Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

**UNIVERSIDAD DEL CAUCA** 

**FACULTAD DE INGENIERIA ELECTRÓNICA Y TELECOMUNICACIONES PROGRAMA DE INGENIERÍA DE SISTEMAS**

**PRACTICA DE LABORATORIO No. 7**

**Definición del proyecto de trabajo**

**OBJETIVOS**: Comprender e implementar el patrón estructural Decorator (Decorador).

**ACTIVIDADES:**

1. Desarrolle la siguiente plantilla para el patrón Decorador:

|  |  |
| --- | --- |
| Patrón creacional: **Decorador** | |
| Intención | Permite adjuntar responsabilidades adicionales a un objeto dinámicamente. Los decoradores ofrecen una alternativa flexible a la subclasificación para extender la funcionalidad. |
| Problema que  soluciona | Cuando se necesita agregar comportamiento o estado a objetos individuales en tiempo de ejecución, y la herencia no es factible porque es estática y se aplica a toda una clase. |
| Solución propuesta | La solución consiste en encapsular el objeto original dentro de una interfaz de envoltura abstracta. Tanto los objetos decoradores como el objeto central heredan de esta interfaz abstracta. La composición recursiva se utiliza para permitir un número ilimitado de "capas" de decoradores que se pueden agregar a cada objeto central. Esto permite que las responsabilidades se agreguen a un objeto sin modificar la interfaz del objeto. |
| Diagrama de clases |  |
| Diagrama de  secuencia |  |
| Participantes | * Cliente (Client): Quien usa el objeto central y los decoradores. * Componente (Component): La interfaz común para el objeto central y los decoradores. * Componente Concreto (ConcreteComponent): El objeto central al que se pueden agregar decoradores. * Decorador (Decorator): La clase base para los decoradores, que también implementa la interfaz Component. * Decorador Concreto (ConcreteDecorator): Implementaciones concretas de decoradores que añaden funcionalidad adicional. |
| Aplicabilidad | * Cuando se necesita agregar funcionalidad adicional a objetos individuales de forma dinámica y flexible. * Cuando se quiere evitar la subclasificación excesiva de clases para agregar nuevas funcionalidades. * Cuando se desea mantener la interfaz del objeto constante para el cliente mientras se agrega responsabilidad adicional. |
| Consecuencias | * Permite una extensión flexible de la funcionalidad de objetos. * Permite un diseño modular y de fácil mantenimiento. * Permite agregar o quitar decoradores sin afectar a otros componentes del sistema. * Sin embargo, puede resultar en una cadena de decoradores larga y compleja si se abusa de él. |