CENTRO UNIVERSITÁRIO RITTER DOS REIS

Padrões Estruturais e Comportamentais

Professor Jean Paul P. Lopes,
PRÁTICAS DE ENGENHARIA DE SOFTWARE - Noite

Eduardo Eli - 2019110719
Fábio Santos -2019121807
Gabriel Zomer - 201010727
Pedro Bueno - 201918771
Ronaldo Teles - 201812318

SUMÁRIO

1.0 Estruturais **3**
1.1 Adapter 3
1.2 Bridge 4
1.3 Composite 5
1.4 Decorator 6
1.5 Facade 7
1.6 FlyWeight 8
1.7 Proxy 9

2.0 Comportamentais **10**
2.1 Chain of responsibility 10
2.2 Command 10
2.3 Iterator 11
2.4 Mediator 12
2.5 Memento 13

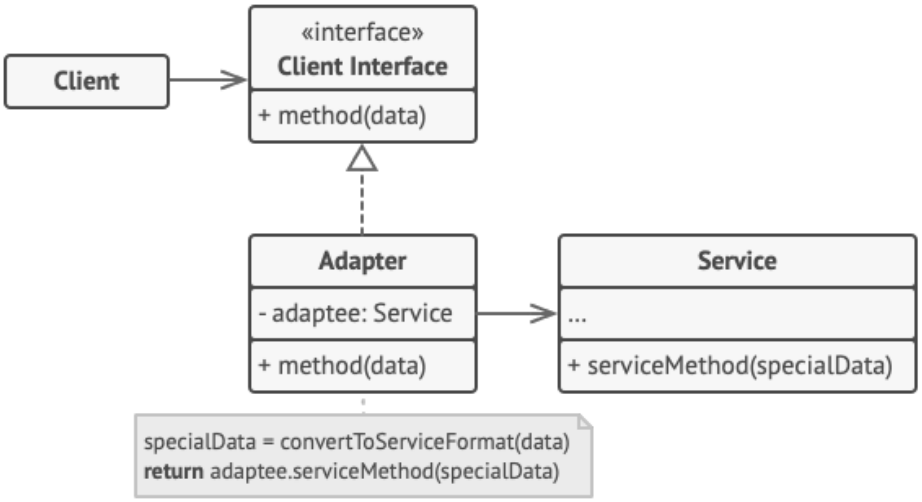
3.0 Criação **14**
3.1 Abstract Factory 14
3.2 Builder 14
3.3 Prototype 15
3.4 Singleton 16

1.0 Estruturais

1.1 Adapter

Tem como função converter a interface de uma classe por outra esperada pelos clientes, adaptando a interface de uma classe para possibilitar que classes incompatíveis trabalhem em conjunto, no qual de outra maneira não seria impossível. Basicamente é utilizada quando se deseja utilizar uma classe existente, mas sua interface não é a mesma que se necessita.

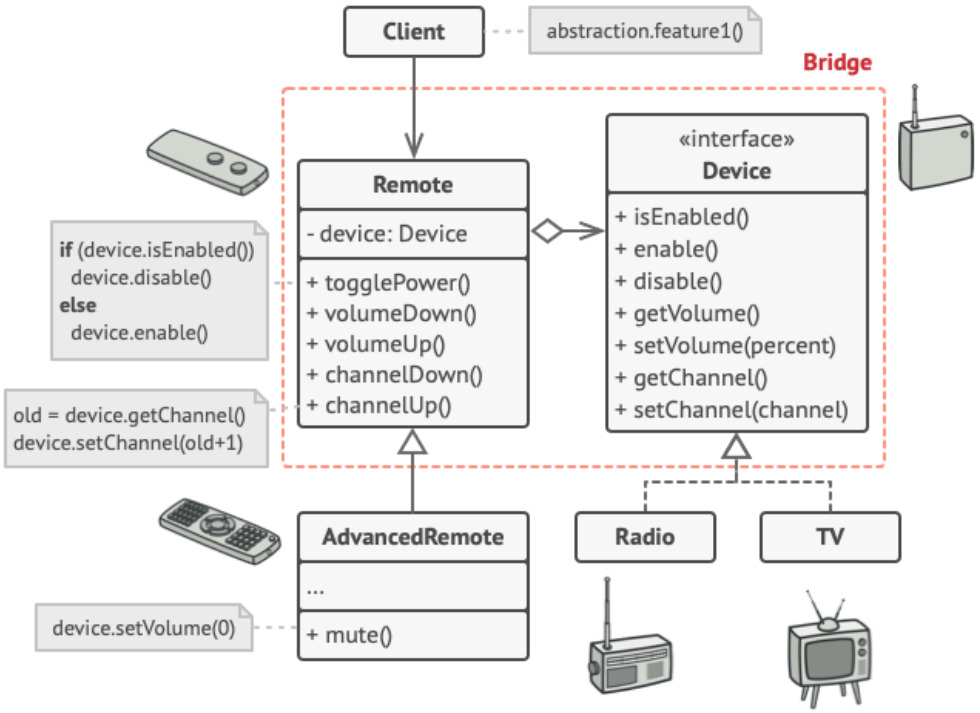
Por exemplo, no código abaixo, onde desejamos inserir pinos quadrados a buracos redondos, temos de adaptá-los.



1.2 Bridge

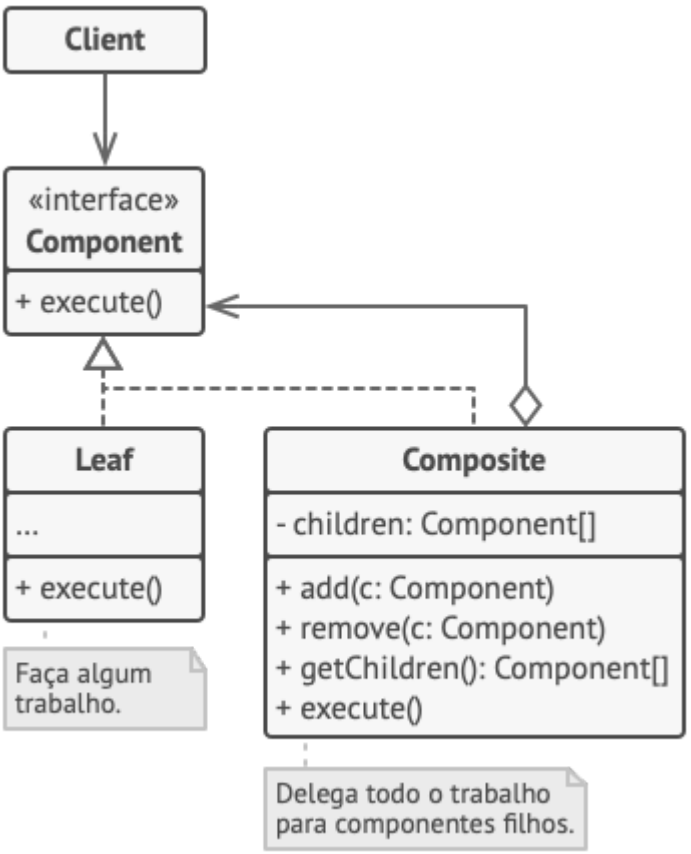
Neste caso, a função bridge é um padrão estrutural no qual conseguimos dividir uma classe grande e ligá-las em hierarquias separadas, no qual podem ser desenvolvidas separadamente. Utiliza-se quando você quer dividir e organizar uma classe que tem várias variantes da mesma funcionalidade.

No exemplo abaixo estamos criando uma interface comum para dispositivos distintos, por exemplo um rádio e uma TV.



1.3 Composite

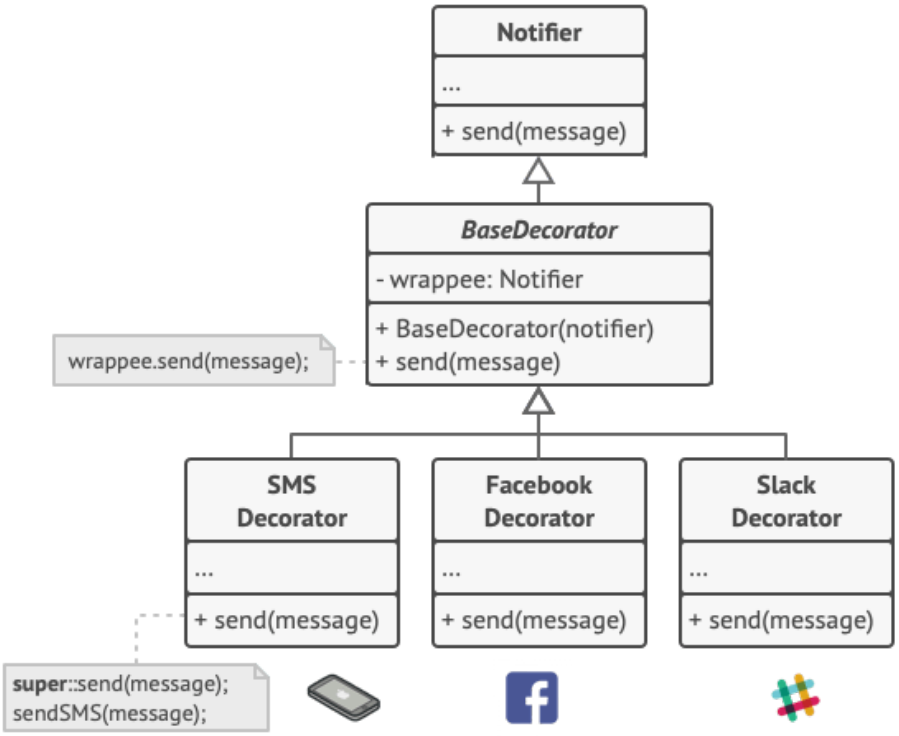
Padrão no qual permite compor objetos em uma estrutura parecida a uma árvore e trabalha com eles como se fosse um objeto individual, usar o composite somente faz sentido quando o modelo central de sua aplicação pode ser representado como uma árvore.



1.4 Decorator

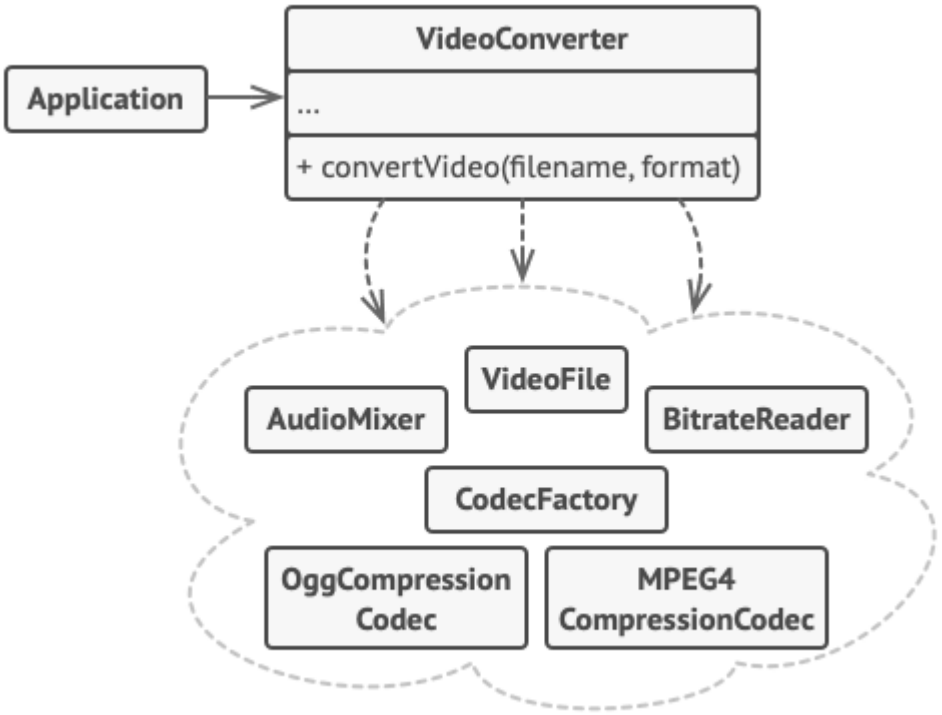
Com padrão estrutural decorator podemos permitir novos comportamentos aos objetos dinamicamente, colocando-os dentro de objetos wrapper especiais.Bastante utilizado em códigos Java, especialmente quando relacionados a fluxos.

Com o exemplo abaixo deixaremos o simples comportamento de notificação por email dentro de uma classe notificador base, e transformar todos os métodos de notificação em decorator.



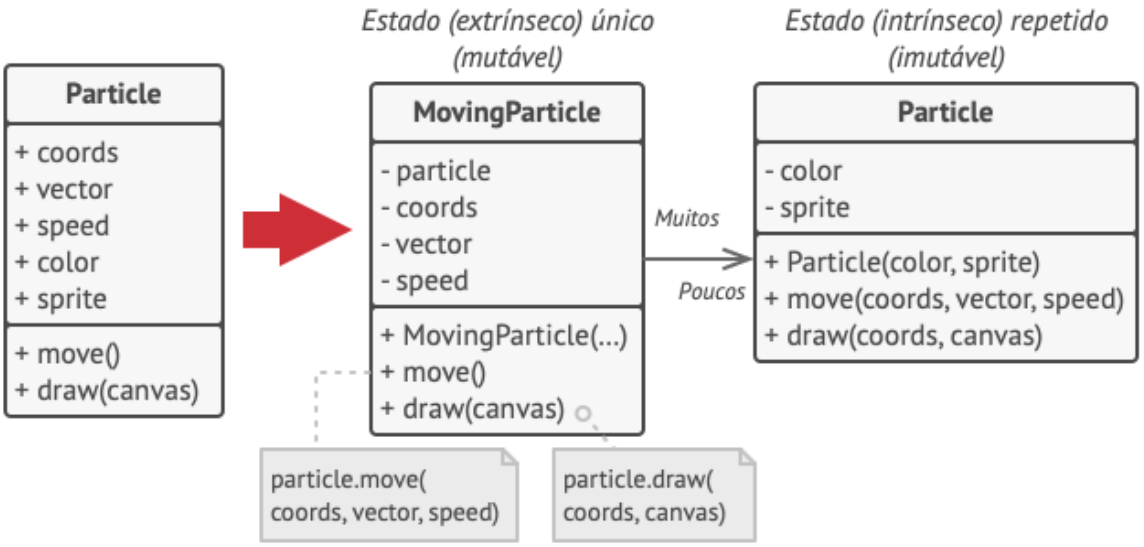
1.5 Facade

Padrão facade tem como princípio ocultar as complexidades do sistema e fornecer uma interface para o cliente na qual o mesmo pode acessar partes do sistema
sem necessariamente saber como a função é feita, mas sim o que ela faz. Assim como a fachada de um edifício, quem vê por fora, enxerga apenas as janelas mas não
consegue ver por dentro do edifício, por esse motivo o padrão de arquitetura tem o nome de facade.
As principais consequências de usar o padrão facade são:
- o sistema se torna mas simples de user usado;
- protege os componentes do sistema;
- permite fraco acoplamento entre os subsistemas e clientes;



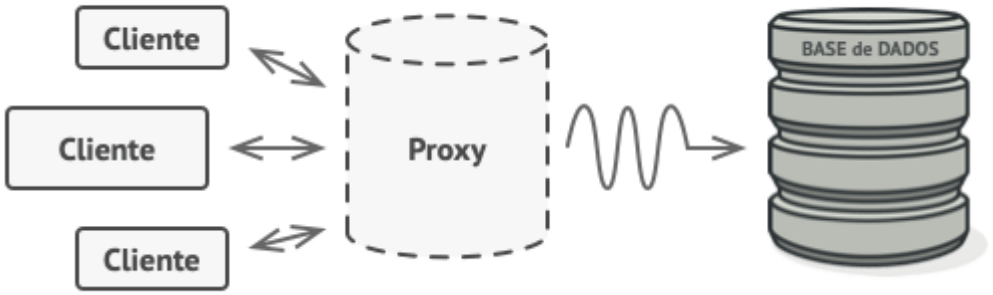
1.6 FlyWeight

O Flyweight é um padrão estrutural que fornece maneiras de diminuir o número de objetos que são criados, o consumo de memória e por consequência aumentar o desempenho.
Além disso, o padrão flyweight fornece maneiras de diminuir a contagem de objetos, melhorando assim a estrutura de objetos do aplicativo.
Uma das principais funções do flyweght é a reutilização de objetos de tipos semelhantes que já existem, armazenando e criando um novo objeto somente quando nenhum objeto correspondente é existente.



1.7 Proxy

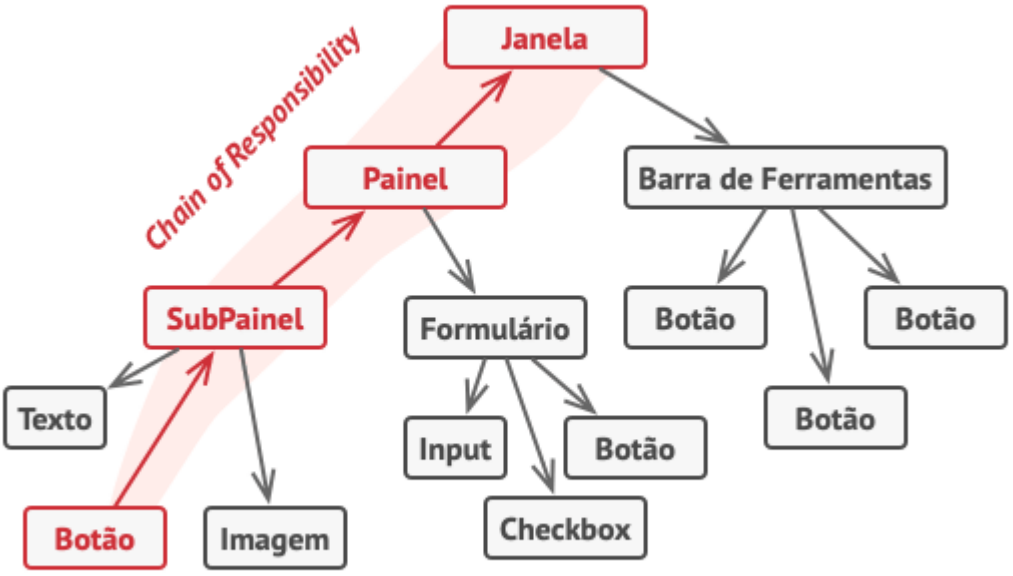
O proxy é um padrão estrutural que controla e gerencia o acesso ao objeto que está protegido. A classe proxy na teoria pode se conectar a qualquer objeto,representando a funcionalidade de outra classe. No padrão proxy é criado um objeto com um objeto real para fazer a interface da funcionalidade se tornar pública. O Proxy é utilizado pois existem algumas situações que ocorrem dependência de um recurso remoto, ou seja, pode existir latência de rede ou mesmo um objeto que leva muito tempo para carregar.



2.0 Comportamentais

2.1 Chain of responsibility

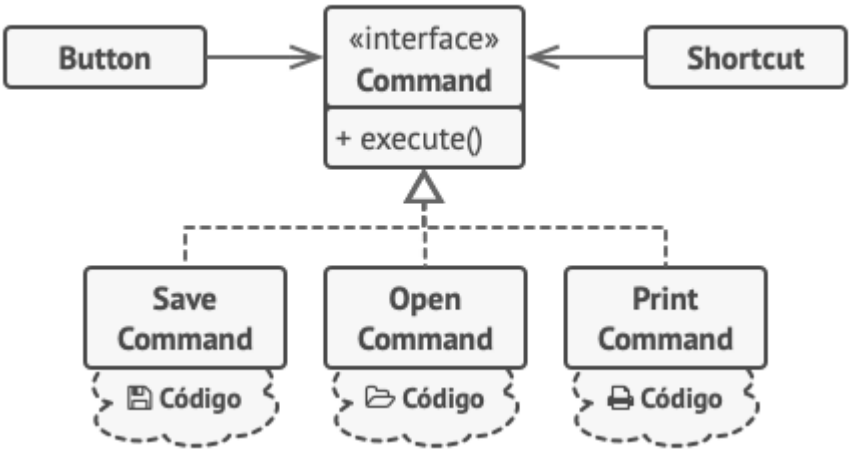
O Padrão de Design cadeia de responsabilidade é usado para obter um acoplamento fraco. O funcionamento dele é simples de ser entendido, o cliente faz uma solicitação e a solicitação é processada por uma cadeia de objetos, e cada objeto da cadeia decide quem processará a solicitação e se a mesma deve ser enviada para o próximo objeto da cadeia.
Além disso, com o padrão de cadeia de responsabilidade, a manutenção do sistema fica mais fácil pois ele tem o princípio aberto/fechado e princípio de responsabilidade única.



Uma corrente pode ser formada por uma secção de um objeto.

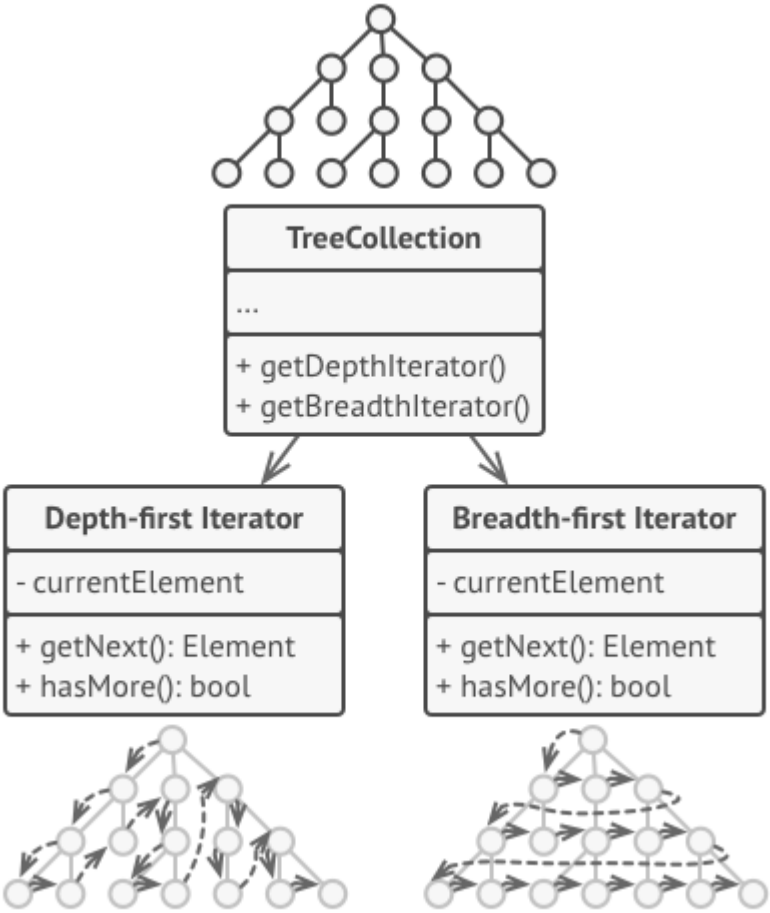
2.2 Command

O Command é um dos padrões comportamentais de projeto, possuindo como principal objetivo funcionar como uma camada de abstração responsável por transformar uma solicitação em um objeto que irá, por sua vez, conter todas as informações necessárias para executá-la.



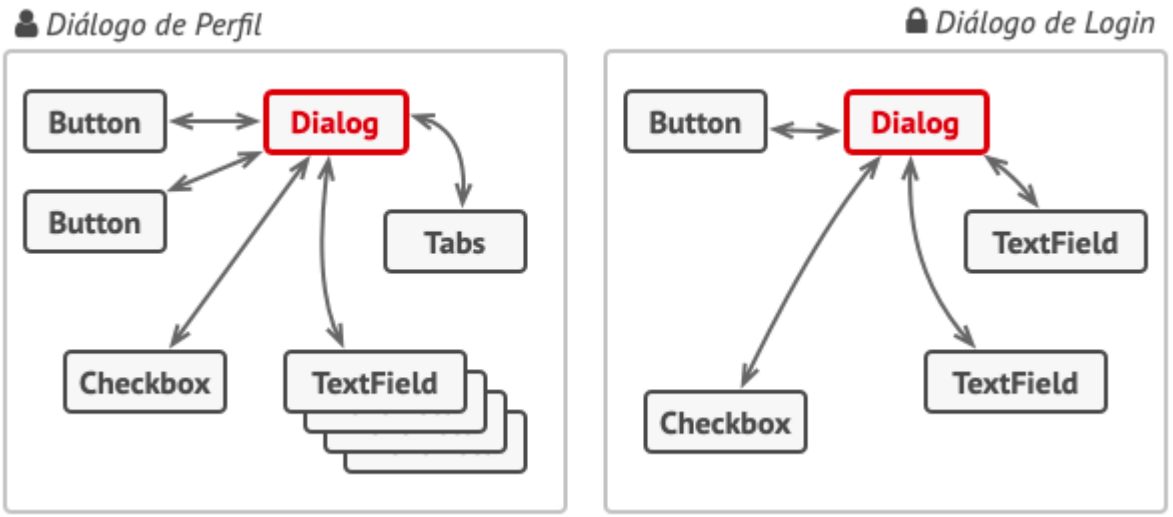
2.3 Iterator

Um Iterator é um padrão de projeto comportamental focado, principalmente, em percorrer os elementos de alguma coleção de dados, sendo elas arrays, listas, tuplas, árvores, entre outras estruturas de dados. O principal objetivo de um objeto iterator é realizar esta iteração em uma coleção de dados sem conhecer a estrutura interna desta coleção.



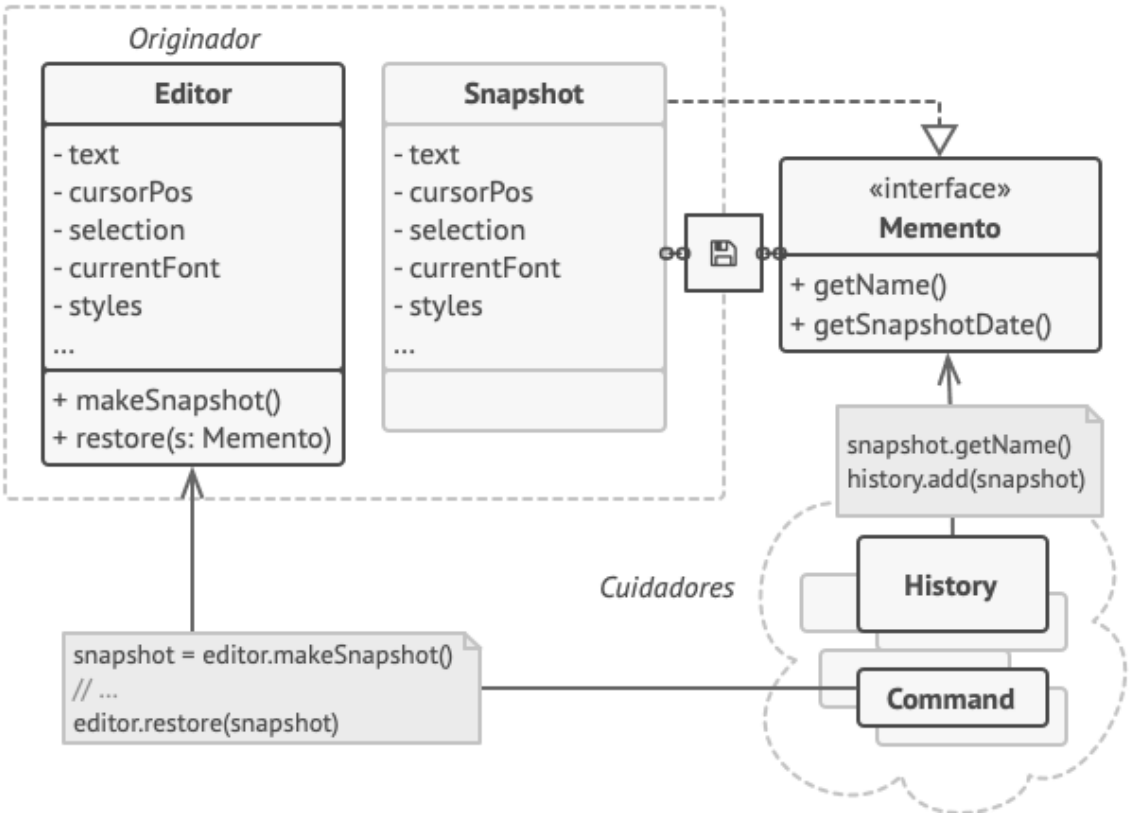
2.4 Mediator

O Mediator tem por objetivo principal não permitir que objetos se comuniquem de forma direta e, com isso faz com que tenham um fraco acoplamento tendo em vista que reduz as dependências entre eles, ou seja, toda a comunicação é feita por meio de um objeto Mediator e deste jeito, a única dependência das demais classes se torna o objeto Mediator.



2.5 Memento

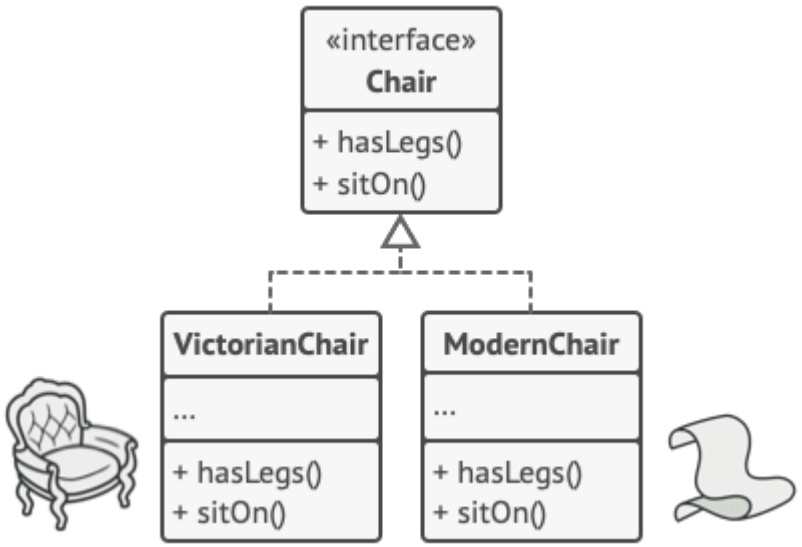
O Memento é um padrão de projeto responsável pela criação de Snapshots de determinadas classes do projeto, sempre respeitando toda a estrutura de encapsulamento da classe que originou o Snapshot. O Memento, além de armazenar o Snapshot de determinada classe deve também conseguir fazer o restore.



3.0 Criação

3.1 Abstract Factory

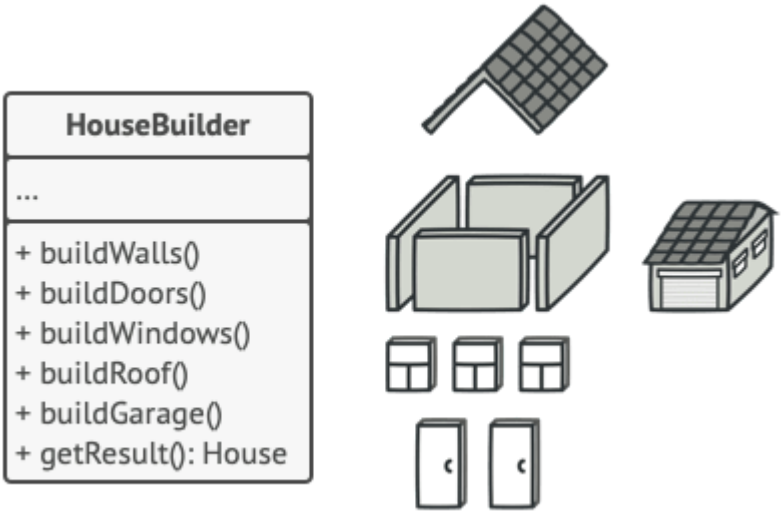
É um padrão de projeto criacional que permite que você produza famílias de objetos relacionados sem ter que especificar suas classes concretas.
A primeira coisa que ele sugere é declarar explicitamente interfaces para cada produto distinto da família de produtos (ex: cadeira, sofá ou mesa de centro). Então você pode fazer todas as variantes dos produtos seguirem essas interfaces. Por exemplo, todas as variantes de cadeira podem implementar a interface `Cadeira`; todas as variantes de mesa de centro podem implementar a interface `MesaDeCentro`, e assim por diante.



Sua utilização é recomendada quando seu código precisa trabalhar com diversas famílias de produtos relacionados, mas que você não quer depender de classes concretas daqueles produtos-eles podem ser desconhecidos de antemão ou você simplesmente quer permitir uma futura escalabilidade.

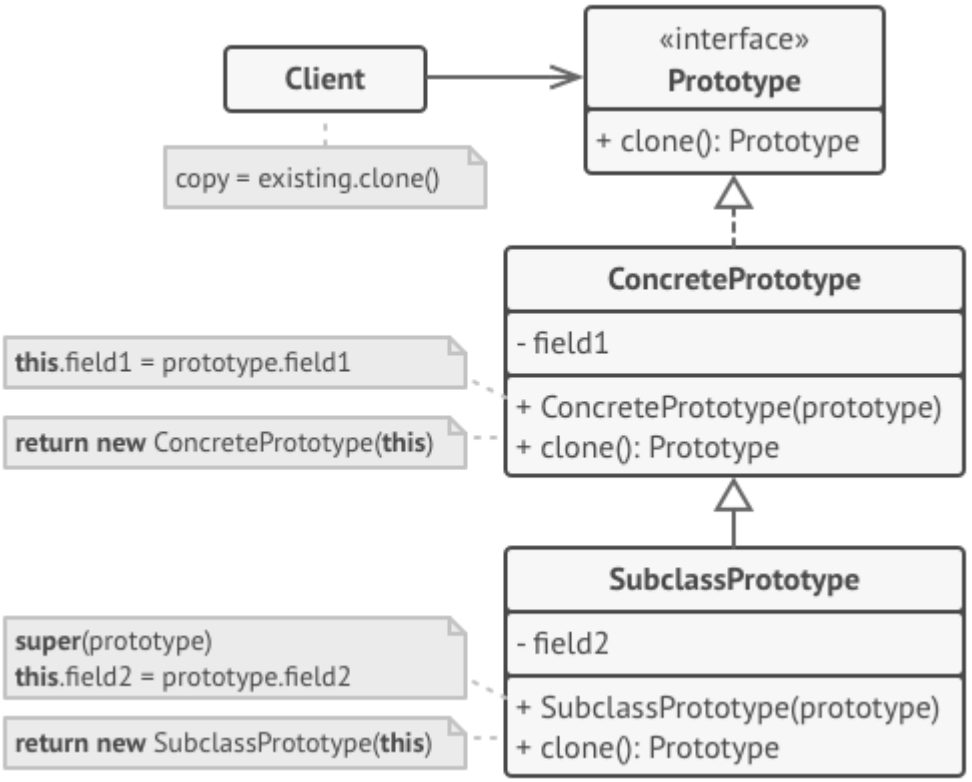
3.2 Builder

O Builder é um padrão de projeto criacional que permite a você construir objetos complexos passo a passo. O padrão permite que você produza diferentes tipos e representações de um objeto usando o mesmo código de construção.
Sua utilização é recomendada quando você quer que seu código seja capaz de criar diferentes representações do mesmo produto (por exemplo, casas de pedra e madeira) ou para construir árvores Composite ou outros objetos complexos.



3.3 Prototype

O Prototype é um padrão de projeto criacional que permite copiar objetos existentes sem fazer seu código ficar dependente de suas classes.
A utilização é recomendada quando seu código não deve depender de classes concretas de objetos que você precisa copiar ou quando precisa reduzir o número de subclasses que somente diferem na forma que inicializam seus respectivos objetos. Alguém pode ter criado essas subclasses para ser capaz de criar objetos com uma configuração específica.



3.4 Singleton

O Singleton é um padrão de projeto criacional que permite a você garantir que uma classe tenha apenas uma instância, enquanto provê um ponto de acesso global para essa instância.
Pode ser utilizado quando uma classe em seu programa deve ter apenas uma instância disponível para todos seus clientes; por exemplo, um objeto de base de dados único compartilhado por diferentes partes do programa ou quando você

precisa de um controle mais estrito sobre as variáveis globais.