**DECORATOR PATTERN**

**LOS PRINCIPIOS DE DISEÑO UTILIZADOS EN ESTE PATRÓN SON:**

* Una clase debe estar abierta a la extensión, pero cerrada a la modificación.

Esto significa que debemos poder agregar funcionalidad a nuestras clases, pero la clase principal no debe ser modificada para cumplir con el objetivo.

Lo que nos permite poder agregar funcionalidad de forma sencilla sin afectar más clases, además de facilitar el mantenimiento del sistema.

**Los posibles casos de uso de este patrón serian.**

* Hay una necesidad de extender la funcionalidad de una clase, pero no hay razones para extenderlo a través de la herencia.
* Se quiere agregar o quitar dinámicamente la funcionalidad de un objeto.

**BENEFICIOS:**

* Puede extender el comportamiento de un objeto sin crear una nueva subclase.
* Puede agregar o quitar responsabilidades de un objeto en tiempo de ejecución.
* Puede combinar varios comportamientos envolviendo un objeto en múltiples decoradores.
* Principio de responsabilidad única.
* Puede dividir una clase monolítica que implementa muchas posibles variantes de comportamiento en varias clases más pequeñas.

**DESVENTAJAS:**

* Los decoradores pueden dar lugar a muchos objetos pequeños en nuestro diseño y el uso excesivo puede terminar en complejidad no deseada.
* Los decoradores pueden complicar el proceso de creación de instancias del componente porque no solo tiene que crear una instancia del componente, sino que se envuelve en varios decoradores.
* Es difícil implementar un decorador de tal manera que su comportamiento no dependa del orden en la pila de decoradores.