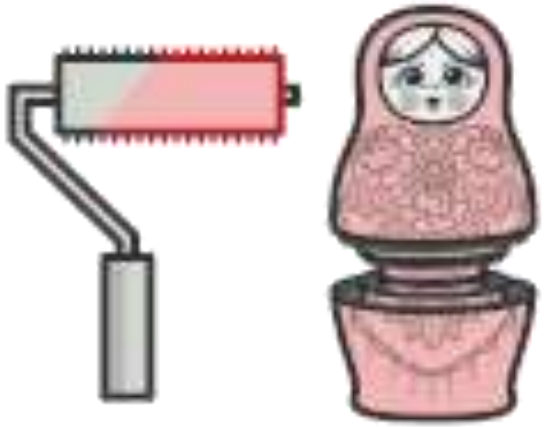


Patrón Decorador

Julio Eduardo Perez Villarreal

Propósito

- Permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.
- Permite añadir nuevas características a los objetos en tiempo de ejecución.
- Es una alternativa más flexible que la herencia.
- Extender dicha funcionalidad sin necesidad de crear clases más complejas.

Aspectos básicos

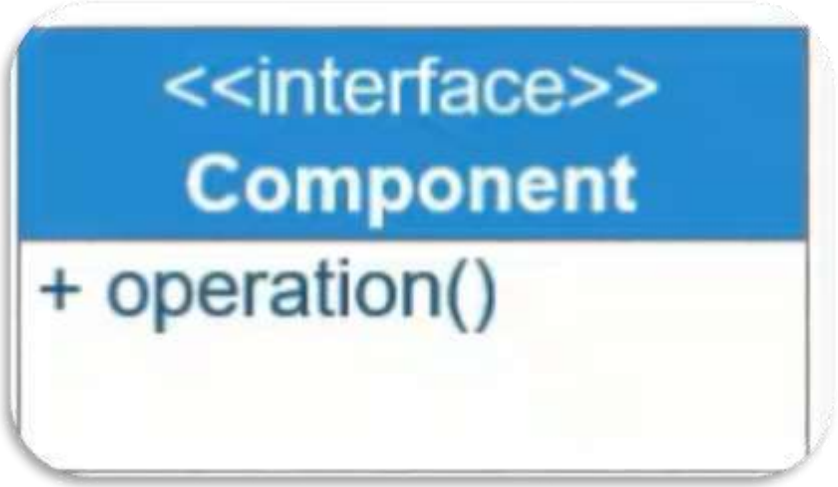
- Decorador o Wrapper

- Patrón de estructura

- Patrón de objetos

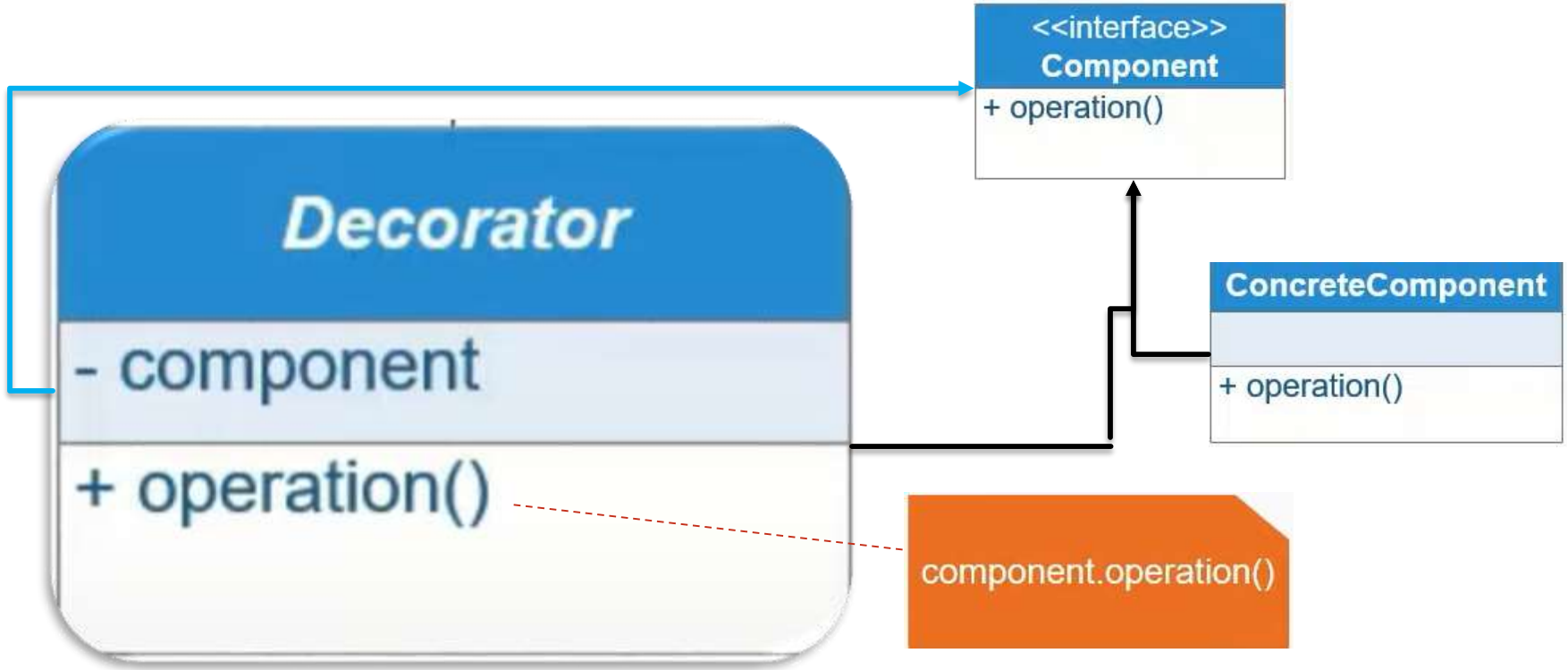
- Participantes
  - Componente
  - Componente Concreto
  - Decorador
  - Decorador concreto

La estructura

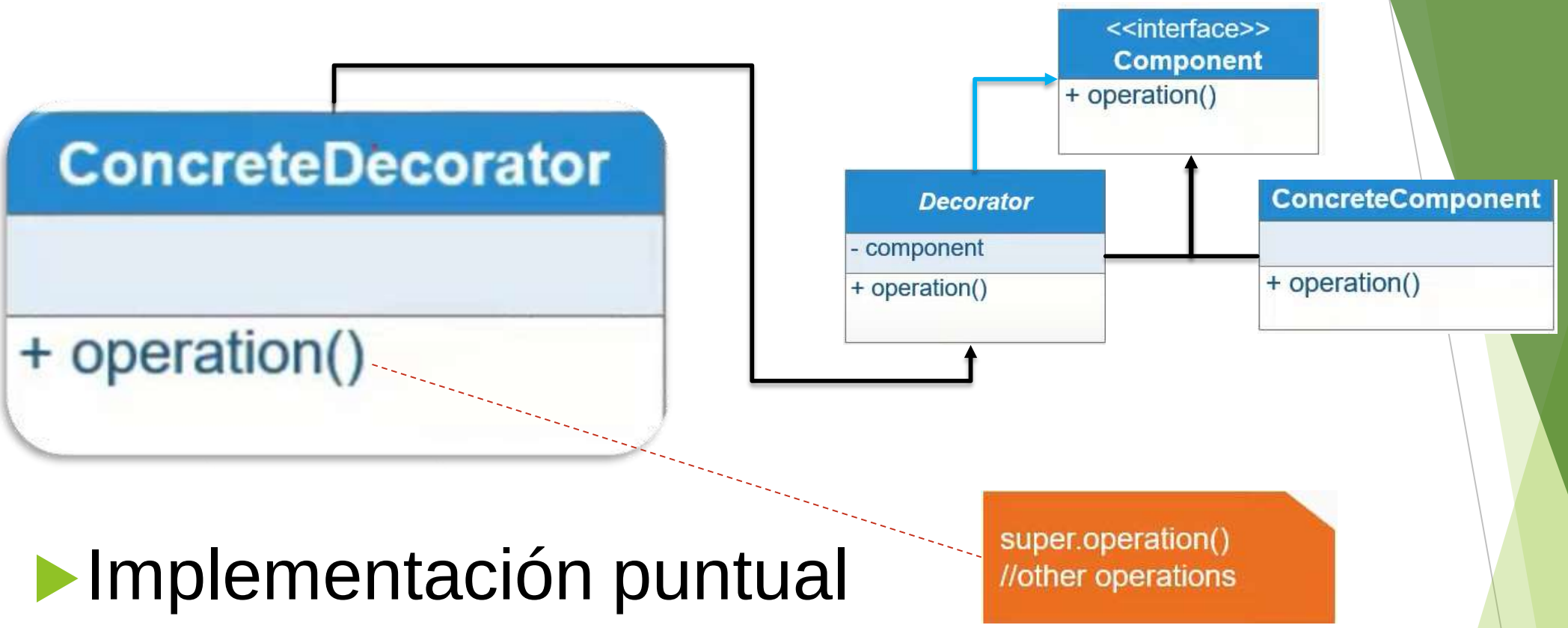


- Objeto que vamos a envolver como el objeto que envuelve
- Hace algo

- Clase que va ser envuelta
- Funcionalidad muy puntual
- Añadir nuevas funcionalidades dinámicamente



- Es un componente cualquiera en el sistema
- Abstracto
- Referencia a componente



- Implementación puntual
- La lógica

En que casos utilizarlos ?

- Añadir responsabilidades a objetos de forma dinámica

- Sin modificar el objeto que se esta decorando
  - Mantenibilidad.
  - Esta en otra librería o clase.
  - Código más organizado.
  - No contaminar con lógica que no le corresponde.

- Añadir responsabilidades temporales
  - No requiera ser decorado todo el tiempo
  - En una actividad bien puntual

- Extender una clase mediante herencia no es practico o viable

Ventajas

- Puedes extender el comportamiento de un objeto sin crear una nueva subclase.
- Puedes añadir o eliminar responsabilidades de un objeto durante el tiempo de ejecución.
- Combinar varios comportamientos envolviendo un objeto con varios decoradores.

Desventajas

- Resulta difícil eliminar un decorador específico de la pila de decoradores
- Es difícil implementar un decorador de tal forma que su comportamiento no dependa del orden en la pila de decoradores.