# *Patrones de Diseño: Patrones Estructurales. Tema 4-5: Decorator*

# Descripción del patrón

- **Nombre:**
  - Decorador
  - También conocido como Wrapper (envoltorio)
- **Propiedades:**
  - Tipo: estructural
  - Nivel: objeto, componente
- **Objetivo o Propósito:**
  - Añadir nuevas responsabilidades dinámicamente a un objeto sin modificar su apariencia externa o su función, es una alternativa a crear demasiadas subclases por herencia.

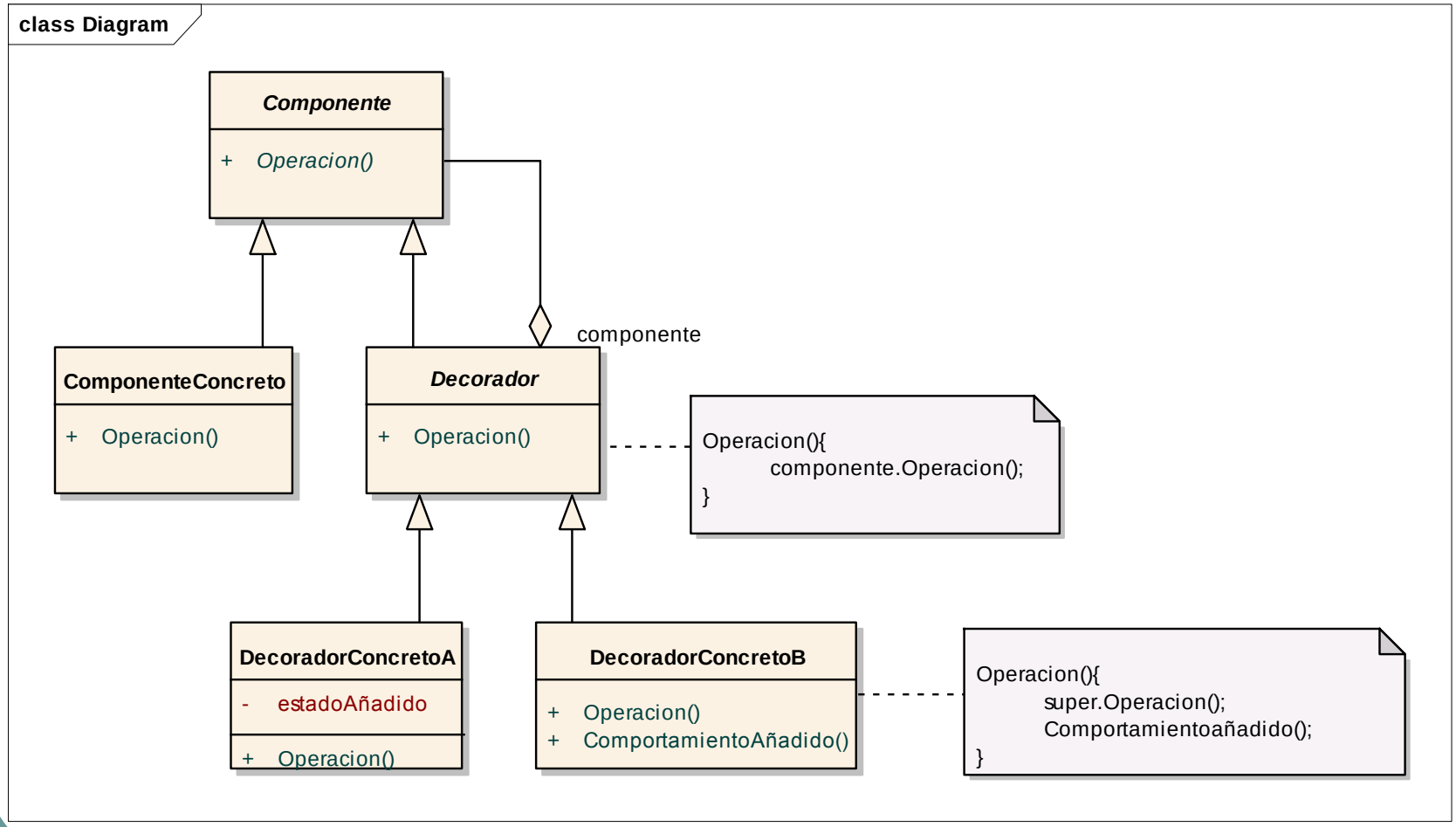


# Aplicabilidad

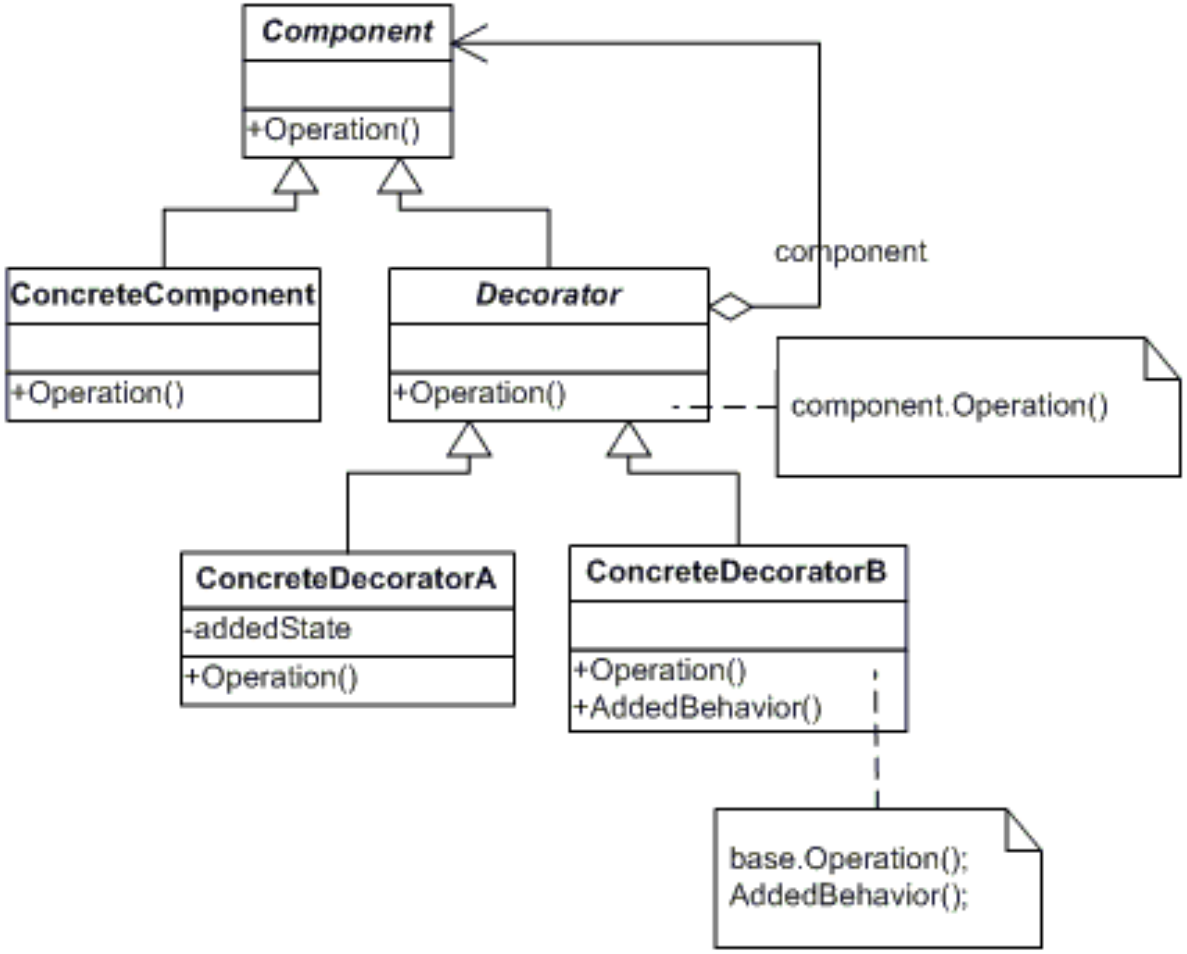
- Use el patrón Decorator cuando:
  - Para añadir funcionalidad a una clase sin las restricciones que implica la utilización de la herencia.
  - Cuando se quiera añadir funcionalidad a una clase de forma dinámica en tiempo de ejecución y que sea transparente a los usuarios.
  - Haya características que varíen independientemente, que deban ser aplicadas de forma dinámica y que se puedan combinar arbitrariamente sobre un componente.



# Estructura


class Diagram
Componente
+ Operacion()
componente
ComponenteConcreto
+ Operacion()
Decorador
+ Operacion()
Operacion(){
componente.Operacion();
}
DecoradorConcretoA
- estadoAñadido
+ Operacion()
DecoradorConcretoB
+ Operacion()
+ ComportamientoAñadido()
Operacion(){
super.Operacion();
Comportamientoañadido();
}

# Estructura


Component
+Operation()
component
ConcreteComponent
+Operation()
Decorator
+Operation()
component.Operation()
ConcreteDecoratorA
-addedState
+Operation()
ConcreteDecoratorB
+Operation()
+AddedBehavior()
base.Operation();
AddedBehavior();

# Estructura. Participantes

- **Componente**: Interfaz común a todas las clases susceptibles de ser ampliadas con el Decorator. Contiene el comportamiento genérico.
- **ComponenteConcreto**: Son las clases cuya funcionalidad se puede extender y en los que se delega en última instancia para realizar las operaciones propias de la clase.
- **Decorador**: Clase abstracta que declara la estructura común a todos los decoradores y declara (o implementa, según) la responsabilidad de mantener una referencia al objeto que se extiende. Es posible que sobrecargue todos los métodos de la clase Componente con llamadas al componente concreto, de forma que aquellos métodos cuya funcionalidad no se extiende simplemente llaman a los originales.
- **DecoradorConcreto**: Son clases concretas que heredan de Decorador e implementan las extensiones de funcionalidad de la clase original ComponenteConcreto.



# Estructura. Variaciones

- **Variaciones del patrón:**
- **Decorador único.** Algunas implementaciones no utilizan una clase Decorador abstracta ya que solo hay una posible variante para el componente.
- **Decoradores redefinidos.** Se pueden redefinir nuevos decoradores redefiniendo los anteriores, lo cual modificará algunas partes del comportamiento del componente.
- Si solamente hay una clase ComponenteConcreto y ninguna interfaz Componente, entonces la clase Decorador es normalmente una subclase de la clase ComponenteConcreto.



# Consecuencias

- Ofrece más flexibilidad que la herencia estática. Con los decoradores se pueden añadir y eliminar responsabilidades en tiempo de ejecución. Por el contrario con la herencia hay que crear una nueva clase para cada nueva responsabilidad.
- Se consiguen componentes pequeños muy parecidos.
- Se reduce el número de clases y el árbol de herencia de clases. Teniendo menos clases se simplifica el diseño y la implementación de los programas.
- Una dificultad asociada al patrón Decorator es que un objeto decorador no es un objeto componente, por lo que se pierde la identidad del objeto. Los objetos componente se esconden detrás de los objetos decorador.


# Patrones relacionados

- **Adapter.** El patrón Adapter está pensado para modificar la interfaz manteniendo la misma funcionalidad, mientras que Decorator no cambia la interfaz sino la funcionalidad.
- **Composite.** Podemos ver un decorador como una versión simple de un Composite que solo tiene un componente.
- **Strategy.** La acción del decorador es modificar o ampliar la funcionalidad externa del objeto, el patrón Strategy nos permite modificar el comportamiento interno del objeto.
- **Template Method.** El patrón Template Method es otra alternativa al patrón Decorator que permite variar el comportamiento en medio de una llamada a un método en lugar de antes o después.

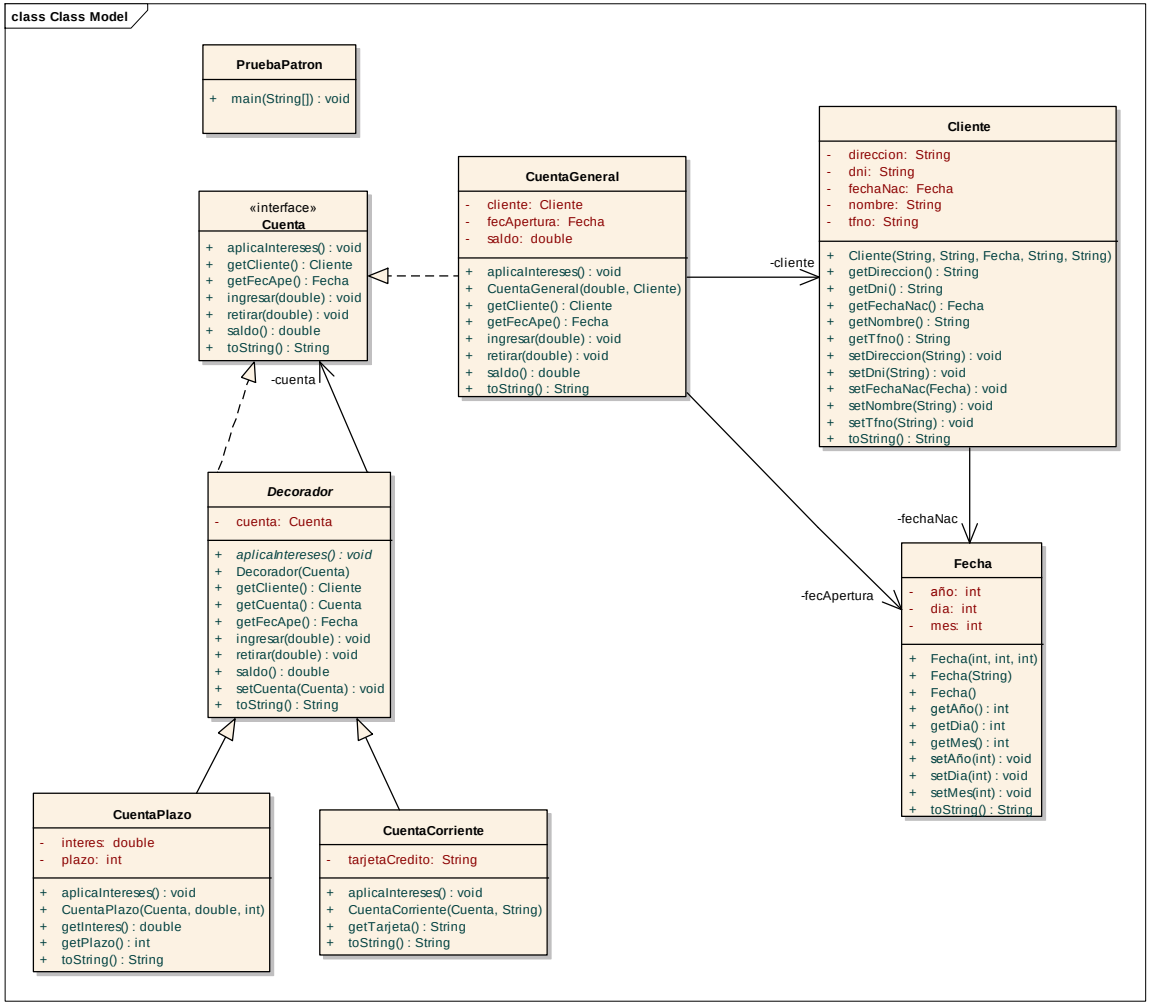


# Código de ejemplo

CUENTAS BANCARIAS



# Código de ejemplo



# Código de ejemplo

- Identificamos a continuación los elementos del patrón:
  - Componente: Cuenta.
  - ComponenteConcreto: CuentaGeneral.
  - Decorador: Decorador.
  - DecoradorConcreto: CuentaCorriente, CuentaPlazo.