM se os clientes criarem mementos com muita frequência.
- Cuidadoras devem acompanhar o ciclo de vida da originadora para serem capazes de destruir mementos obsoletos.
- A maioria das linguagens de programação dinâmicas, tais como

PHP, Python, e JavaScript, não conseguem garantir que o estado dentro do memento permaneça intacto.

Relações com outros padrões

- Você pode usar o Command e o Memento juntos quando implementando um “desfazer”. Neste caso, os comandos são responsáveis pela realização de várias operações sobre um objeto alvo, enquanto que os mementos salvam o estado daquele objeto momentos antes de um comando ser executado.
- Você pode usar o Memento junto com o Iterator para capturar o estado de iteração atual e revertê-lo se necessário.
- 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.