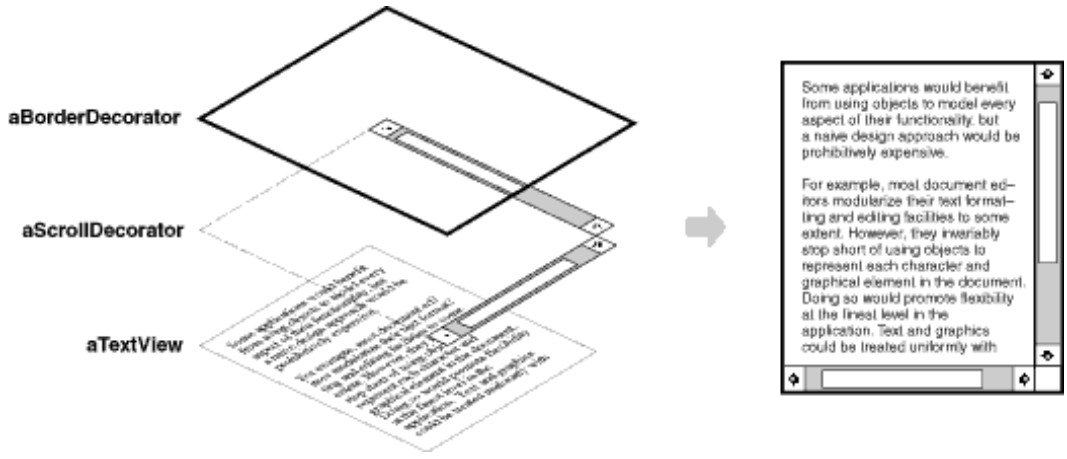
Decorador (Decorator)

- *Patrón Estructural*
- *Propósito*

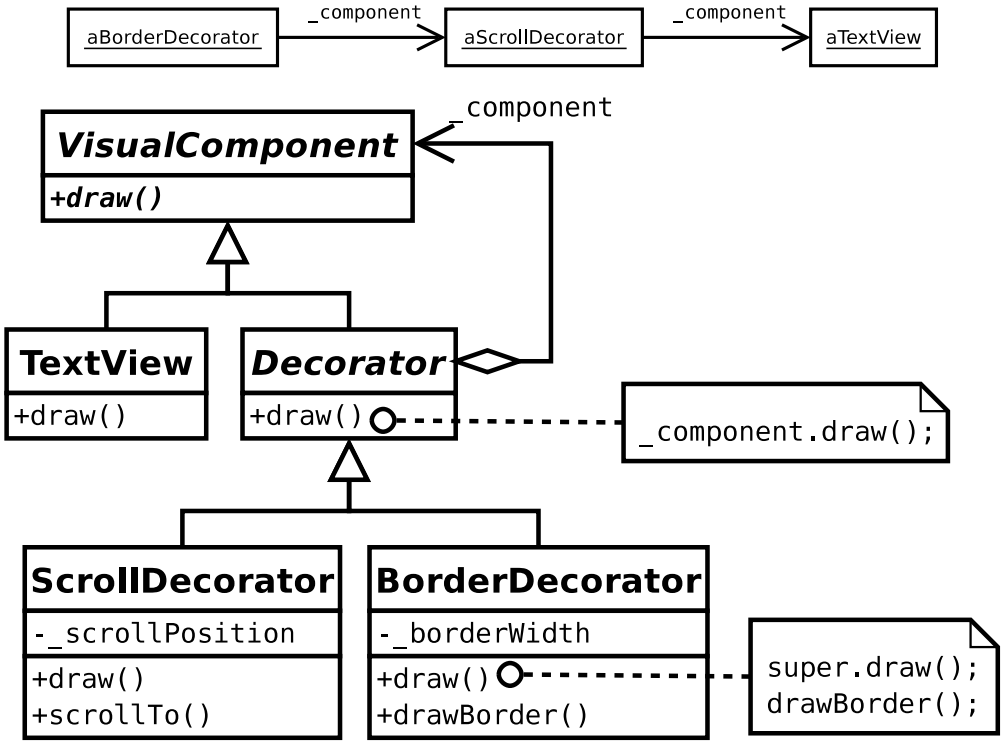
  Añadir responsabilidades a un objeto dinámicamente, proporcionando una alternativa flexible a la extensión de una clase

- *Motivación*

  - Deseo de añadir responsabilidades a objetos individuales, no a una clase
  - Ejemplo: Interfaz gráfica que permite añadir bordes o barras de desplazamiento a cualquier componente
  - Solución: Utilizar herencia para extender las responsabilidades de la clase
  - Problema: Inflexible! (estático)
  - Solución: Encapsular dentro de otro objeto que añade las nuevas responsabilidades (*decorador*)



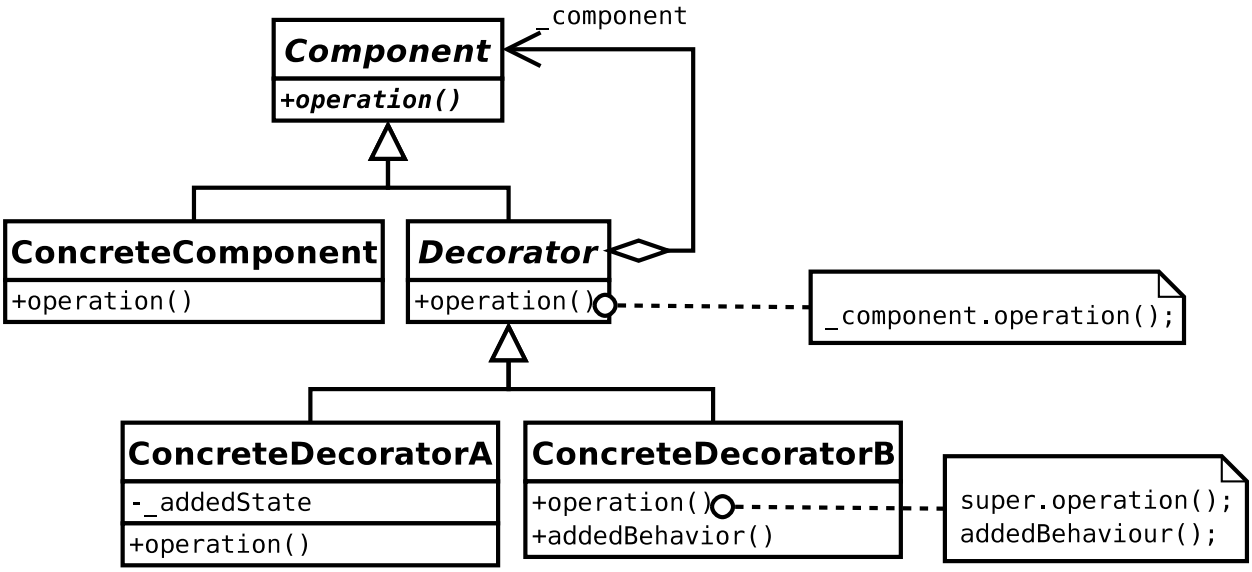
- *Motivación* (cont.)



  - La superclase decoradora implementa la interfaz del componente visual, redirigiendo los métodos al componente que encapsula
  - Las subclases decoradoras refinan los métodos del componente, añadiendo responsabilidades
  - Los clientes de los componentes visuales no hacen distinción entre los componentes decorados y sin decorar

- *Aplicabilidad*
  - Añadir responsabilidades a objetos de manera dinámica y transparente
  - Para suprimir responsabilidades de los objetos
  - Cuando no es práctico el uso de la herencia

- *Estructura*



- *Participantes*

  - Componente (Component)

    Define la interfaz para los objetos que pueden tener responsabilidades añadidas

  - Componente Concreto (ConcreteComponent)

    Define un objeto al cual se le pueden agregar responsabilidades adicionales

  - Decorador (Decorator)

    Mantiene referencia al componente asociado

    Implementa la interfaz de la superclase Componente delegando en el componente asociado

  - Decorador Concreto (ConcreteDecorator)

    Añade responsabilidades al componente (refinamiento)

- *Colaboraciones*
  - El decorador redirige las peticiones al componente asociado
  - Opcionalmente puede realizar tareas adicionales antes y después de redirigir la petición
- *Consecuencias*
  - Más flexible que la herencia
    - Configurable en tiempo de ejecución
    - Evita herencia múltiple
  - Evita la aparición de clases con muchas responsabilidades en las clases superiores de la jerarquía
    - Incorporación incremental de responsabilidades
    - No es necesario pagar por funciones no requeridas
  - Problemas con la identidad de objetos
  - Gran número de objetos pequeños
- *Implementación*
  - Decorador debe cumplir la interfaz Componente
  - Omisión de la clase abstracta Decorador
  - Mantener la clase Componente ligera
  - Cambiar la *piel* del objeto (Decorador) vs. cambiar las *tripas* (Estrategia)