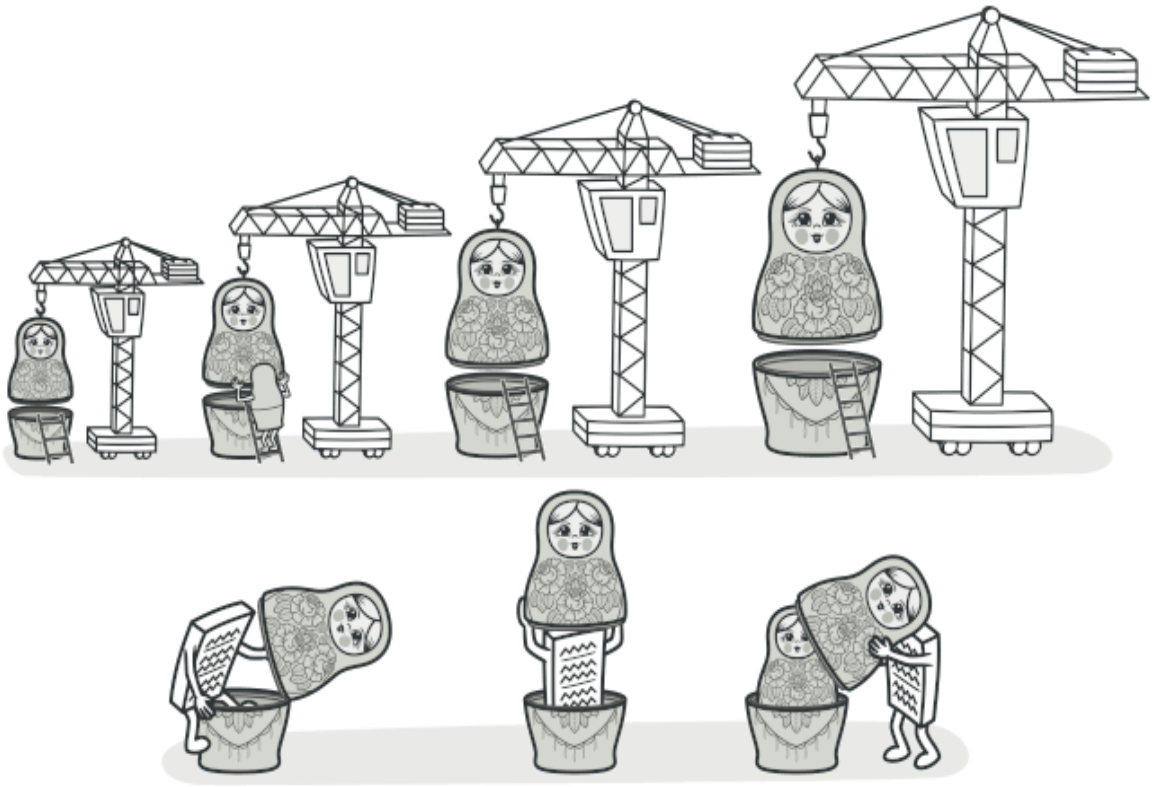
Structural Design Patterns

Decorator

Aka the wrapper

Made by: Ahmed Saladin Ahmed

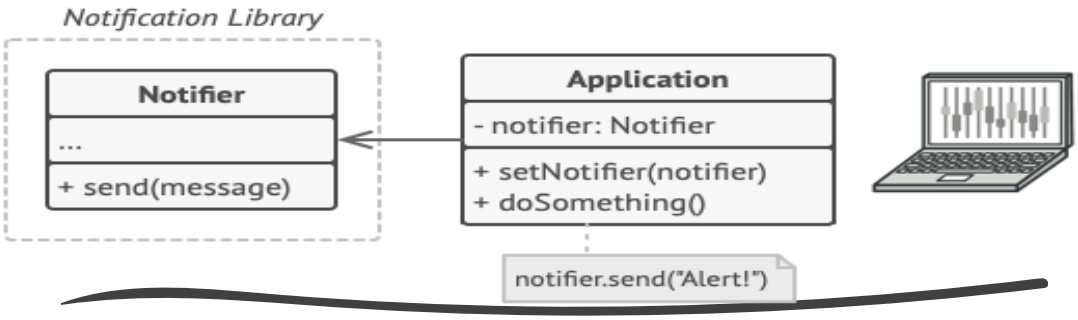


Decorator

is a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.

The problem

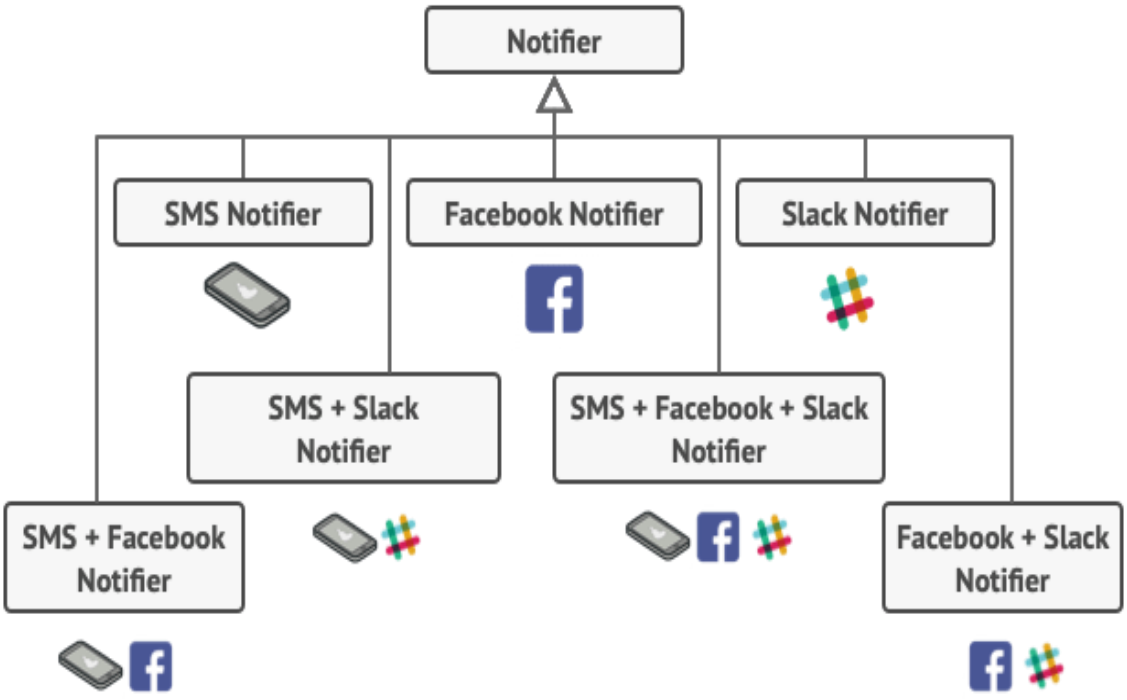
Imagine that you're working on a notification library which lets other programs notify their users about important events.



At some point, you realize that users of the library expect more than just email notifications. Many of them would like to receive an SMS about critical issues. Others would like to be notified on Facebook and, of course, the corporate users would love to get Slack notifications.

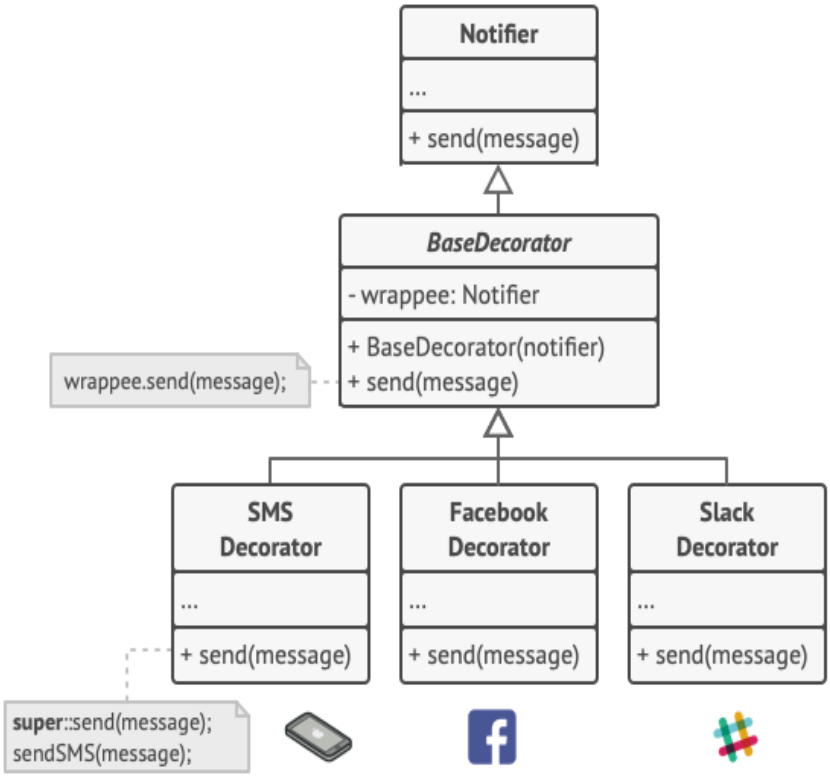
Why can't you use several notification types at once?

How hard can that be? You extended the Notifier class and put the additional notification methods into new subclasses. Now the client was supposed to instantiate the desired notification class and use it for all further notifications.



Solution

structure your business logic into layers, create a decorator for each layer and compose objects with various combinations of this logic at runtime. The client code can treat all these objects in the same way, since they all follow a common interface.



Cons

It’s hard to remove a specific wrapper from the wrappers stack.

It’s hard to implement a decorator in such a way that its behavior doesn’t depend on the order in the decorators stack.

The initial configuration code of layers might look pretty ugly.

pros

You can extend an object’s behavior without making a new subclass.

You can add or remove responsibilities from an object at runtime.

You can combine several behaviors by wrapping an object into multiple decorators.

Single Responsibility Principle. You can divide a monolithic class that implements many possible variants of behavior into several smaller classes.

Reference

https://www.dofactory.com/javascript/design-patterns/decorator

https://www.lambdatest.com/blog/comprehensive-guide-to-javascript-design-patterns/

https://refactoring.guru/design-patterns/decorator