Decorator and Chain of Responsibility Patterns

Sergej Kurakin

Software design pattern is a general, reusable solution to a commonly occurring problem.

https://en.wikipedia.org/wiki/Software_design_pattern

Decorator Pattern

You have Class you want to change.

Modify some behaviour.

You can edit code in that class.

Disadvantages

- Class may become bigger and harder to maintain.
- Tests should be added/changed for that class.
- Some parts of your system may change behaviour.

You can extend that class.

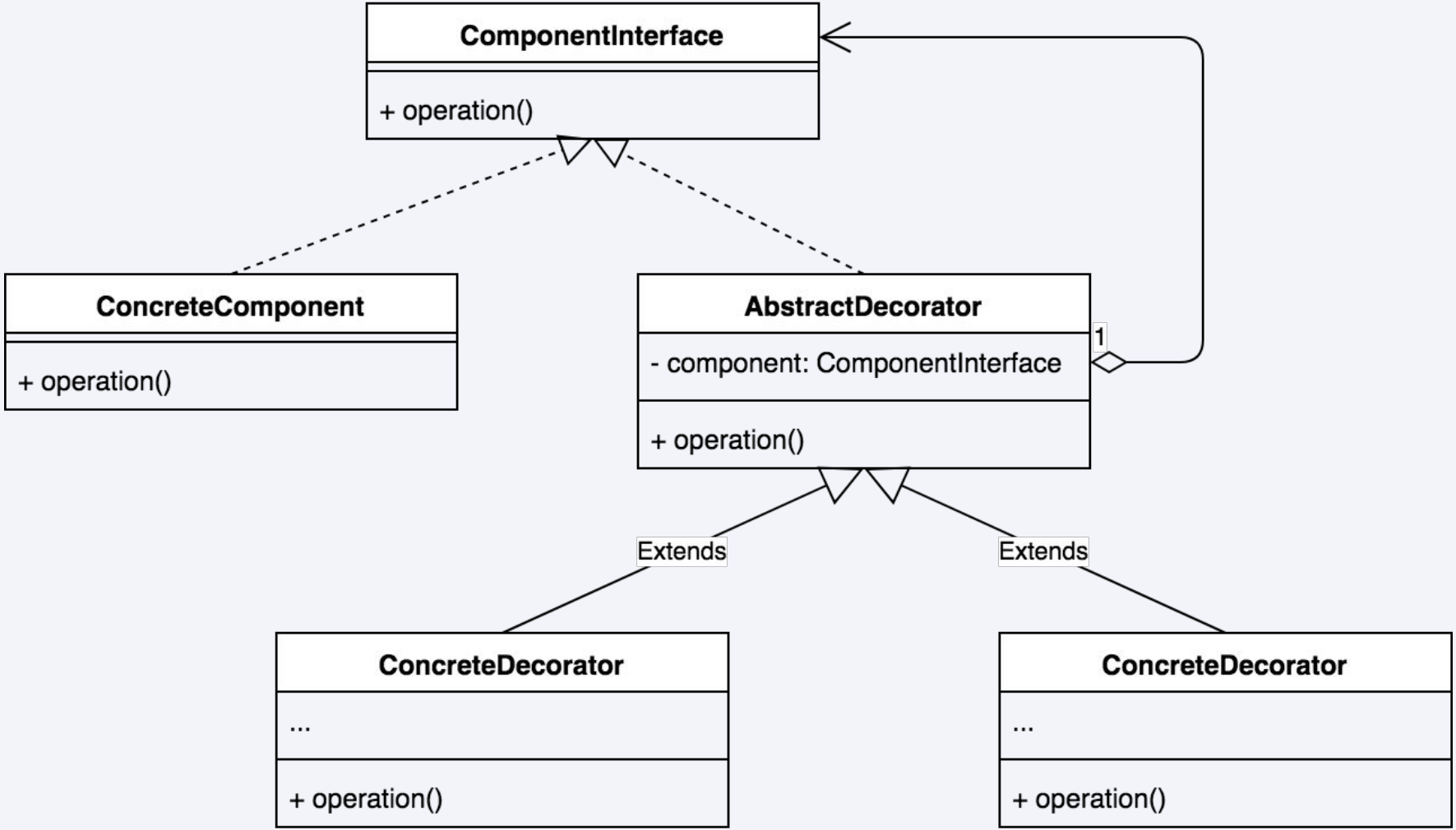
Disadvantages

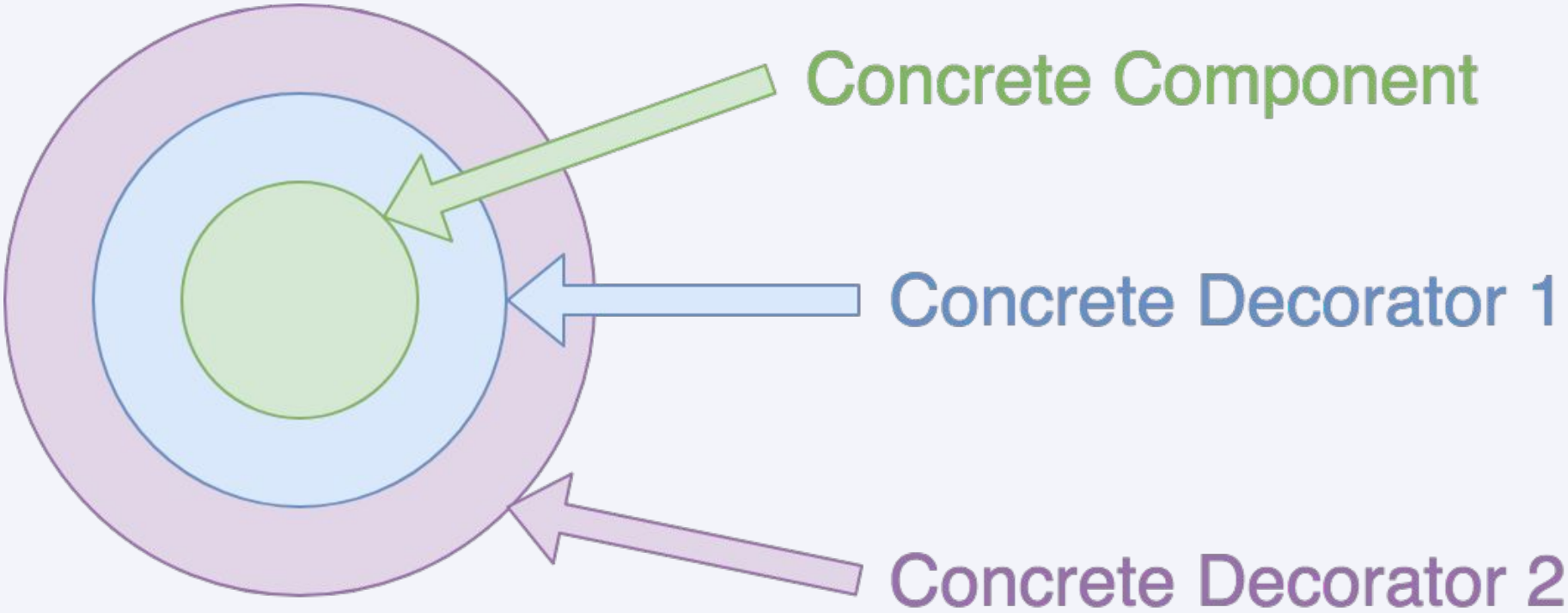
- Tests should be added for extending class.
- Parent Class behaviour may change.
- May cause huge hierarchy of classes.

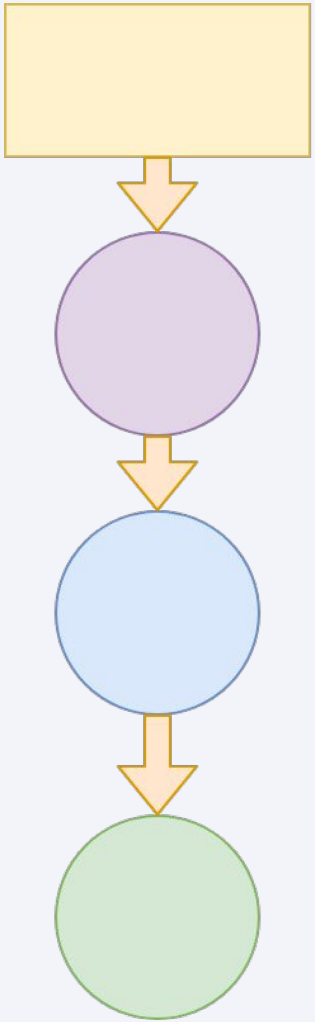
Decorator Pattern

Decorator pattern allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class.

https://en.wikipedia.org/wiki/Decorator_pattern


ComponentInterface
+ operation()
ConcreteComponent
+ operation()
AbstractDecorator
- component: ComponentInterface
+ operation()
1
Extends
Extends
ConcreteDecorator
...
+ operation()
ConcreteDecorator
...
+ operation()


Concrete Component
Concrete Decorator 1
Concrete Decorator 2

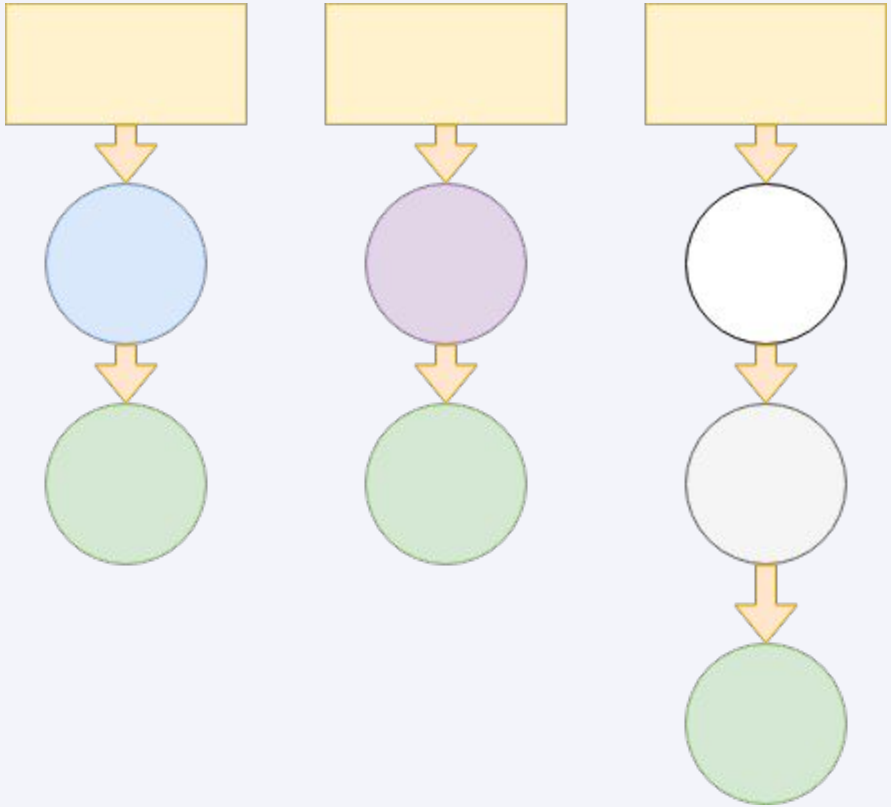


Client

Concrete Decorator 2

Concrete Decorator 1

Concrete Component



Advantages

- Flexible
- Change behaviours
- Composing

Disadvantages

- Multi-wrapped object
- Small classes

Examples

Transformers

Services

Chain of Responsibility Pattern

You have Class you want to change.

Change it’s responsibility.

You can edit code in that class.

Disadvantages

- More and more “if/else” added.
- Tests should be added/changed for that class.
- Class may become huge and harder to maintain.

You can extend that class.

Disadvantages

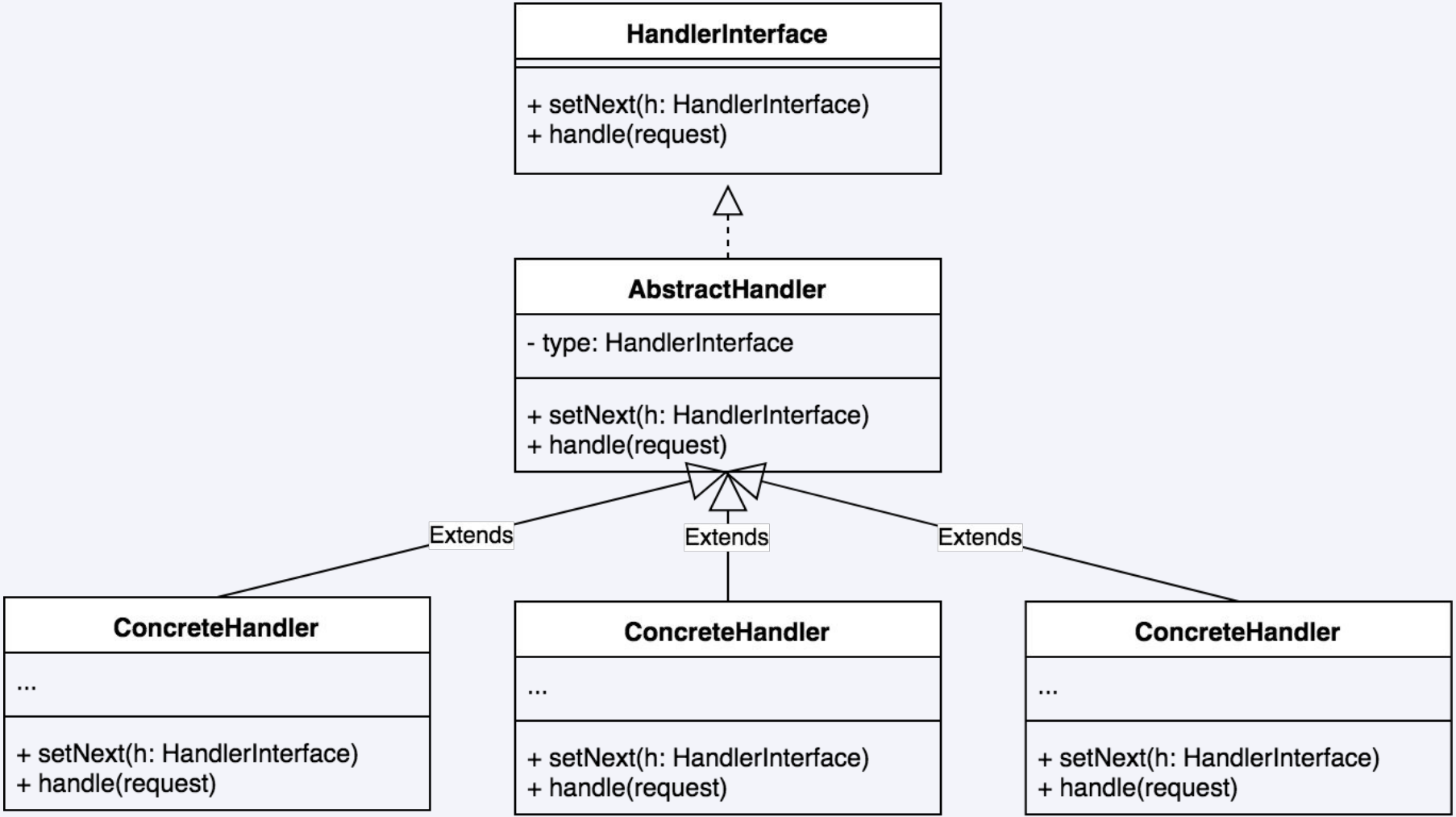
- Tests should be added for extending class.
- Parent Class behaviour may change.
- May cause huge hierarchy of classes.

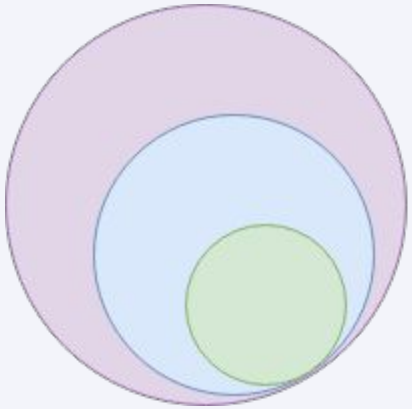
Chain of Responsibility Pattern

Chain of Responsibility pattern consists of a source of command objects and a series of processing objects.

https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern

if ... else if ... else if ... else if ... else if ... else if ... else if ... else if ... else if ... else if ..... else ... endif

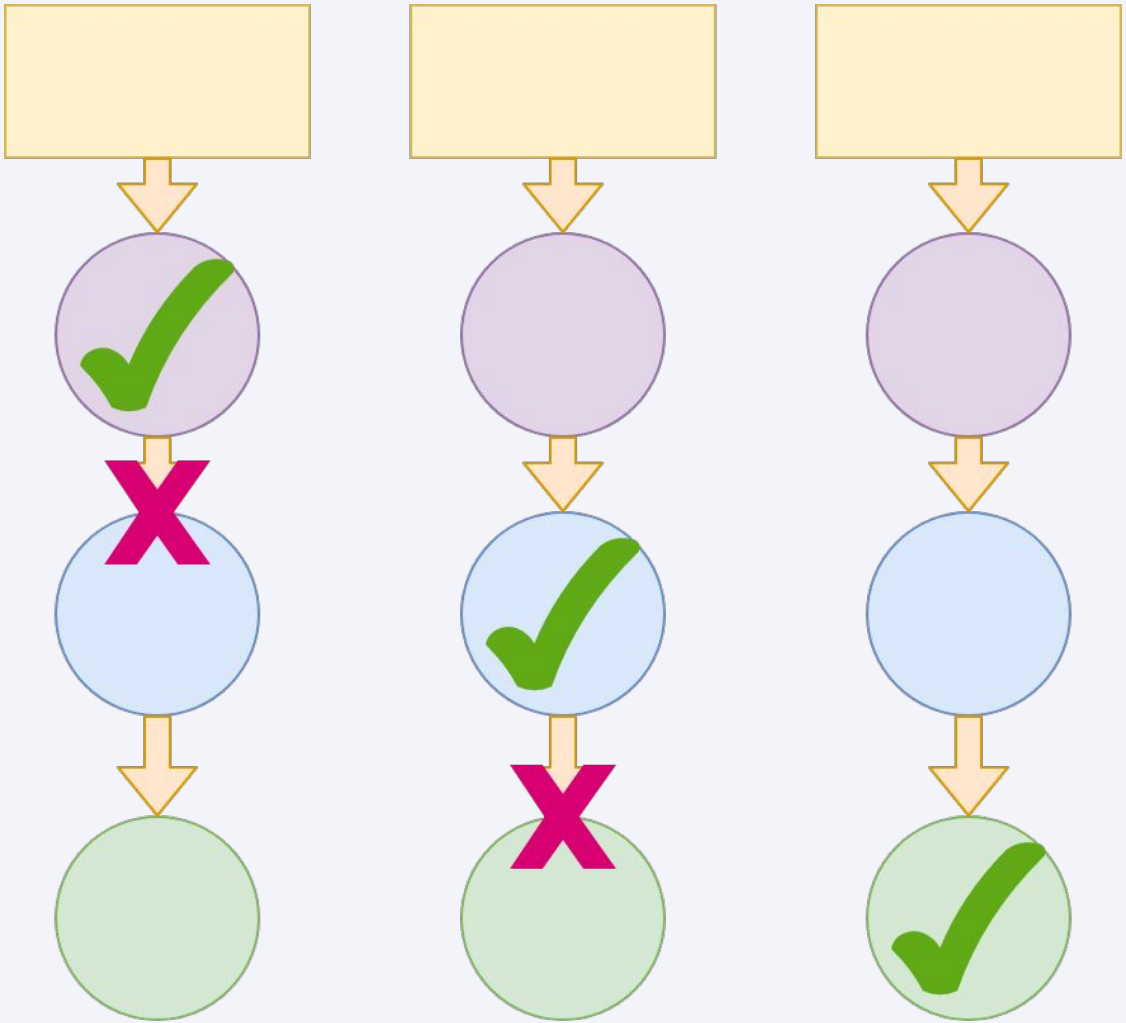

HandlerInterface
+ setNext(h: HandlerInterface)
+ handle(request)
AbstractHandler
- type: HandlerInterface
+ setNext(h: HandlerInterface)
+ handle(request)
Extends
Extends
Extends
ConcreteHandler
...
+ setNext(h: HandlerInterface)
+ handle(request)
ConcreteHandler
...
+ setNext(h: HandlerInterface)
+ handle(request)
ConcreteHandler
...
+ setNext(h: HandlerInterface)
+ handle(request)



Handler 1

Handler 2

Handler 3



Advantages

- Reduces coupling
- Single Responsibility Principle
- Open/Closed Principle

Disadvantages

- Unhandled calls

Examples

Services

Share your thoughts!

Links

https://en.wikipedia.org/wiki/Software_design_pattern
https://en.wikipedia.org/wiki/Decorator_pattern
https://refactoring.guru/design-patterns/decorator
https://www.youtube.com/watch?v=GCraGHx6gso
https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern
https://refactoring.guru/design-patterns/chain-of-responsibility

Thanks!

Oberlo

shopify