Practica 3: Patrones de diseño ‘Decorator’ y ‘Template’

Parte Teórica

Menciona los principios de diseño esenciales de los patrones Decorator y Adapter.
Menciona una desventaja de cada patrón.

Decorator: Dotar de funcionalidades dinámicamente a objetos. Es además una alternativa a las herencias de subclases engorrosas.
Los "decoradores" del objeto pueden crecer todo lo que se necesite hasta llenar las necesidades de nuestro sistema.

Adapter: Convierte la interfaz de una clase ya definida, a una que se adapta a la interfaz que el cliente necesita para seguir su funcionamiento original, permitiendo así que trabajen de forma colaborativa siendo que en un principio eran incompatibles.

DESVENTAJAS.

Decorator: Uno de las desventajas que llegué a encontrar mientras trabajaba fue que, el patrón nos da un diseño de clases de extensiones escasas, pero en grandes cantidades. Haciendo más complicada la depuración.
Ahora bien, hay que destacar que un decorador y sus componentes no son idénticos, al momento de referirnos a ellos como objetos. Aunque para el programador o el usuario eso puede no ser (tan) evidente.

Adapter: Añade una complejidad que algunos podrían ver innecesaria para el proyecto, perdiendo cierta flexibilidad porque el adaptador no puede trabajar con una clase y sus hijos a la vez, y causando dificultades al momento de redefinir el comportamiento de la clase adaptada.

Notas

Existen en la práctica tal vez unos comentarios que quizá podrían entrar dentro de llamarse paja, pero tanto por ver el código más homogéneo, como por ciertos detalles que seguramente en el futuro pueda llegar a olvidar. Cosas como constructores o variables auxiliares para un uso específico, preferí dejarlos.
Omití el comentario de algunos métodos dónde su comentario sería igual a la de la clase abstracta. La ejecución es solo con el main de la clase menu, y solo necesitas meter números para navegar entre los menús