

Padrões e Desenho de Software

Professor:
José Luis Oliveira
Sérgio Matos

Padrões de desenho de Software:

Composite & Command

Hugo Paiva, 93195



DETI
Universidade de Aveiro
05-06-2020

Conteúdo

1 Composite	2
1.1 Descrição	2
1.2 Problema	2
1.3 Solução	2
2 Command	3
2.1 Descrição	3
2.2 Problema	3
2.3 Solução	3
3 Bibliografia	4

1 Composite

1.1 Descrição

Composite pertence aos padrões de desenho de software estrutural, dando, portanto, conselhos relacionados com a composição de classes e/ou objetos.

Todos os problemas que podem ser modelados em estruturas de árvore são potenciais candidatos para este padrão de software no entanto, *Composite* resolve especialmente os problemas em que um objeto composto é manipulado de forma diferente que um objeto singular, evitando verificações ou qualquer outro tipo de mecanismo para selecionar diferentes implementações entre estes dois tipos de objetos. Em suma, independentemente do tipo de objeto, usando este padrão, deve-se poder tratá-lo de igual forma.

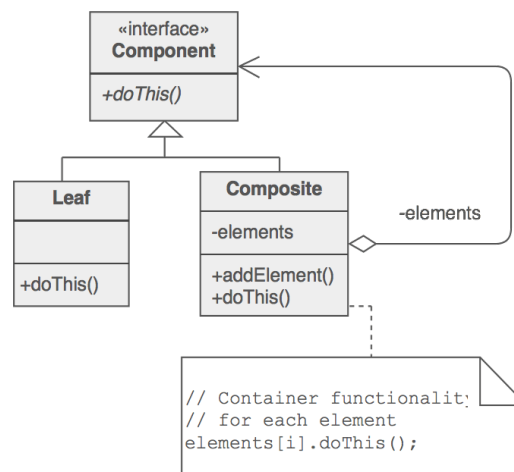


Figura 1: Estrutura usual do padrão *Composite* em um diagrama de classes

Como é possível observar a partir da Figura 1, independentemente do tipo de objeto, ambos implementam o mesmo método, permitindo, no entanto, a adição de outros objetos no objeto composto. De notar que, apesar do método ter o mesmo nome, apresentando transparência e facilidade ao cliente, dependendo do tipo do objeto, a implementação é diferente, sendo usual percorrer o filhos e chamar o mesmo método no objeto composto, como é possível observar na Figura.

Existem, no entanto, outros tipo de implementações ao utilizar este padrão com a possibilidade de permitir a remoção de objetos contidos no objeto composto ou, por exemplo, permitir apenas a substituição dos objetos guardados, por um novo, ao invés de adicionar, com o benefício da melhor compreensão, mas reduzindo a flexibilidade. Dito isto, todas estas pequenas modificações sugeridas ao exemplo da Figura 1 continuam a manter a base do padrão *Composite*, sendo totalmente válidas.

1.2 Problema

1.3 Solução

2 Command

2.1 Descrição

É expectável que um padrão de desenho de software comportamental, para além de fornecer boas práticas para resolver um determinado problema, foque as mesmas na forma como as classes e objetos interagem e distribuem responsabilidades. *Command*, sendo um destes tipos de padrões, foca-se exactamente nesse ponto.

O problema que este padrão resolve incide não só na repetição de código mas também na organização geral das classes e das responsabilidades. Seguindo o *Single Responsibility Principle* e *Open/Closed Principle*, este padrão separa a entidade que invoca uma ação, da entidade que a executa, colocando uma entidade intermédia, o *Command*, que é responsável por chamar o método que desempenha a ação na entidade receptora, tal como o primeiro princípio citado o dita. Ao fazer isto, são evitados os problemas de repetição de código, pois a entidade intermédia pode ser reutilizada múltiplas vezes, bem como problemas relacionados com limitações de comandos ou de alterações às outras entidades, de acordo com o segundo princípio citado.

No fundo, está-se a encapsular um pedido num objeto, criando a possibilidade de adicionar mais métodos ao objeto criado, complementares à simples ação pedida pelo cliente. A aproximação mais usual é a adição de um método que permite reverter o comando no entanto, o céu é o limite, podemos criar métodos que permitem adicionar um simples atraso à operação, até à execução de múltiplos comandos com apenas um pedido.

EXEMPLOS

Considerando, por exemplo, um comando de televisão, este executa múltiplas ações sobre uma televisão, repetindo as ações ao longo da sua utilização. Aproximando este exemplo a uma linguagem de programação, não faria sentido, sempre que um determinado botão fosse clicado, criar um novo objeto com um determinado método ou até mesmo executar um código repetidamente, que apenas funcionaria para uma televisão em específico. Seguindo a lógica do *Command*, faz muito mais sentido instanciar uma classe que armazena a televisão receptora do pedido e que, daí em diante, apenas necessita que lhe chamem o seu método específico para executar a ação, reutilizando e uniformizando o código.

Um dos exemplo mais banais desta situação é toda a gestão que um programa ou sistema operativo faz dos *inputs* de um utilizador. Ao utilizar, por exemplo, qualquer tipo de editor de texto, o sistema operativo permite ao utilizador inserir caracteres num campo mas, permite, também, reverter essas inserções (com a ajuda do atalho *ctrl+z*). Ora, isto é um exemplo visível onde faria sentido aplicar este padrão de desenho de software.

Pessoalmente, sendo eu um utilizador casual de algum software de edição de imagens e vídeo (*Adobe Photoshop*, *Adobe Illustrator* e *Final Cut Pro*), ao associar o *Command* à gestão que estes programas fazem das ações dos utilizadores, os conceitos análogos ao mesmo tornaram-se ainda mais claros. Pegando no exemplo do *Adobe Photoshop*, todas as alterações que são feitas a uma imagem podem ser desfeitas múltiplas vezes, tanto pela mesma ordem em que foram executadas, como também voltar a refazê-las, novamente, na ordem inversa. Ora, aproximando novamente a uma linguagem de programação, estas ações podem muito bem ser executadas com recurso ao padrão *Command* ao encapsular os pedidos do utilizador em objetos. Estes objetos podem também ser introduzidos numa lista onde constam todos os pedidos recentes. Ao utilizar os famosos atalhos para desfazer e refazer (*ctrl+z* e *ctrl+shift+z*), o programa apenas teria de utilizar o método para desfazer a ação e percorrer a lista na ordem inversa. O mesmo se aplica para refazer as ações desfeitas, sendo apenas necessário utilizar o método para executar a ação e percorrer novamente a lista já na ordem "normal".

Como em todos os padrões de desenho de software, a implementação mais correta é sempre discutível pois, apesar de uma solução se desviar da solução-base, esta pode ser justificada de acordo com o problema.

2.2 Problema

2.3 Solução

3 Bibliografia

Composite

- [1] <https://www.geeksforgeeks.org/composite-design-pattern/>
- [2] <https://refactoring.guru/design-patterns/composite>
- [3] https://sourcemaking.com/design_patterns/composite
- [4] https://www.tutorialspoint.com/design_pattern/composite_pattern.htm
- [5] <https://www.baeldung.com/java-composite-pattern>

Command

- [1] https://sourcemaking.com/design_patterns/command
- [2] <https://refactoring.guru/design-patterns/command>
- [3] https://www.tutorialspoint.com/design_pattern/command_pattern.htm
- [4] <https://www.geeksforgeeks.org/command-pattern/>
- [5] <https://medium.com/better-programming/the-command-design-pattern-2313909122b5>
- [6] <https://www.nku.edu/~foxr/CSC360/Programs/MazeWithGraphics.java>