TÓPICO 19 - MEMENTO

Design Patterns - Professor Ramon Venson - SATC 2026.1

Motivação

Muitos sistemas precisam voltar a um estado anterior.

O desafio é fazer isso sem expor os detalhes internos do objeto que teve seu estado alterado.

Problema

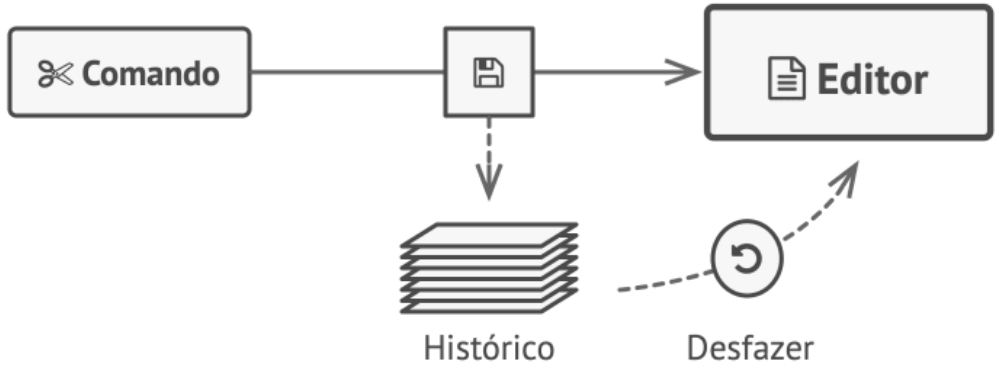
private = não pode copiar

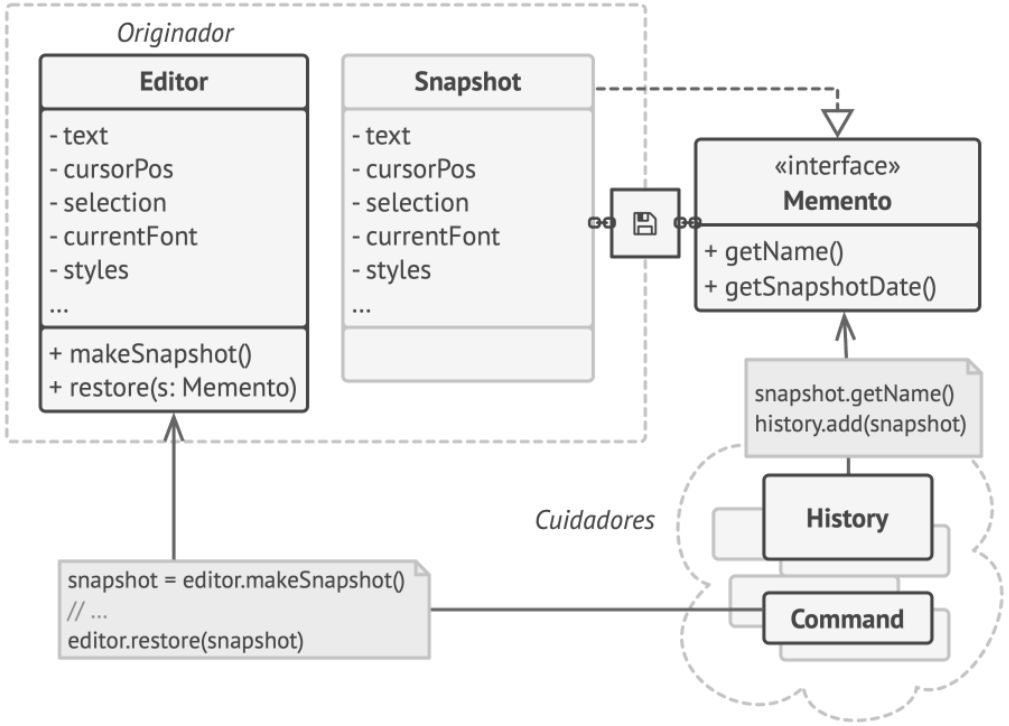
public = inseguro

- Há necessidade de `undo`, histórico ou rollback
- O estado relevante está protegido dentro do objeto
- Copiar o estado de fora quebra encapsulamento
- Classes externas passam a depender da estrutura interna
- Refatorações internas se espalham para o restante do sistema

Conceito principal

- **Memento** é um padrão comportamental
- O próprio objeto cria um retrato do seu estado
- Esse retrato pode ser guardado para uso futuro
- Outro objeto administra o histórico
- A restauração ocorre sem revelar detalhes internos





Papéis do padrão

- **Originador**: cria e restaura mementos
- **Memento**: snapshot do estado em um momento específico
- **Cuidador**: armazena o histórico e decide quando usá-lo
- O cuidador não deveria manipular o conteúdo interno do snapshot
- O foco é preservar encapsulamento com recuperação de estado

Como funciona

- Antes de uma mudança relevante, o cuidador solicita um snapshot
- O originador monta o memento com acesso ao próprio estado
- O snapshot é armazenado em pilha, lista ou histórico
- Quando necessário, o cuidador devolve o memento ao originador
- O originador restaura a versão anterior com segurança

Código: originador e memento

```
class EditorTexto {
    private String conteudo;
    private int cursor;

    public Memento criarSnapshot() {
        return new Memento(conteudo, cursor);
    }

    public void restaurar(Memento memento) {
        this.conteudo = memento.conteudo;
        this.cursor = memento.cursor;
    }
}
```

Código: cuidador

```
class HistoricoEditor {
    private final Stack<EditorTexto.Memento> pilha = new Stack<>();

    public void salvar(EditorTexto editor) {
        pilha.push(editor.criarSnapshot());
    }

    public void desfazer(EditorTexto editor) {
        if (!pilha.isEmpty()) {
            editor.restaurar(pilha.pop());
        }
    }
}
```

Quando usar

- Há necessidade real de restaurar estados anteriores
- O sistema oferece `undo`, `redo` ou rollback
- O estado interno não deve ser exposto
- O histórico faz parte importante da experiência do usuário
- O custo de armazenar snapshots é aceitável

Quando não usar

- O estado pode ser reconstruído facilmente
- Há poucas alterações e nenhum histórico necessário
- O snapshot seria grande e caro demais
- A equipe quer aplicar o padrão só por catálogo
- Uma solução mais simples resolve o problema com menos custo

Vantagens

- Preserva o encapsulamento
- Facilita `undo` e rollback
- Separa histórico da lógica principal do objeto
- Deixa explícita a noção de snapshot
- Funciona bem com fluxos transacionais e editores

Desvantagens

- Pode consumir muita memória
- Adiciona indireção e mais classes
- Exige política de descarte do histórico
- Snapshots completos podem ser caros
- Mau uso pode gerar otimização e complexidade desnecessárias

Relações com outros padrões

- **Command**: muito comum em operações com desfazer
- **Prototype**: clona objetos; Memento registra estado para restauração
- **Iterator**: pode registrar a posição atual da navegação
- **Encapsulamento**: é a principal preocupação do padrão
- **Imutabilidade**: ajuda a manter snapshots confiáveis

Resumo final

- Memento salva retratos de estado sem expor detalhes internos
- O originador cria e restaura o snapshot
- O cuidador organiza o histórico
- O padrão é útil em `undo`, rollback e checkpoints
- Use com critério para evitar custo excessivo de memória