Skip to navigation

Marcos Brizeno

Desenvolvimento de Software #showmethecode

Memento

novembro 5, 2011

Mão na massa: Memento

Problema:

Qualquer bom editor, seja de video, texto, imagens, etc. oferece uma maneira de desfazer ações, recuperando estados anteriores. Como é possível modelar a arquitetura do sistema de maneira que seja possível salvar estados dos elementos?

Para exemplificar vamos pensar em editor de texto que precisa manter o controle apenas do texto que é digitado, um notepad por exemplo. Uma primeira saída poderia ser criar um objeto que representasse o texto com suas informações e guardar estes objetos em uma lista.

O problema desta implementação está em como recuperar os estados sem ferir o encapsulamento da classe? Pois, ao restaurar o objeto precisaríamos de, no mínimo, getters para acessar as informações do objeto e poder copiá-las para o objeto a ser restaurado.

O padrão Memento oferece uma maneira simples de evitar o problema da quebra de encapsulamento e manter o controle das alterações feitas em um objeto. Vejamos então o padrão:

Memento

Intenção:

“Sem violar o encapsulamento, capturar e externalizar um estado interno de um objeto, de maneira que o objeto possa ser restaurado para esse estado mais tarde.” [1]

Pela intenção do padrão podemos facilmente notar sua aplicabilidade. O estado interno do objeto seria, para o exemplo acima, o texto que está sendo digitado pelo usuário. Assim, o padrão Memento permitiria capturar o estado do texto par que depois ele possa ser reutilizado.

Vamos então ao código do problema. Vamos iniciar com a classe Memento. Ela simplesmente mantém a String que representa o texto e oferece um getter para esta String, permitindo que ela seja recuperada mais tarde.

```
1  public class TextoMemento {
2      protected String estadoTexto;
3
4      public TextoMemento(String texto) {
5          estadoTexto = texto;
6      }
7
8      public String getTextoSalvo() {
9          return estadoTexto;
10     }
11 }
```

Além do Memento, existe outra figura importante, o Caretaker. O Caretaker vai guardar todos os Memento, permitindo que eles sejam restaurados. Como a aplicação é de um editor de texto os Memento devem ser recuperados de maneira LIFO, *Last in First out*, assim o último memento adicionar será o primeiro a ser recuperado. Vejamos o código a seguir:

```
1  public class TextoCareTaker {
2      protected ArrayList<TextoMemento> estados;
3
4      public TextoCareTaker() {
5          estados = new ArrayList<TextoMemento>();
6      }
7
8      public void adicionarMemento(TextoMemento memento) {
9          estados.add(memento);
10     }
11
12     public TextoMemento getUltimoEstadoSalvo() {
13         if (estados.size() <= 0) {
14             return new TextoMemento("");
15         }
16         TextoMemento estadoSalvo = estados.get(estados.size() - 1);
17         estados.remove(estados.size() - 1);
18         return estadoSalvo;
19     }
20 }
```

A partir do Caretaker é possível armazenar e recuperar um estado. No método que retorna o último estado salvo é necessário fazer uma verificação se existe algum estado a ser retornado, caso contrário é retornado um memento vazio.

Uma pequena observação: retornar nulo exigiria um verificação que poderia facilmente ser esquecida, causando vários problemas na execução do programa. O ideal seria disparar uma exceção, mas como estamos apenas exemplificando, retornar um objeto que não tenha informações é mais simples. Lembre-se: se você precisa retornar nulo, é melhor repensar no seu método.

Agora vamos analisar a classe que representa o Texto:

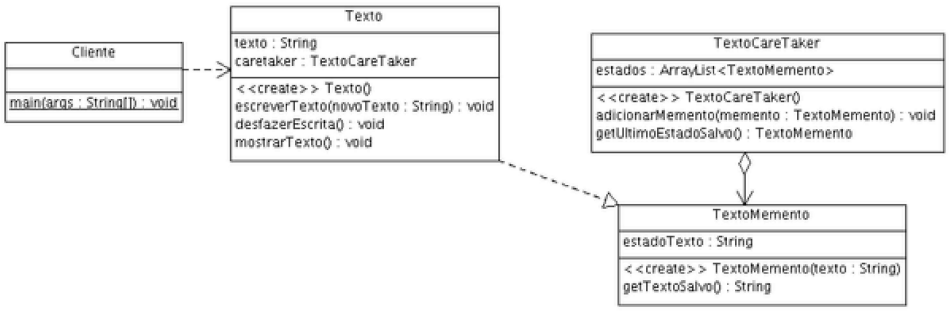
```
1  public class Texto {
2      protected String texto;
3      TextoCareTaker caretaker;
4
5      public Texto() {
6          caretaker = new TextoCareTaker();
7          texto = new String();
8      }
9
10     public void escreverTexto(String novoTexto) {
11         caretaker.adicionarMemento(new TextoMemento(texto));
12         texto += novoTexto;
13     }
14
15     public void desfazerEscrita() {
16         texto = caretaker.getUltimoEstadoSalvo().getTextoSalvo();
17     }
18
19     public void mostrarTexto() {
20         System.out.println(texto);
21     }
22 }
```

A classe texto possui uma interface que permite escrever um texto, desfazer a operação de escrita e exibir o texto no terminal. Ao escrever um novo texto, primeiro o estado é salvo, então a alteração é feita. Ao desfazer a escrita é solicitado ao Caretaker que pegue o último estado salvo, a partir deste estado é possível pegar o texto e restaurá-lo.

A utilização do padrão seria algo do tipo:

```
1  public static void main(String[] args) {
2      Texto texto = new Texto();
3      texto.escreverTexto("Primeira linha do texto\n");
4      texto.escreverTexto("Segunda linha do texto\n");
5      texto.escreverTexto("Terceira linha do texto\n");
6      texto.mostrarTexto();
7      texto.desfazerEscrita();
8      texto.mostrarTexto();
9      texto.desfazerEscrita();
10     texto.mostrarTexto();
11     texto.desfazerEscrita();
12     texto.mostrarTexto();
13     texto.desfazerEscrita();
14     texto.mostrarTexto();
15 }
```

O diagrama UML para este exemplo seria:



Um pouco de teoria

O padrão Memento oferece uma maneira simples de salvar estados internos de um objeto. Basta salvar todas as informações necessárias em Memento e mais tarde recuperá-las. Ele transfere a responsabilidade de fornecer maneiras de acessar o estado para o objeto Memento, deixando o Originator (no nosso exemplo, a classe Texto) livre destas preocupações.

Um desvantagem fácil de ser notada é que armazenar a lista de Memento pode ser caro, computacionalmente. Assim, caso o estado seja muito complexo, pode-se utilizar uma classe intermediária que armazena o estado para simplificar a arquitetura, mas não a complexidade. Em muitos editores é comum notar que é possível configurar a quantidade de espaço a ser utilizado para salvar estados do programa. Assim, ao salvar um Memento, o Caretaker pode verificar se o limite foi atingido e eliminar os Memento mais antigos, caso seja necessário.

Outra desvantagem é que é necessário tomar cuidado para que não seja possível ter acesso ao objeto Memento, pois nada impede que apenas o Caretaker, ou o Originator acessem o estado do Memento. Em C++ é possível utilizar o operador **friend** para que os campos privados sejam visíveis somente em algumas classes.

Código fonte completo

O código completo pode ser baixado no seguinte repositório Git: https://github.com/MarcosX/Padroes-de-Projeto.

Os arquivos estão como um projeto do eclipse, então basta colocar no seu Workspace e fazer o import.

Se gostou do post compartilhe com seus amigos e colegas, senão, comente o que pode ser melhorado. Encontrou algum erro no código? Comente também. Possui alguma outra opinião ou alguma informação adicional? Comenta ai! 😀

Referências:

[1] GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.

 Memento, Padrões de Projeto Java, Memento, Padrões, Projeto 4 Comentários

__PRESENT