# Composite

## Propósito

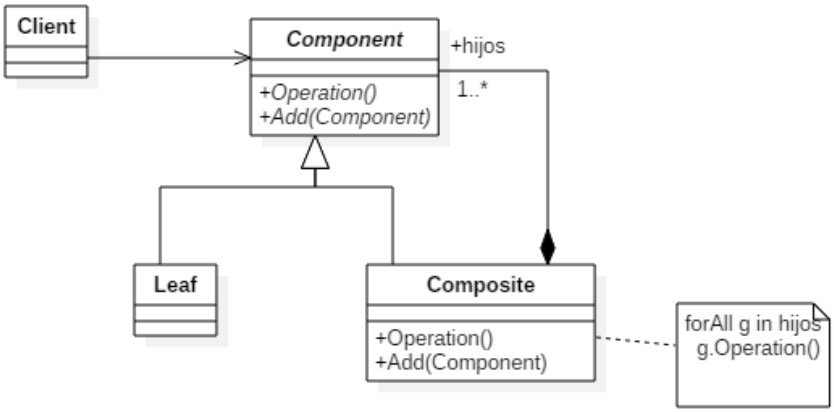
Compone objetos en una estructura de árbol para representar jerarquías todo-parte. El Composite permite que el cliente trate de manera uniforme tanto a objetos individuales como a objetos compuestos.

## Aplicabilidad

El Composite se utiliza cuando:

- Se desea representar jerarquías de objetos todo-parte
- Se desea que los clientes ignoren las diferencias entre objetos compuestos y simples. Los clientes tratan de manera uniforme a todos los objetos

## Estructura




## Participantes

- Component (componente):
  - Declara la interface de los objetos de la estructura compuesta
  - Implementa el comportamiento predeterminado de la interface común, según convenga
  - Declara una interface para acceder y gestionar los hijos
  - Opcionalmente define una interface para acceder al componente padre de la jerarquía recursiva, y la implementa si conviene
- Leaf (hoja):
  - Representa a los objetos hoja de la composición
  - Una hoja no tiene hijos
  - Define el comportamiento de los objetos individuales de la composición
- Client (cliente):
  - Manipula los objetos de la composición mediante la interface Component

## Colaboraciones

- Los clientes utilizan la interface Component para interactuar con los objetos de la estructura compuesta. Si el receptor de la petición es una hoja (Leaf), entonces la petición se maneja directamente. Si el receptor de la petición es un Composite, entonces normalmente se reenvía la petición a sus componentes hijo, realizando posiblemente alguna operación antes o después de reenviar la petición.

## Consecuencias

El patrón Composite:

- Define jerarquías de clases que consisten en objetos compuestos y objetos primitivos. Los objetos primitivos forman parte de objetos compuestos que, a su vez, pueden ser parte de otros objetos compuestos, y así recursivamente
- Simplifica a los clientes. Los clientes pueden tratar de manera uniforme tanto a las estructuras compuestas como a objetos individuales. Los clientes normalmente no conocen (y no deben conocer) si están tratando con un objeto hoja o con un objeto compuesto. Esto simplifica el código de los clientes porque evita que tengan que escribir funciones al estilo switch-case para determinar el tipo de clase de la composición
- Facilita agregar nuevos tipos de componentes. Una nueva clase hoja o componente funciona de manera automática con el código cliente que teníamos. En principio no hay que modificar el código del cliente si agregamos nuevos tipos de componente
- Puede hacer que el diseño resulte demasiado genérico. La desventaja que aparece como consecuencia de facilitar agregar componentes, es que hace difícil restringir los componentes de la estructura. En ocasiones se desea que un compuesto tenga solo ciertos tipos de componentes. Con el patrón Composite no se puede s recurrir al sistema de tipos para definir esas restricciones. Hay que recurrir a los chequeos en tiempo de ejecución.