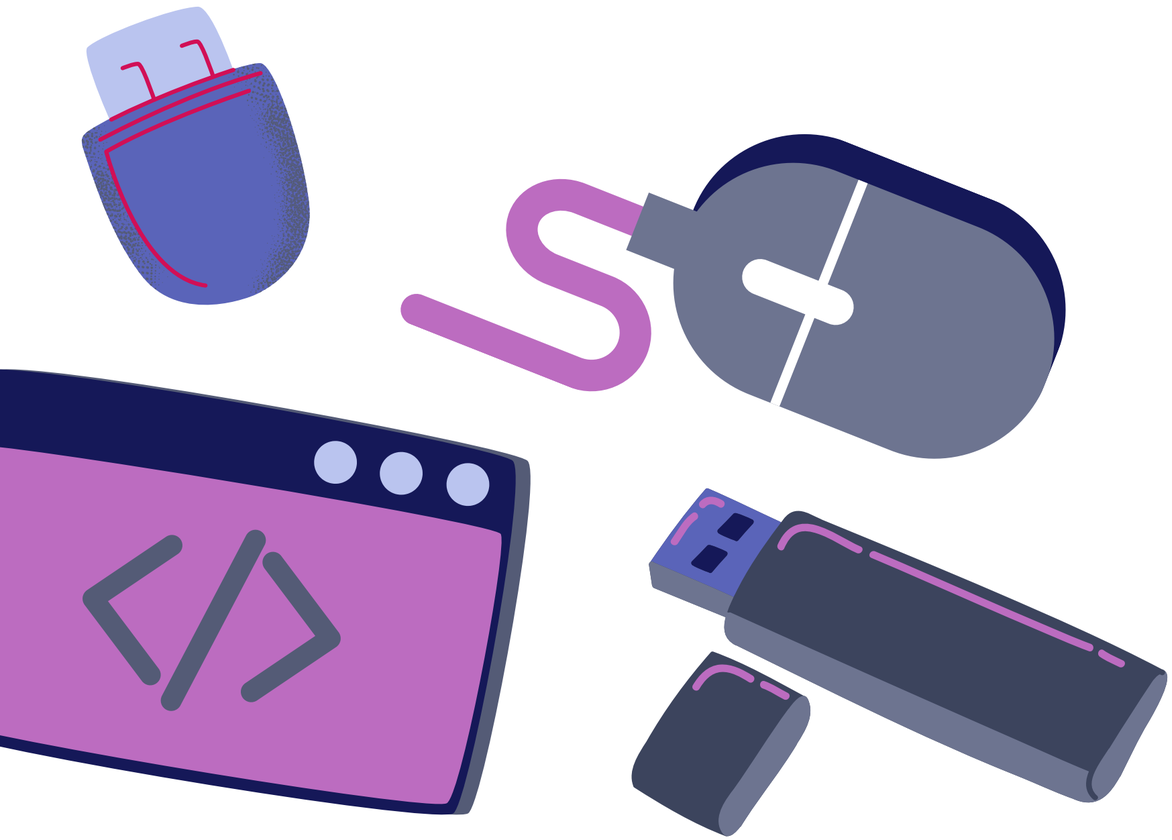
Proyecto Realizado por Iván Caro Romero

DECORATOR

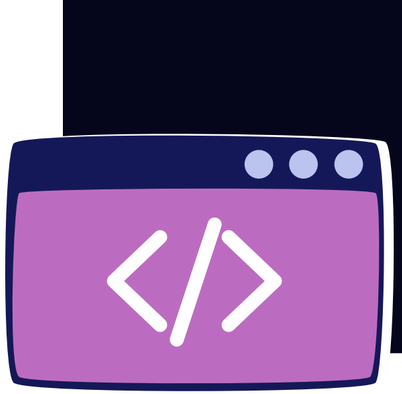
Patrón de diseño

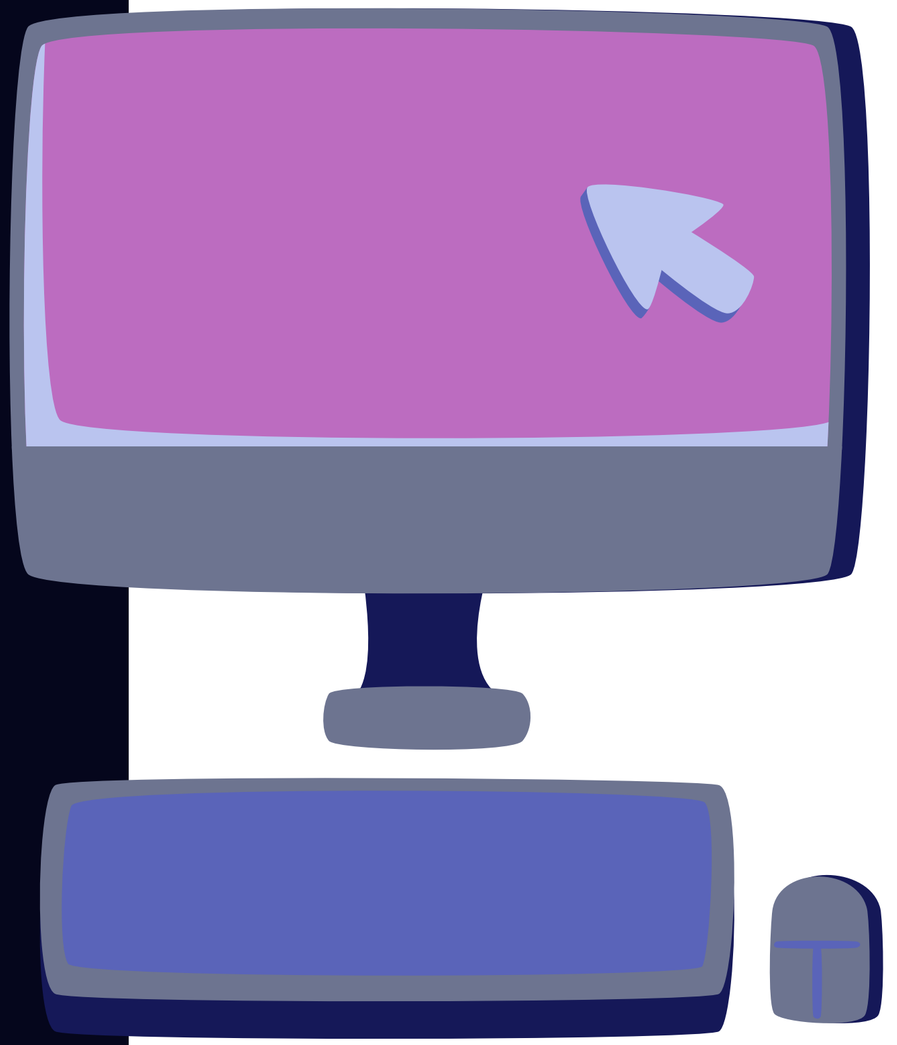
ÍNDICE



1. Introducción

Decorator es un patrón de diseño estructural que te permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

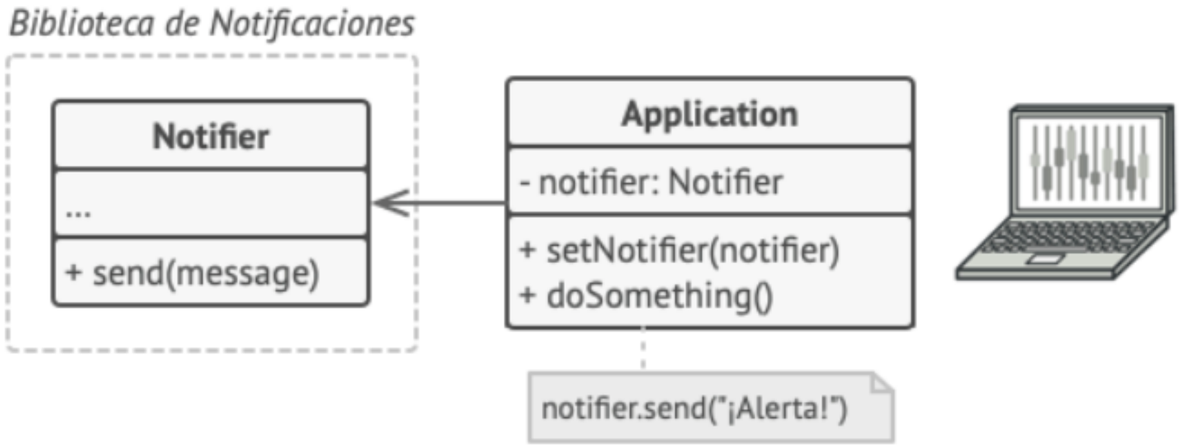




2. Problemas que soluciona

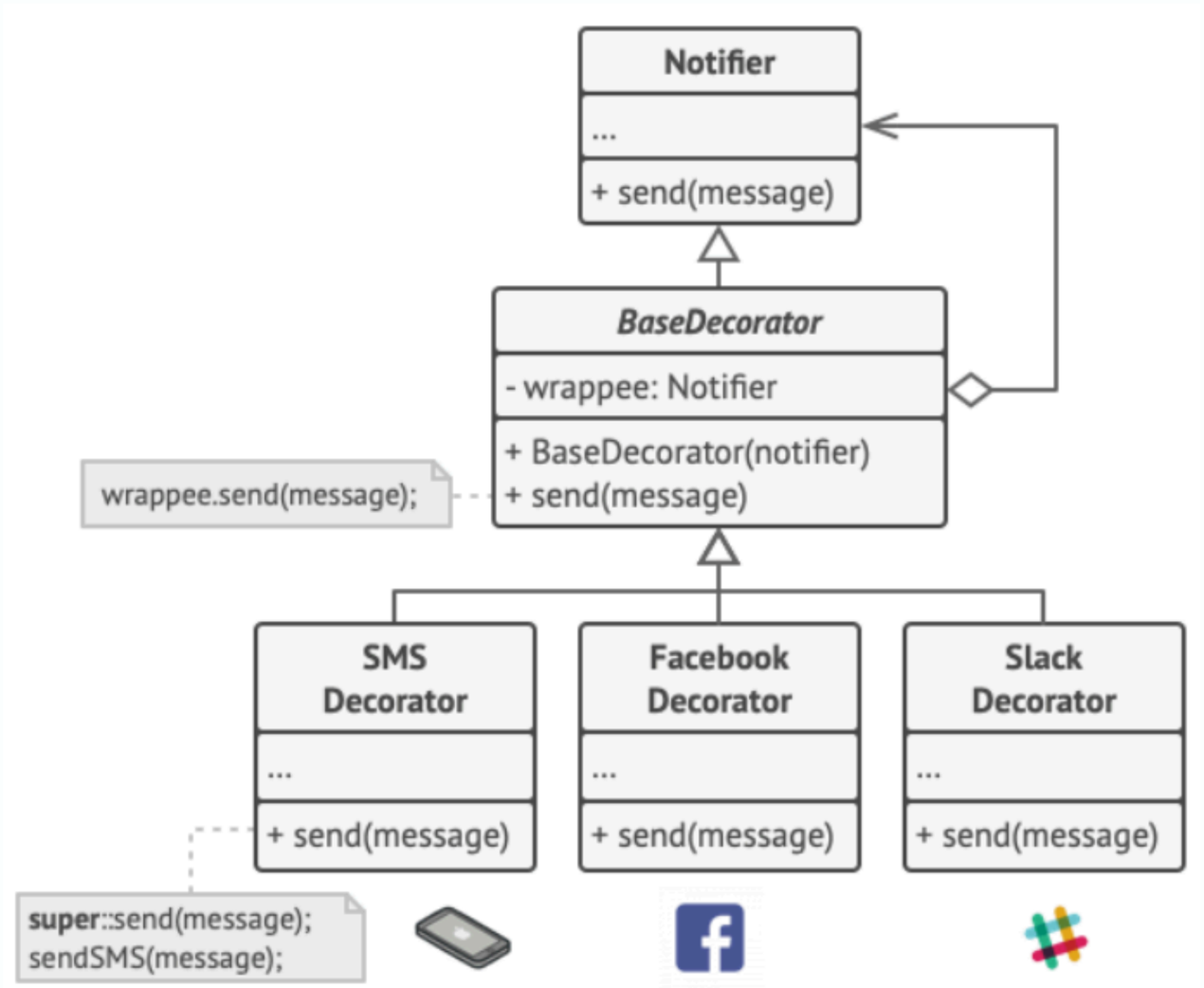
Problema

Originalmente, nuestra biblioteca de notificaciones solo permitía un tipo de notificación, el correo electrónico. Sin embargo, los usuarios deseaban recibir notificaciones a través de diversos canales, como SMS, Facebook y Slack. La solución inicial fue crear subclases para cada tipo de notificación, pero esto llevó a una explosión combinatoria de subclases y dificultó la combinación de múltiples tipos de notificación al mismo tiempo.



Solución

Para abordar este problema, aplicamos el patrón Decorator. En lugar de crear subclases para cada combinación de tipos de notificación, utilizamos decoradores para agregar funcionalidades de notificación adicionales de manera dinámica. Cada tipo de notificación se implementó como un decorador que se puede apilar sobre el objeto notificador base. Esto permitió a los clientes configurar fácilmente pilas de decoradores que satisfacen sus necesidades específicas de notificación, sin necesidad de crear nuevas subclases para cada combinación posible. Además, al utilizar decoradores, mantenemos la flexibilidad para agregar nuevos tipos de notificación en el futuro sin necesidad de modificar la clase base.



3. VENTAJAS Y DESVENTAJAS

VENTAJAS

- Alto grado de flexibilidad.
- Ampliación de las funciones de la clase sin herencia.
- Código de programa legible.
- Funcionalidades optimizadas para los recrusos.

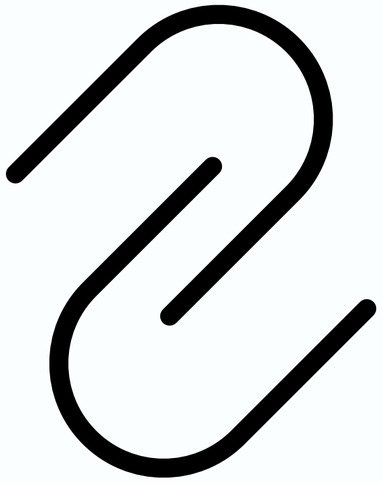
DESVENTAJAS

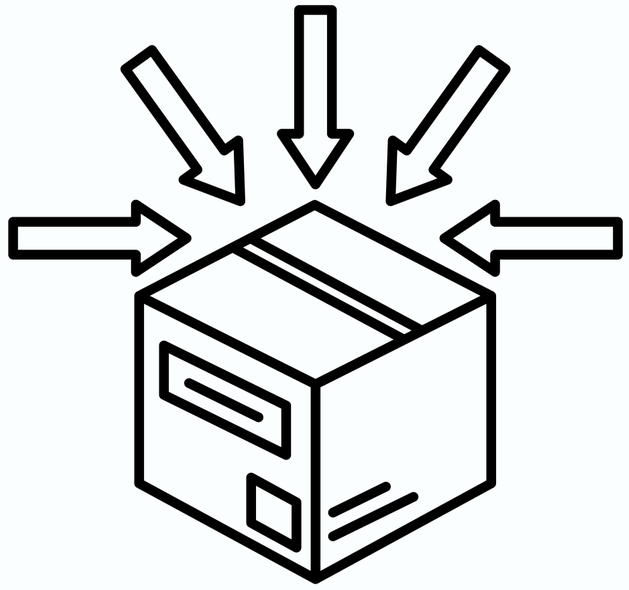
- Alta complejidad de software.
- De difícil comprensión al principio.
- Alto número de objetos.
- Proceso de depuración difícil.

4. Relación con otros patrones

Singleton

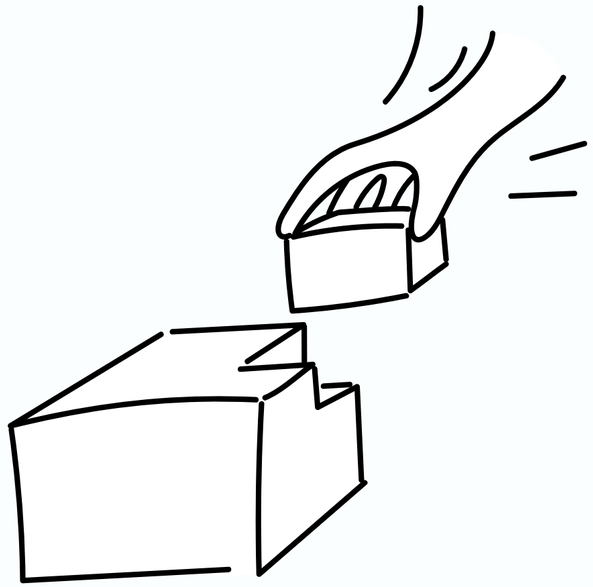
Aunque el Singleton se centra en garantizar que solo haya una instancia de una clase, puede combinarse con el patrón Decorator para asegurar que solo haya una instancia decorada de un objeto. Esto puede ser útil cuando se desea agregar funcionalidades adicionales a una instancia única de un objeto.





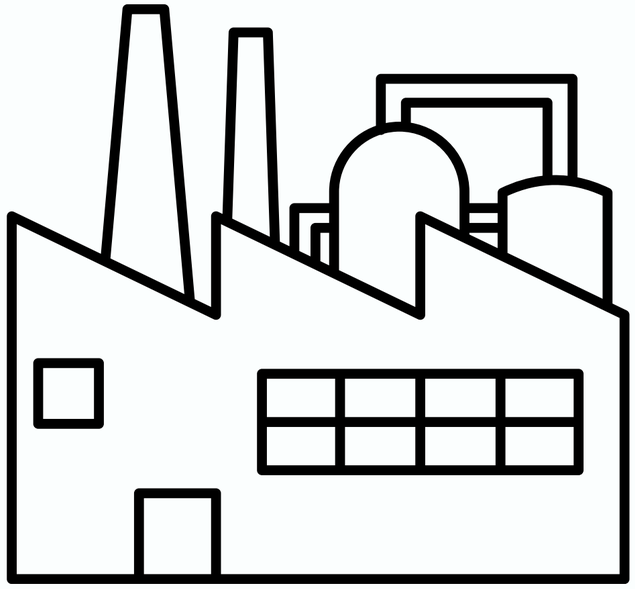
Factory Method

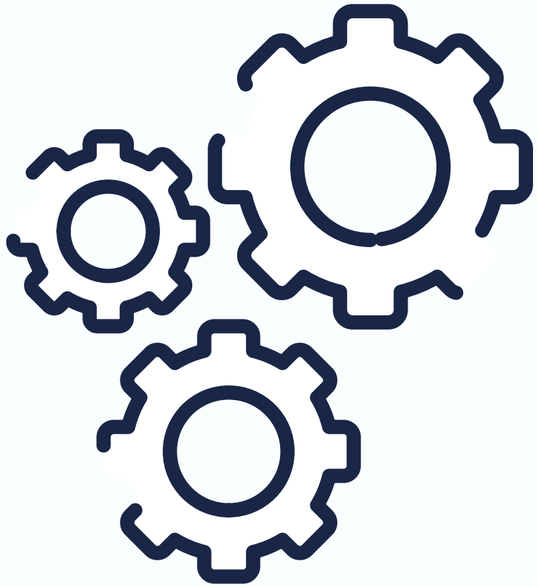
Al igual que el patrón Factory Method, el Decorator también utiliza la creación de objetos, pero se centra en añadir funcionalidades adicionales dinámicamente a un objeto existente, mientras que el Factory Method se centra en la creación de diferentes tipos de objetos.



Composite

El patrón Composite y el Decorator tienen en común la idea de tratar a los objetos de manera similar. Mientras que el Composite se centra en tratar a las colecciones de objetos de manera uniforme junto con los objetos individuales, el Decorator se centra en añadir funcionalidades adicionales a objetos individuales sin afectar a otros objetos.





5. Ejemplos de implemetncación {

Este código crea una bebida básica (un café) y luego añade decoradores para agregar opciones adicionales como leche y azúcar. Cada decorador modifica la descripción y el costo de la bebida base. Esto ilustra cómo el patrón Decorator permite añadir funcionalidades adicionales de manera dinámica a un objeto existente.

VER EN GITHUB

}

6. FUENTES DE INFORMACIÓN

Refactoring.guru

ionos.es

<!--Decorator-->

Gracias {

<Por="Iván Caro"/>

}