Decorator

Propósito

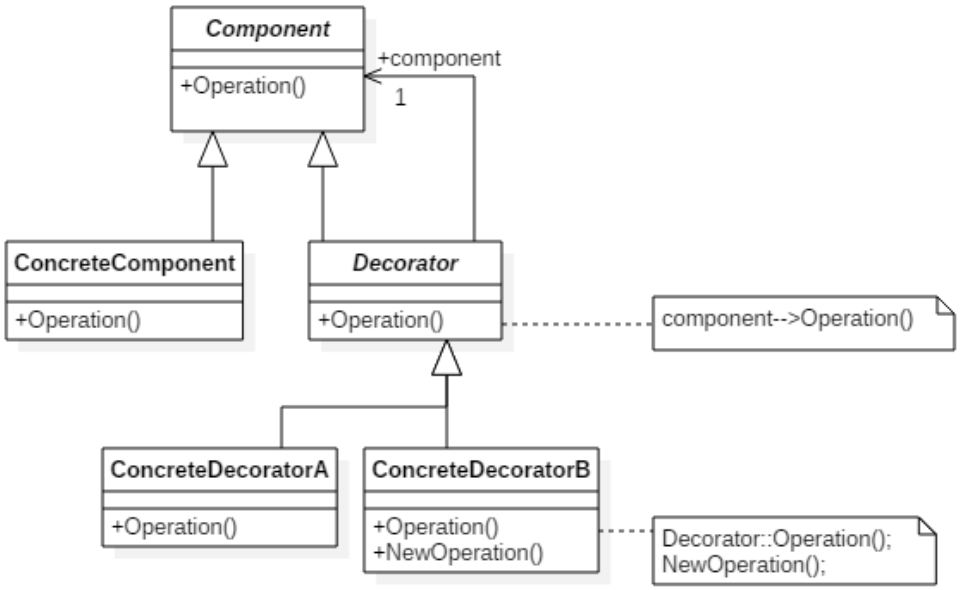
Asigna dinámicamente responsabilidades adicionales a un objeto. Los decoradores proporcionan una alternativa flexible a la subclasificación para extender la funcionalidad.

Aplicabilidad

Utilice el Decorator:

- Para agregar dinámicamente y de forma transparente responsabilidades a un objeto, sin afectar a otros objetos
- Para poder retirar responsabilidades
- Cuando extender la funcionalidad mediante la subclasificación no es práctico. En ocasiones se definen un gran número de extensiones independientes que podría producir una explosión de subclases para poder soportar todas las combinaciones.

Estructura



Participantes

- Component: define la interface para los objetos a los que se puede añadir responsabilidades dinámicamente
- ConcreteComponent: define un objeto al que se puede agregar responsabilidades
- Decorator: mantiene una referencia a un objeto Component y define una interface que es conforme con la interface de Component
- ConcreteDecorator: Añade responsabilidades al componente

Colaboraciones

El Decorator reenvía las peticiones al objeto Component. Opcionalmente puede realizar operaciones adicionales antes o después de reenviar la petición.

Consecuencias

El patrón Decorator tiene al menos dos beneficios clave y dos cargas:

1. Más flexibilidad que la herencia estática. El patrón Decorator proporciona una manera más flexible de agregar responsabilidades a objetos que la que se puede obtener mediante la herencia estática (posiblemente múltiple). Con decoradores, las responsabilidades se pueden agregar o quitar dinámicamente simplemente asignándolas o desasignándolas. Por el contrario, la herencia exige crear una nueva clase para cada nueva responsabilidad.
2. Evitan tener clases cargadas de características en la parte alta de la jerarquía de clases. El Decorator ofrece una solución pago-por-uso al agregar responsabilidades. En lugar de tratar de soportar todas las características (atributos) imaginables in una clase compleja y personalizable, puedes optar por definir una clase sencilla y agregar funcionalidad de forma incremental mediante objetos Decorators. La funcionalidad se puede componer a partir de piezas simples. De este modo, una aplicación no tiene necesidad de "pagar" por características que no utiliza. Además, también es fácil definir nuevas clases de Decorators que sean independientes de los objetos que decoran, incluso para extensiones que no habíamos previsto. Extender una clase compleja tiene a exponer detalles que no están relacionados con la funcionalidad que se agrega.
3. Los componentes y los decoradores no son idénticos. Un decorador actúa como un recinto transparente. Desde un punto de vista de identidad de objetos, un componente decorado no es idéntico al componente. Con lo cual, hay que tener cuidado al utilizar la identidad de objetos en presencia de decoradores.
4. Muchos objetos pequeños. Un diseño que utilice el Decorator produce un sistema que se compone de muchos objetos pequeños y parecidos. Los objetos difieren en la forma en que se conectan, más que en su clase o en el valor de las variables. Aunque estos sistemas son fáciles de personalizar por aquellos que los entienden, pueden resultar complejos de aprender y depurar.