Padrões de Projeto

Memento

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Rio Grande

Agenda

Introdução

- O padrão de projeto Memento é um padrão comportamental que permite capturar e armazenar o estado interno de um objeto em um determinado momento, sem violar o encapsulamento.
- Isso significa que você pode restaurar o objeto para um estado anterior, se necessário

- O Memento é um padrão de projeto comportamental que permite tirar um retrato do estado de um objeto e restaurá-lo no futuro.
- O Memento não compromete a estrutura interna do objeto com o qual trabalha, nem os dados mantidos dentro dos retratos.

Aplicações

- **Editores de texto**: Para desfazer ações.
- **Jogos:** Para salvar o progresso do jogador.
- **Sistemas de backup:** Para restaurar dados a um ponto anterior.

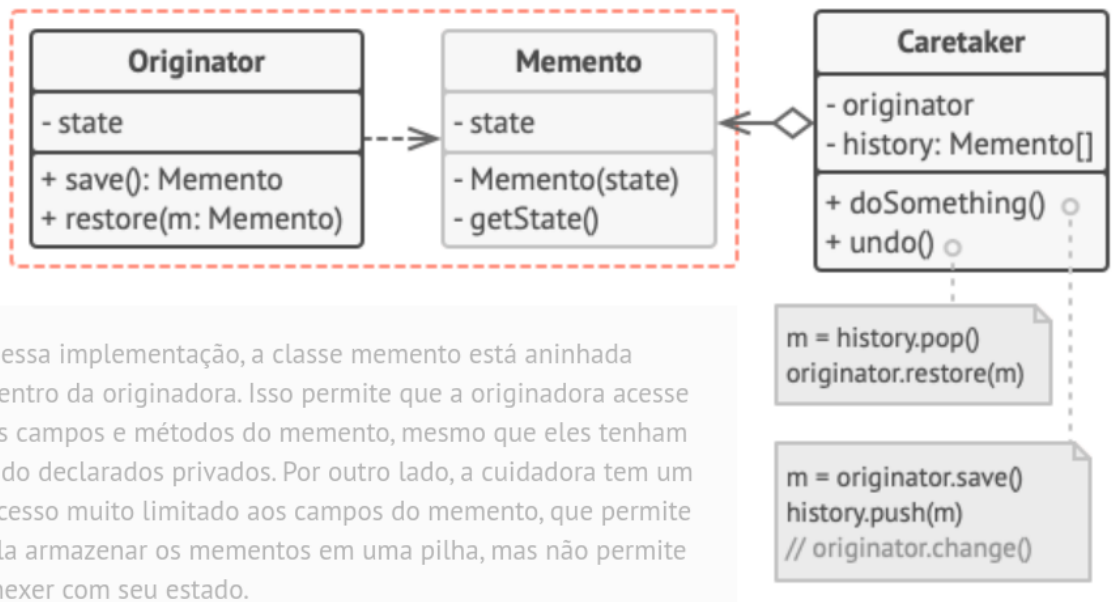
Esse padrão é especialmente útil em situações onde é necessário desfaz operações ou restaurar estados anteriores sem expor detalhes internos do objeto.

Como Funciona

Implementação baseada em classes aninhadas

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.

Como Funciona

1. A classe **Originator** 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.

Como Funciona

3. A Cuidadora sabe não só **quando** e **por quê** capturar o estado da originadora, mas também quando o estado deve ser restaurado.

- Uma **Caretaker** pode manter registros do histórico da originadora armazenando os mementos em um pilha.
- Quando a **Originator** precisa voltar atrás no histórico, a **Caretaker** busca o **Memento** mais do topo da pilha e o passa para o método de restauração da originadora.

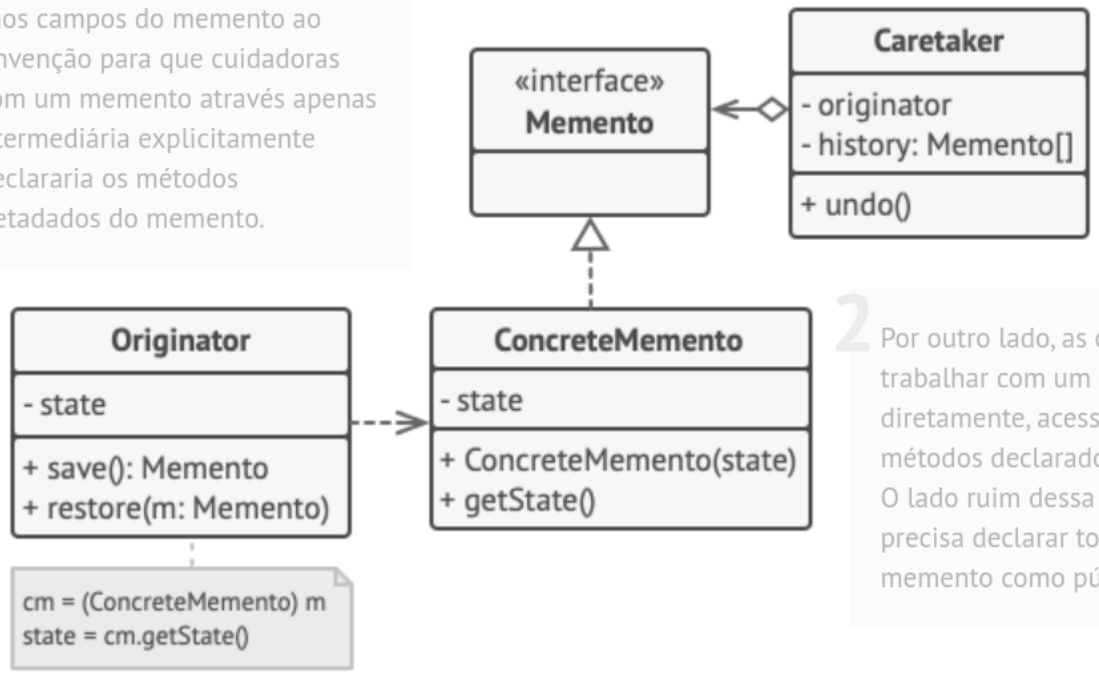
Como Funciona

- Nessa implementação, a classe **Memento** está aninhada dentro da **Originator**.
- Isso permite que a **Originator** acesse os campos e métodos do **Memento**, mesmo que eles tenham sido declarados privados.
- Por outro lado, a **Caretaker** 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.

Como funciona

Implementação baseada em uma interface intermediária

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.

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 (rollback)
    - isto é, se você precisa reverter uma operação quando se depara com um erro.

Aplicabilidade

- 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 **Originator**.
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.

Como implementar

4. Se a sua linguagem de programação suporta classes aninhadas, aninhe o memento dentro da **Originator**.
   - 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 **Originator**.
5. Adicione um método para produção de mementos na classe **Originator**.
   - A **Originator** deve passar seu estado para o **Memento** através de um ou múltiplos argumentos do construtor do memento.

Como Implementar

- 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.

Como implementar

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.

Como implementar

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.

Exemplo

```
1  // Originador
2  public class TextEditor {
3      private StringBuilder content = new StringBuilder();
4
5      public void write(String text) {
6          content.append(text);
7      }
8      public String getContent() {
9          return content.toString();
10     }
11     public TextEditorMemento save() {
12         return new TextEditorMemento(content.toString());
13     }
14     public void restore(TextEditorMemento memento) {
15         content = new StringBuilder(memento.getState());
16     }
17 }
```

Exemplo

```
1   // Memento
2   public class TextEditorMemento {
3       private final String state;
4
5       public TextEditorMemento(String state) {
6           this.state = state;
7       }
8
9       public String getState() {
10          return state;
11      }
12  }
```

Exemplo

```
1  // Caretaker
2  public class TextEditorCaretaker {
3      private final Stack<TextEditorMemento> history = new Stack<>();
4
5      public void save(TextEditor editor) {
6          history.push(editor.save());
7      }
8
9      public void undo(TextEditor editor) {
10         if (!history.isEmpty()) {
11             editor.restore(history.pop());
12         }
13     }
14 }
```

Exemplo

```java
public class Main {
    public static void main(String[] args) {
        TextEditor editor = new TextEditor();
        TextEditorCaretaker caretaker = new TextEditorCaretaker();
        editor.write("Olá, ");
        caretaker.save(editor);
        System.out.println(editor.getContent()); // Saída: Olá,
        editor.write("mundo!");
        caretaker.save(editor);
        System.out.println(editor.getContent()); // Saída: Olá, mundo!
        caretaker.undo(editor);
        System.out.println(editor.getContent()); // Saída: Olá,
        caretaker.undo(editor);
        System.out.println(editor.getContent()); // Saída: (vazio)
    }
}
```

Vantagens

Preservação do Encapsulamento:

- O Memento permite salvar e restaurar o estado de um objeto sem expor os detalhes internos desse objeto.
  - Isso mantém o encapsulamento intacto e protege a integridade dos dados

.

Facilidade de Implementação de Desfazer/Refazer:

- É ideal para implementar funcionalidades de desfazer (undo) e refazer (redo) em aplicações, como editores de texto e gráficos, onde os usuários frequentemente precisam reverter ações.

Vantagens

Simplicidade na Recuperação de Estados:

- Permite a recuperação fácil e eficiente de estados anteriores, o que é útil em sistemas de backup e recuperação de dados.

Flexibilidade:

- Pode ser usado em uma variedade de contextos, desde jogos que precisam salvar o progresso do jogador até aplicações empresariais que necessitam de auditoria e rastreamento de mudanças.

Histórico de Estados:

- Facilita a manutenção de um histórico de estados, permitindo que o sistema volte a qualquer ponto anterior no tempo, o que é útil para depuração e análise.

Desvantagens

Consumo de Memória:

- Armazenar muitos Mementos pode consumir uma quantidade significativa de memória, especialmente se os estados dos objetos forem grandes ou se os Mementos forem criados com muita frequência.

Complexidade Adicional:

- A implementação do padrão Memento pode adicionar complexidade ao código, especialmente na gestão do ciclo de vida dos Mementos e na garantia de que estados obsoletos sejam descartados corretamente.
  - Como ter certeza do estado atual do objeto em ambientes multithread?

Desvantagens

Desempenho:

- A criação e restauração de Mementos pode impactar o desempenho da aplicação, especialmente em sistemas onde a performance é crítica.

Desvantagens

Manutenção:

- Manter o código que utiliza o padrão Memento pode ser desafiador, especialmente em sistemas grandes e complexos, onde é necessário garantir que todos os estados relevantes sejam corretamente capturados e restaurados.

Compatibilidade com Linguagens Dinâmicas:

- Em linguagens de programação dinâmicas, como Python e JavaScript, pode ser difícil garantir que o estado dentro do Memento permaneça intacto, o que pode levar a problemas de integridade dos dados.

Memento vs Command

A diferença principal entre os padrões de projeto Memento e Command está no que cada um encapsula e como são utilizados:

Memento

- **Encapsula o estado interno de um objeto.**
- Permite salvar e restaurar o estado de um objeto sem violar o encapsulamento.

Command

- **Encapsula uma solicitação como um objeto.**
- Permite parametrizar métodos com diferentes solicitações, enfileirar ou registrar solicitações, e suportar operações de desfazer.

Memento vs Command

Diferenças Chave

- **Memento foca em salvar e restaurar estados.**
- **Command foca em executar ações e pode incluir funcionalidades de desfazer, mas através da reexecução de comandos.**

Memento Vs State

- **Memento**
  - Capturar e restaurar o estado interno de um objeto sem violar o encapsulamento.
- **State**
  - Permitir que um objeto altere seu comportamento quando seu estado interno muda, parecendo mudar sua classe.

Diferenças Chave

- Memento **foca em salvar e restaurar estados sem expor detalhes internos.**
- State **foca em alterar o comportamento de um objeto com base em seu estado interno atual**

Padrões de Projeto

Memento

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Rio Grande