



PADRÃO DE PROJETO - DECORATOR

EQUIPE:

APARÍCIO RIAN SILVA

FILIPPE SILVA NASCIMENTO


FRANCISCO RAMON S. HOLANDA

HECTOR JOSÉ R. SALGUEIROS

LIZANDRO WELSON H. C. GALDINO

SHEILA PALOMA S. BRITO

PICOS, 2022

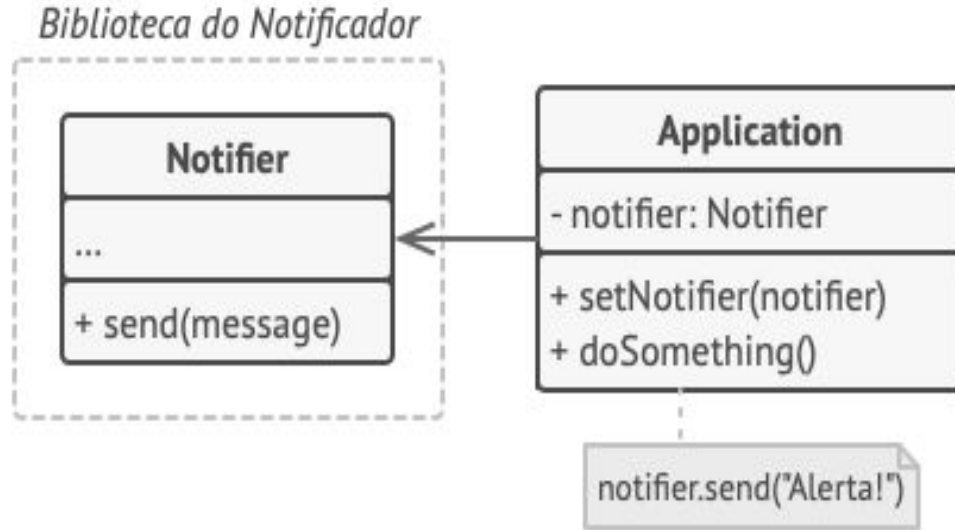
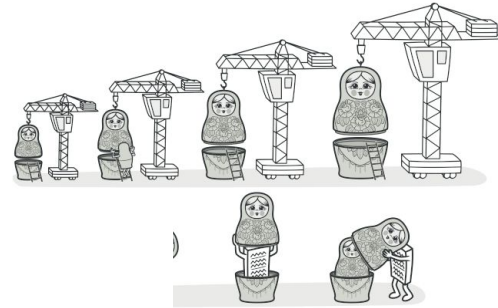


Padrão de projeto estrutural que permite acoplar novos comportamentos para objetos ao colocá-los dentro de invólucros de objetos que contém os comportamentos.

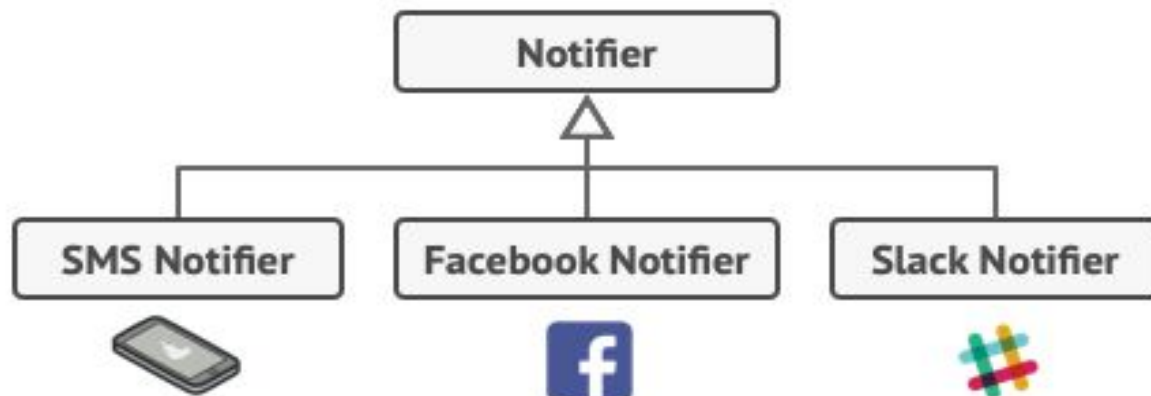
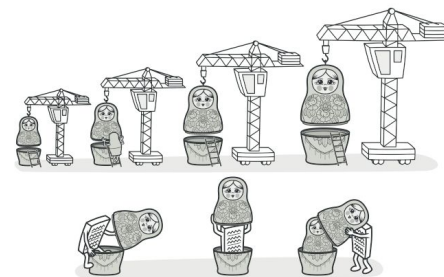
A versão inicial da biblioteca => baseada na classe Notificador (um construtor, e um único método, enviar).



Um programa poderá utilizar a classe notificador para enviar notificações sobre eventos importantes para um conjunto predefinido de emails.

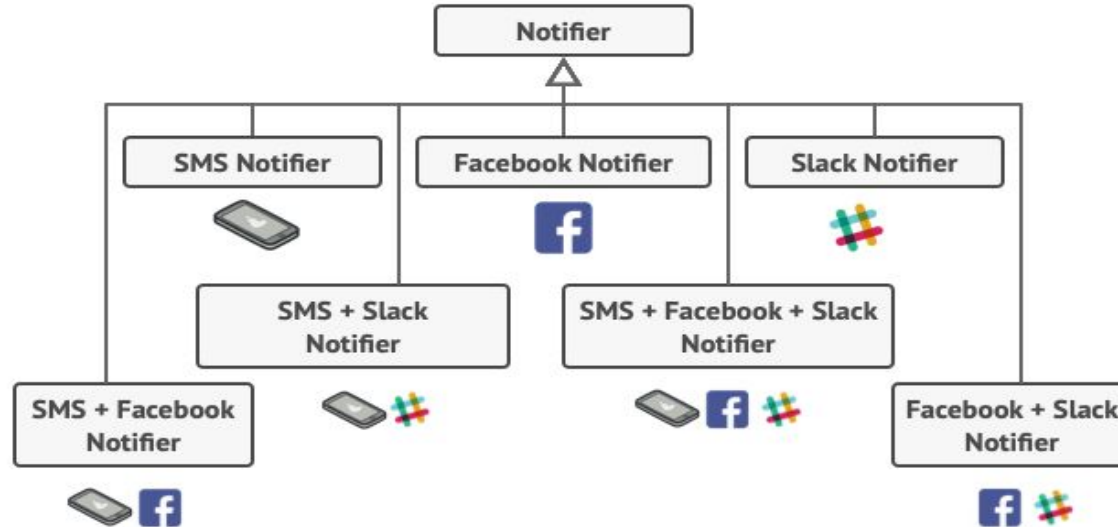
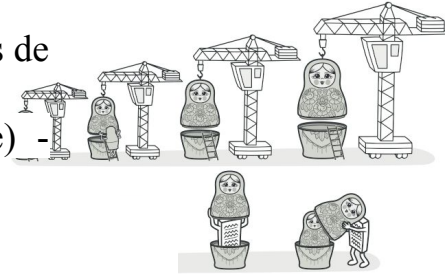


- Cada tipo de notificação é implementada em uma subclasse do notificador.
- Estende-se a classe Notificador e acrescenta métodos de notificação nas novas subclasses.
- O cliente deve ser instanciado à classe de notificação que deseja utilizar em futuras notificações.



- Por que não realizar diversos tipos de notificação de uma só vez?
- Seria necessário criar subclasses especiais que combinam diversos tipos de métodos de notificação dentro de uma classe.
- Consequência, irá inflar o código imensamente (biblioteca e o código cliente) -

Combinação explosiva de subclasses





Herança

- A herança é estática.
- As subclasses só podem ter uma classe pai.
- O próprio objeto é capaz de fazer a função, herdando o comportamento da sua superclasse.

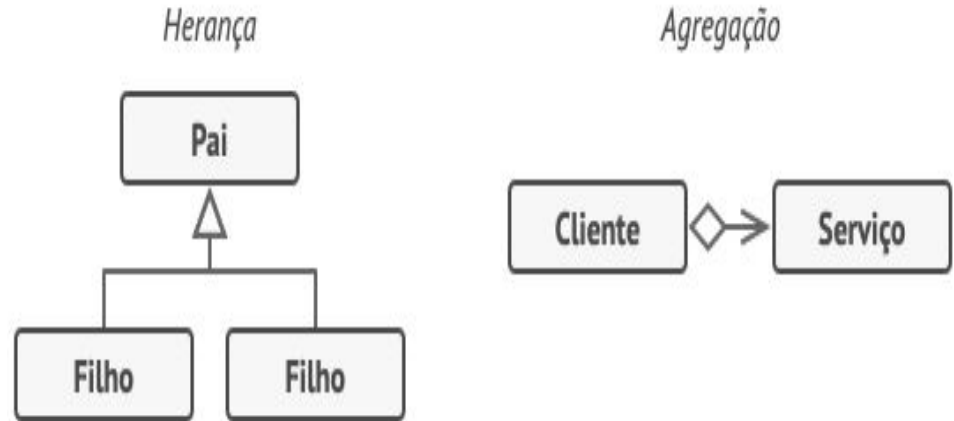


Agregação/Composição

- **Agregação:** O objeto A contém objeto B, B pode viver sem A.
- **Composição:** O objeto A consiste de objetos B; A gerencia o ciclo de vida de B; B não vive sem A.
- Um objeto tem uma referência com outro e delega alguma funcionalidade

Agregação/Composição

- Um objeto pode usar o comportamento de várias classes.
- Ter referências a múltiplos objetos.
- Delegar qualquer tipo de trabalho a eles.



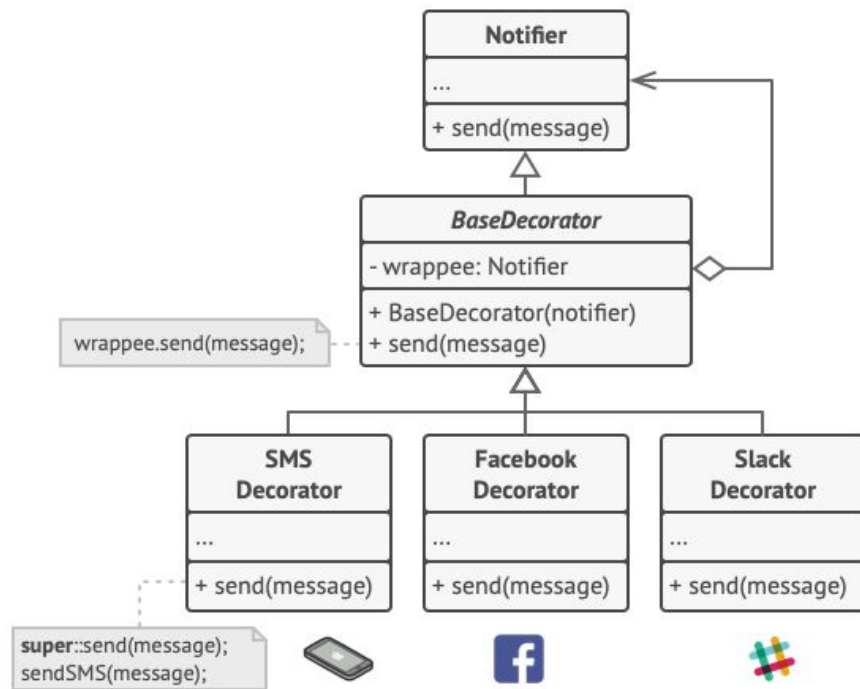
Herança vs. Agregação



Envoltório (Wrapper)

- Apelido para o padrão **Decorator** - é um objeto que pode ser ligado com outro objeto *alvo*.
- Contém o mesmo conjunto de métodos que o alvo e delega-o todos os pedidos que recebe.
- Implementar a mesma interface que o objeto envolvido.
- Permitirá cobrir um objeto em múltiplos envoltórios, adicionando o comportamento combinado de todos os envoltórios a ele.

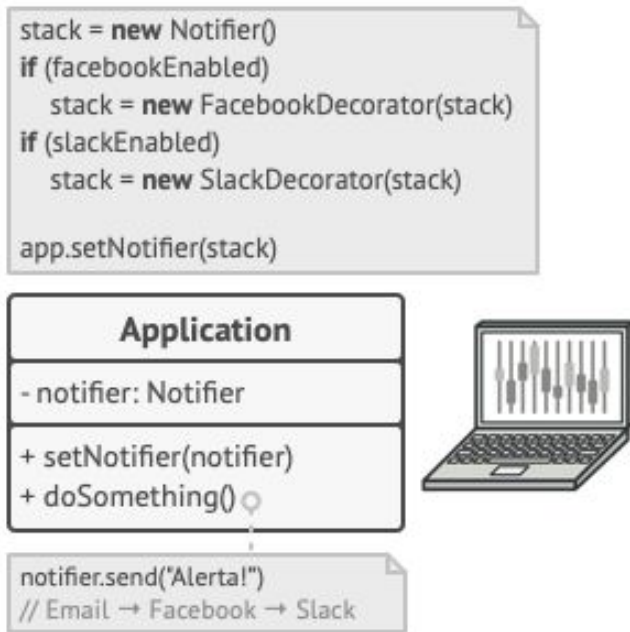
- Ex: manter o comportamento de notificação por email dentro da classe Notificador base, mas transformar todos os métodos de notificação em decoradores.



Vários métodos de notificação se tornam decoradores.



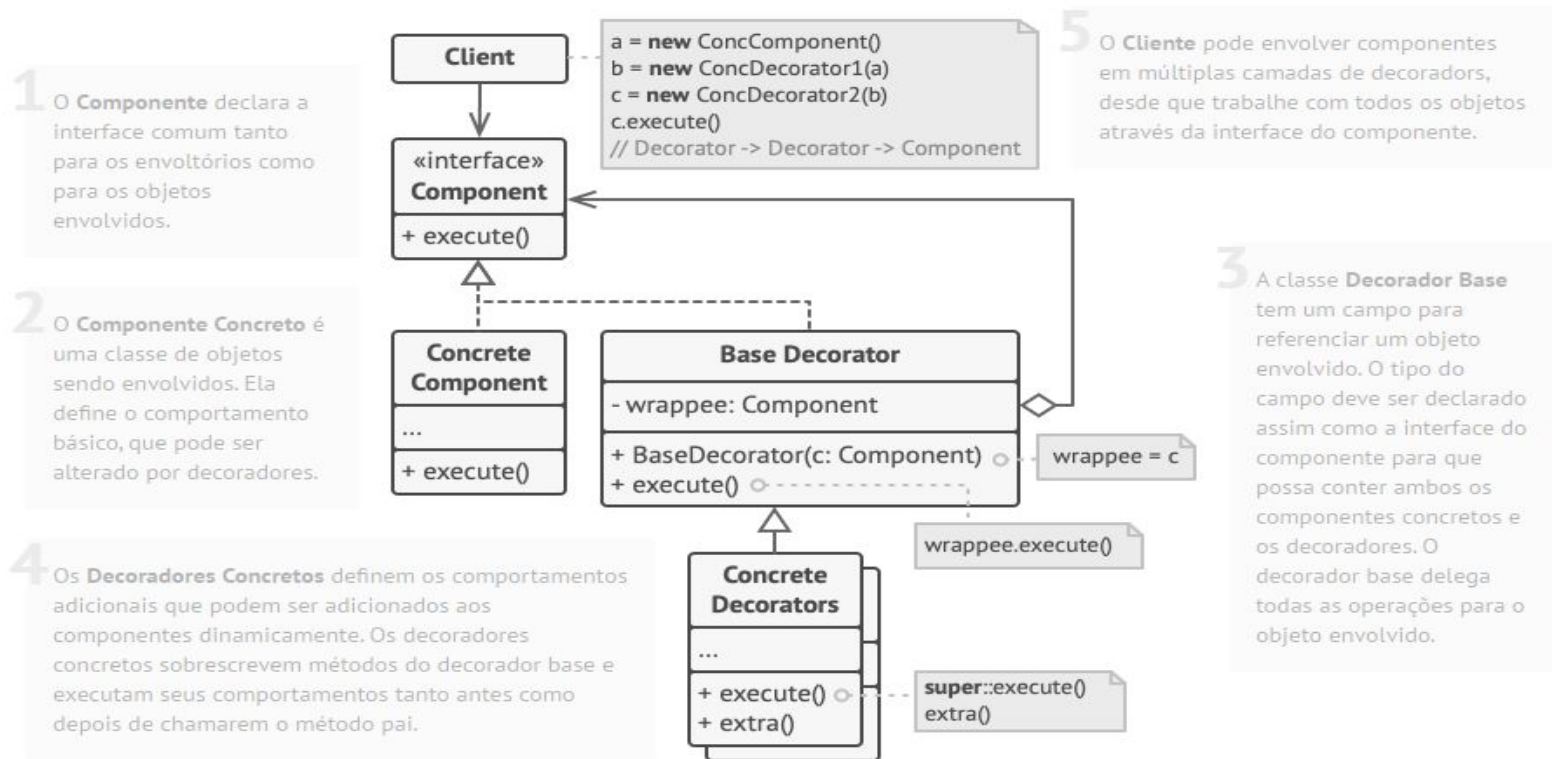
- O código cliente precisa envolver um objeto notificador básico em um conjunto de decoradores que coincidem com as preferências do cliente.
- Os objetos resultantes serão estruturados como uma pilha.
- O último decorador na pilha seria o objeto que o cliente realmente trabalha.



As aplicações pode configurar pilhas complexas de notificações decoradores

PADRÃO DE PROJETO - DECORATOR

🏗️ Estrutura

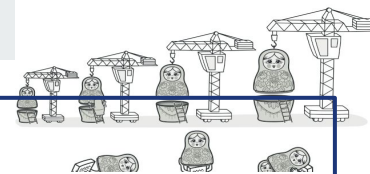


Aplicabilidade



- O padrão *Decorator* permite comprimir e encriptar dados sensíveis independentemente do código que verdadeiramente usa esses dados.
- A aplicação envolve o objeto da fonte de dados com um par de decoradores. Ambos invólucros mudam a maneira que os dados são escritos e lidos no disco:
 - Antes dos dados serem **escritos** no disco, os decoradores encriptam e comprimem;
 - Antes dos dados serem **lidos** do disco, ele passa pelos mesmos decoradores que descomprimem e decodificam.
- Os decoradores e a classe da fonte de dados implementam a mesma interface, que os torna intercomunicáveis dentro do código cliente.

PADRÃO DE PROJETO - DECORATOR



Aplicabilidade:

- projetar comportamentos adicionais para objetos em tempo de execução sem quebrar o código;
- É complicado/impossível estender o comportamento de um objeto usando herança.

Vantagens	Desvantagens
Estender o comportamento de um objeto sem fazer uma nova subclasse.	Difícil remover um invólucro de uma pilha de invólucros.
Adicionar ou remover responsabilidades de um objeto no momento da execução.	Difícil implementar um decorador de tal maneira que seu comportamento não dependa da ordem do pilha de decoradores.
Combinar diversos comportamentos ao envolver o objeto com múltiplos decoradores.	A configuração inicial do código de camadas pode tornar-se pouco atrativa
<i>Princípio de responsabilidade única.</i> dividir uma classe monolítica em diversas classes menores.	



Aplicações que utilizam decorator

Flask



Flask

web development,
one drop at a time

django

django

PADRÃO DE PROJETO - DECORATOR

Implementação.

