Design Patterns: Mediator & Decorator

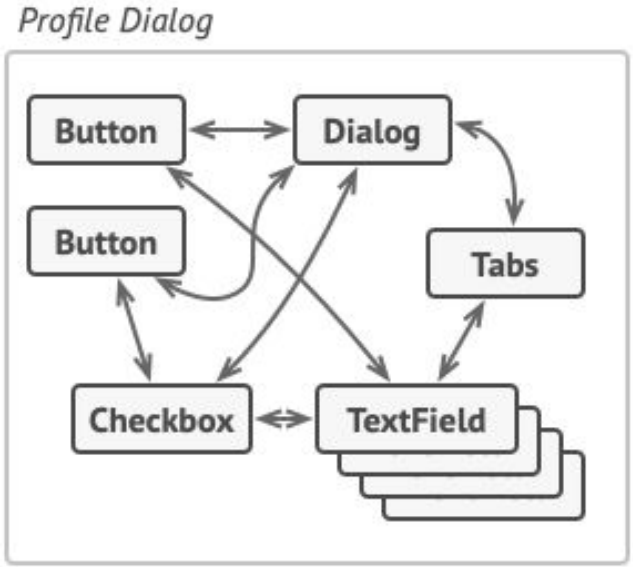
Arquiteturas Aplicacionais

Elaborado por:
Ana Murta (PG50184)
Beatriz Oliveira (PG50942)
Gonçalo Soares (PG50393)
Joana Alves (PG50457)
Vicente Moreira (PG50799)

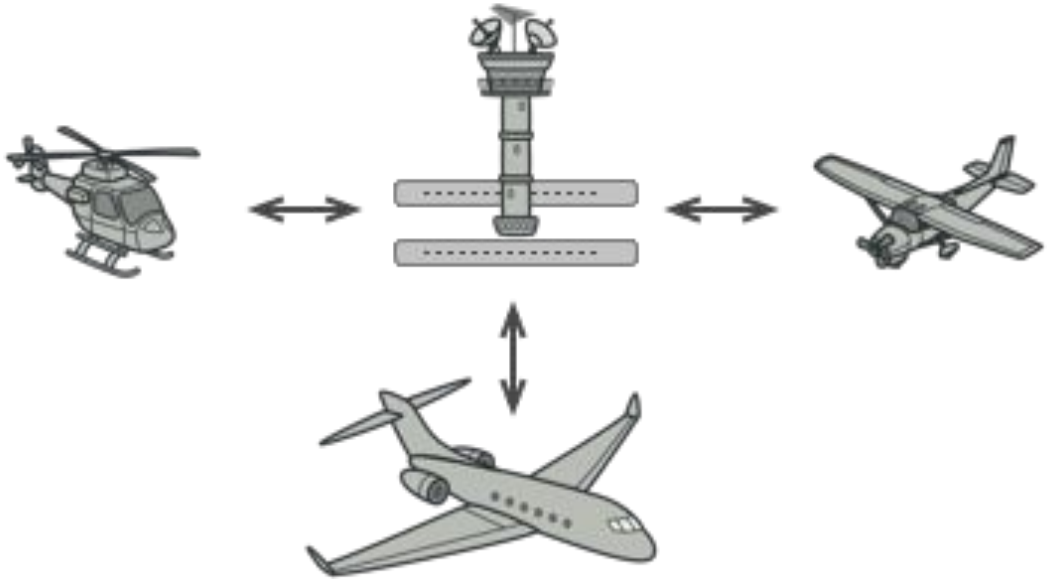
Mediator - Problem

- As aplicações podem conter classes e elementos que interagem e dependem uns dos outros.

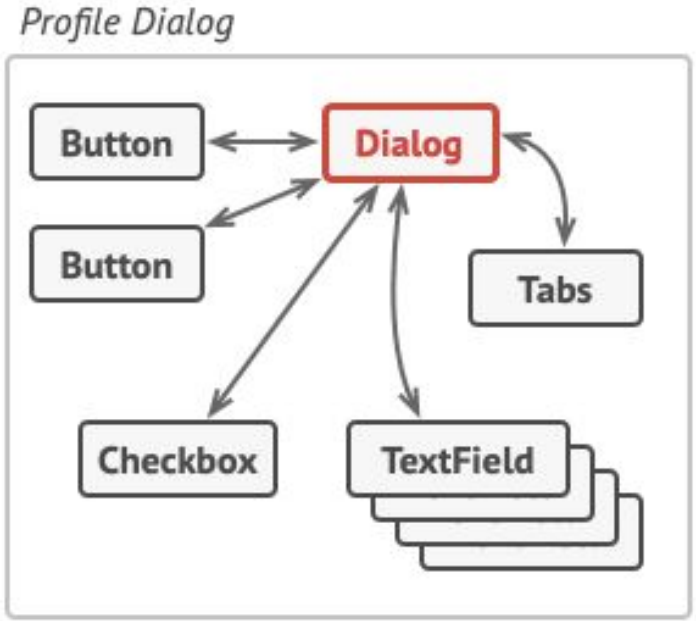
- Solução difícil de gerir com classes pouco reutilizáveis.

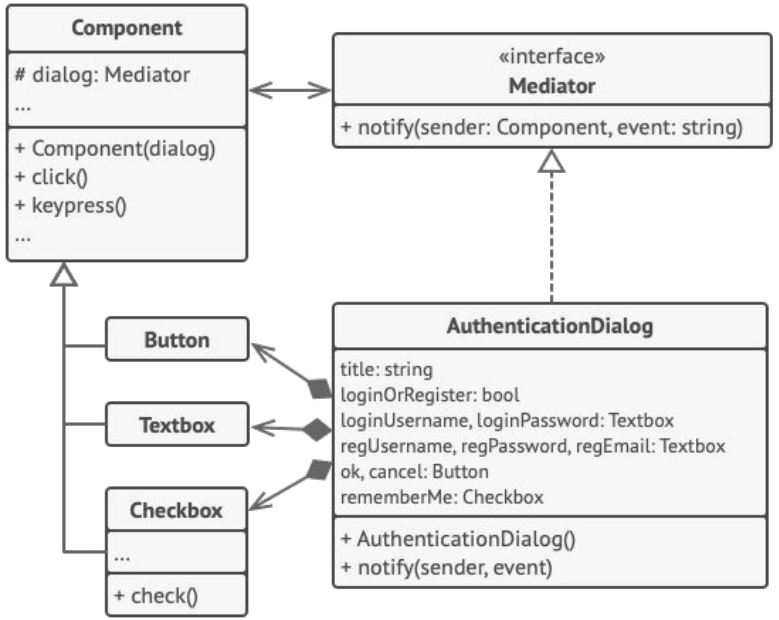


Mediator - Real World Analogy



Mediator - Solution & Implementation





Mediator - Pros & Cons

Redução de dependências entre os objetos, aumentado a reusabilidade.

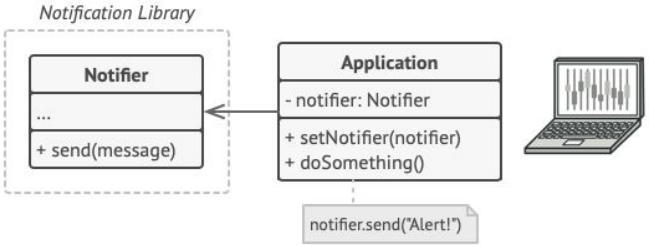
Maior flexibilidade, visto que permite a introdução e remoção de novos objetos.

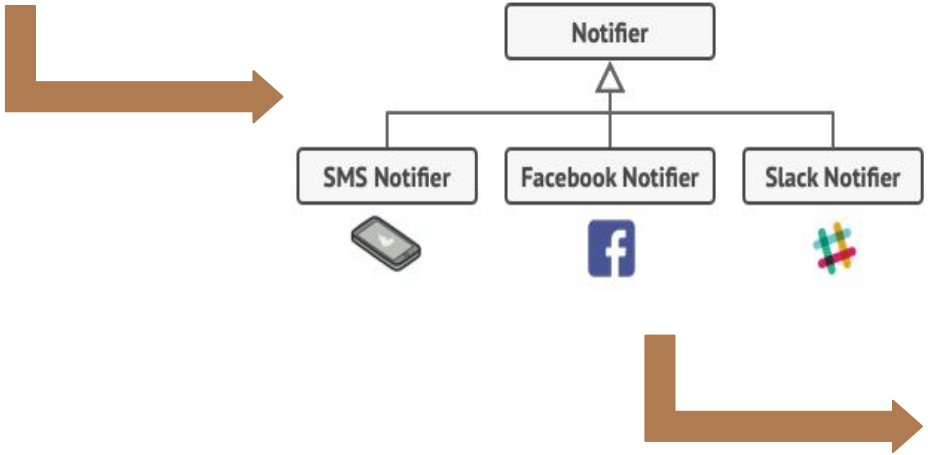
Simplifica e organiza a comunicação num sistema.

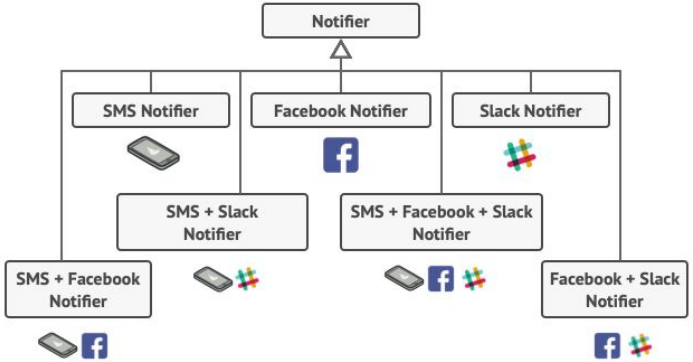
Aumento do *overhead,* visto que as comunicações têm que passar pelo Mediator.

Introdução de um potencial "único ponto de falha".

Decorator - Problem

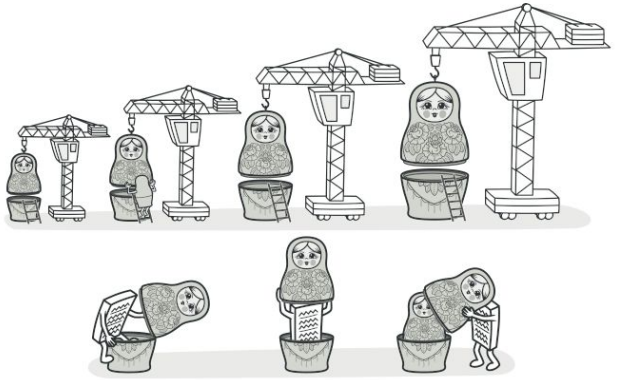




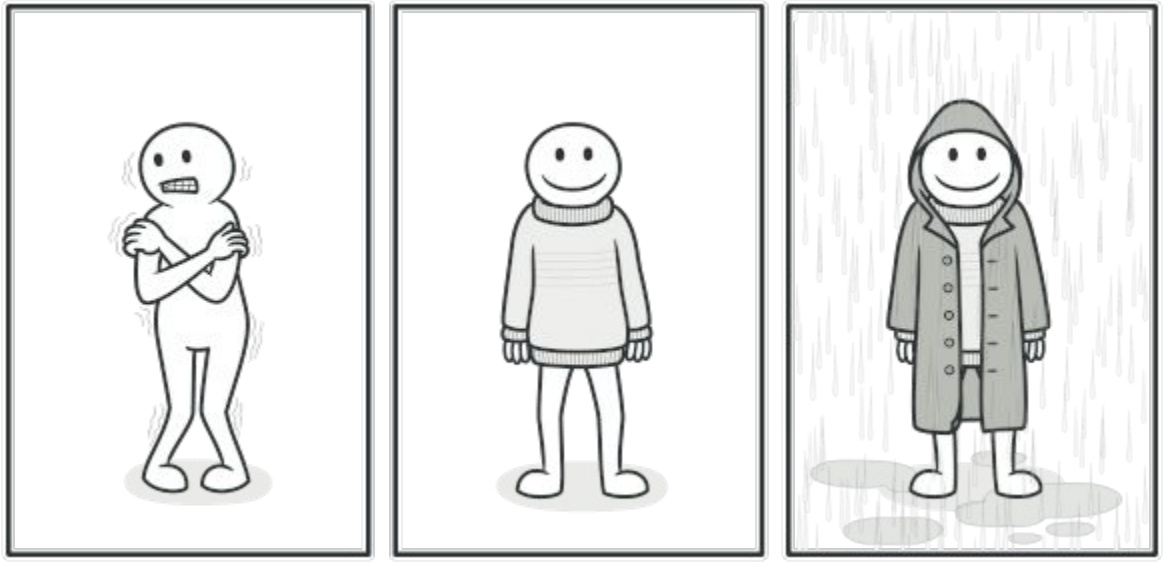


Decorator

- Atribuir, dinamicamente, responsabilidades adicionais a um objeto
- Alternativa flexível à extensão por subclasses



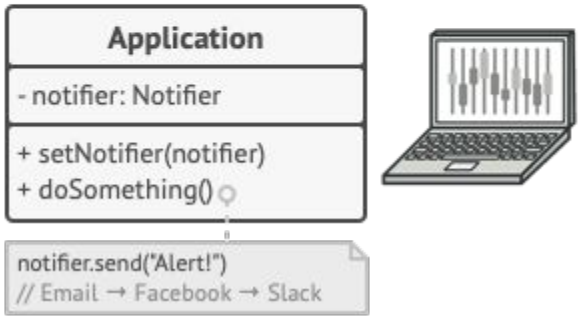
Decorator - Real World Analogy

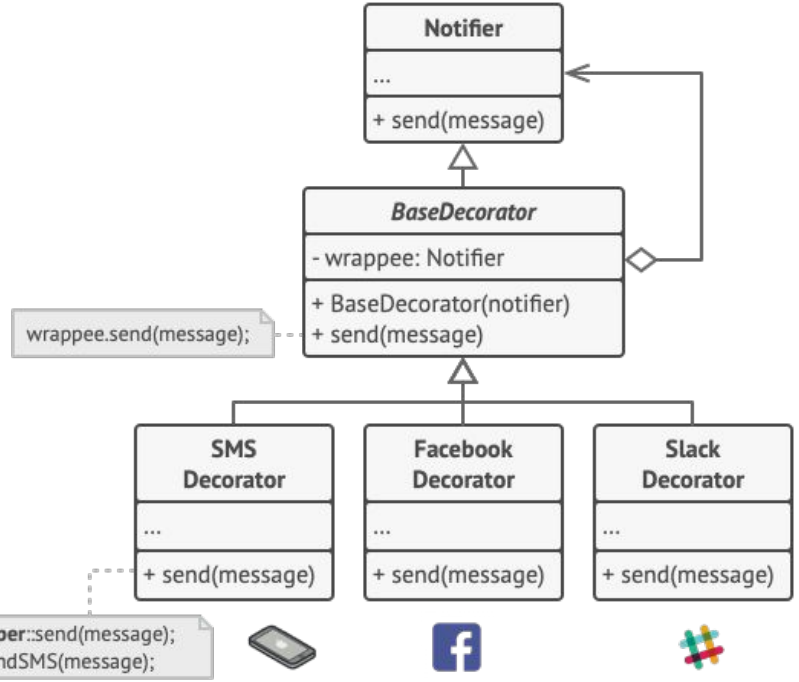


Decorator - Solution & Implementation

```
stack = new Notifier()
if (facebookEnabled)
    stack = new FacebookDecorator(stack)
if (slackEnabled)
    stack = new SlackDecorator(stack)

app.setNotifier(stack)
```





Decorator - Pros & Cons

- É possível estender o comportamento de uma classe sem criar uma nova subclasse
- É possível combinar vários comportamentos utilizando vários *decorators*
- É possível remover e adicionar responsabilidades a um objeto em *runtime*

- É difícil implementar um *decorator* que não dependa da ordem da stack
- É difícil remover um decorator específico da stack