Universidad de los Andes

DISEÑO Y PROGRAMACIÓN O.O. - TALLER # 5 PATRONES

Nombre: Andrés Felipe Alfonso Gamba

Código: 202210412

Carreras: Administración de Empresas e Ingeniería de Sistemas

Desarrollo taller

- **Información General del Patrón:** Me parece apropiado antes de empezar describir el patrón de GOF que se trabajará en el siguiente documento, el cual es el patrón *Decorador* (*Decorator* en inglés). El Decorador es un patrón de diseño estructural que permite agregar funcionalidades a objetos colocándolos dentro de otros objetos encapsuladores. Este patrón está estructurado en 5 partes:
  1. **Componente:** Declara la interfaz común.
  2. **Componente Concreto:** Es una clase de objetos relacionados que definen el comportamiento básico que los decoradores pueden alterar.
  3. **Decoradora Base:** Esta clase referencia al objeto envuelto. Esto se realiza en un campo cuyo tipo debe declararse como la interfaz del componente, esto con el fin de que tenga la capacidad de contener los componentes concretos y decoradores.
  4. **Decoradores Concretos:** Estos decoradores definen funcionalidades adicionales que sobrescriben métodos de la clase decoradora base y ejecutan su comportamiento, bien sea antes o después de invocar al método padre.
  5. **Cliente:** La clase Cliente envuelve componentes en varias capas de decoradores, eso con la condición de que trabajen siempre con todos los objetos a través de la interfaz del componente.
- **Información General del Proyecto:** Inicialmente aclaro que la forma de encontrar el proyecto fue mediante la búsqueda de "*proyectos que utilicen patrones de gof github*" en el buscador de Google, con lo que encontré un repositorio considerablemente grande en el cuál me llamó la atención el "*Decorador*". (https://github.com/sidlors/patrones/tree/master/12_Decorador).
  Aclarando eso, lo que entendí del proyecto es que sirve para adicionar objetos a un objeto más grande, que en este caso son ingredientes (Lechuga, Queso y Tomate) a una hamburguesa, que ya cuenta con Carne y Pan.
  Su diseño está distribuido en 6 clases, las cuales 4 se encargan de tener la hamburguesa y las adiciones, otra se encarga de agregar las adiciones deseadas y la última se encarga de la comunicación con el usuario. Esta

última clase es muy útil debido a que simplifica el trabajo para el usuario, pues simplemente con un par de teclas puede formar la hamburguesa con lo que más le gusta. Realmente es un código muy simple, por lo que consideraría que lo más complicado es verificar que todos los elementos sean añadidos de forma correcta.

- **Información del patrón aplicado al proyecto:** Como se mencionó en el inicio, hay varios componentes para el Decorador. En este caso la clase Cliente se encuentra en la clase *PruebaDecorador*, que se encarga de comunicarse con el usuario. El grupo de Componentes Concretos es el grupo de ingredientes adiciones que se encuentran disponibles, mientras que el Decorador Concreto es el *DecoradorHamburguesa*, y el Decorador Base es la clase *Hamburguesa*. Las razones de las asignaciones se explican al inicio.
- **Ventajas y Desventajas de la utilización del patrón en ese punto del proyecto:** La principal ventaja de la implementación del patrón en este caso es la posibilidad de agregar funcionalidades a clases, como los ingredientes, sin alterar el código fuente de la hamburguesa. Además de ello, se encuentran la reutilización del código en caso de ser necesario. Por otro lado, se consideraría como "desventaja" el uso de varias clases que finalmente terminarán sumando a la memoria final del proyecto, aunque esto no será muy relevante debido a que no será en gran medida, y de igual forma es más grande su ventaja que desventaja.
- **Otra solución:** Aunque sigo considerando la opción del patrón decorador como la más útil, la más simple sería ignorar la división de clases y dejar todo en una misma clase, donde si bien se tiene todo el código dentro de un mismo lugar, en caso de querer realizar un cambio no será tan sencillo como al aplicar el patrón. Soluciones adicionales con otros patrones no son consideradas como soluciones debido a la poca similitud que tienen como el Decorador.

- **Bibliografía:**
  - Anónimo, (S.F.) *Decorator*. Refactoring Guru. Tomado de https://refactoring.guru/es/design-patterns/decorator