Patrón Estructural Composite

Universidad del Cauca

Vigilada Mineducación

Laboratorio de Ingeniería de Software II

Practica de Laboratorio No. 7

Presentado por:

Jeferson Castaño Ossa

David Santiago Giron Muñoz

Profesor:

Ricardo Antonio Zambrano Segura

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Ingeniería de Sistemas

Popayán, Octubre de 2023

Patrón Estructural Composite

Intención:

La intención del patrón composite es brindar una estructura la cual simplifica la manipulación de objetos dentro de una jerarquía. Es decir, permite crear estructuras jerárquicas de objetos donde los clientes puede tratar tanto a los objetos individuales (hojas dentro del árbol) como a las composiciones de objetos de manera uniforme (de la misma manera), para de esta manera trabajar con jerarquías de objetos complejas de manera consistente y mucho más flexible y simplificado, permitiendo así también la composición recursiva.

Problema que soluciona:

El principal problema que intenta resolver el patrón composite es el tratar a objetos primitivos y contenedores de estos objetos de manera diferente dentro de una jerarquía que los involucra en un problema concreto. Esto debido a que el tener que consultar y distinguir constantemente el tipo de objeto con el que se está trabajando antes de realizar cierta operación hace la aplicación mucho más compleja, además de no resultar óptimo y deseable.

En gran medida debido a que la mayor parte del tiempo se quiere o se tratan de manera idéntica, compartiendo los mismos “comportamientos” e incluso habiendo la necesidad de realizar consultas que involucran todos los subobjetos dentro de un contenedor. Es decir, recorrer y operar toda la estructura donde la solución a la vista más sencilla es iterar todos y cada uno de estos objetos, pero como ya se mencionó previamente, se deben manejar y conocer en todo momento sus tipos y composiciones haciendo que sea una solución nada optima.

De esta manera, el patrón composite busca una manera efectiva de gestionar y trabajar de manera simplificada con composiciones recursivas, de manera que el cliente no tenga que hacer distinciones entre los tipos de objetos con los que está trabajando constantemente ya sean objetos individuales, o composiciones de objetos y otros contenedores. Brindando una recursión efectiva en la estructura jerárquica sin una lógica especial.

Solución propuesta:

La solución que propone el patrón composite es la creación de estructuras jerárquicas de objetos donde los objetos individuales y las composiciones de esos objetos se van tratar de manera uniforme. Proporciona una forma de crear estructuras de objetos complejas y anidadas, como árboles o estructuras de árbol.

- **Composición jerárquica**: Permite a los clientes (código que utiliza la estructura) tratar a los objetos individuales y a las composiciones de objetos de manera uniforme. Esto significa que los clientes pueden interactuar con un objeto individual o con una composición (que contiene varios objetos) de la misma manera.
- **Recursión**: El patrón Composite permite una estructura jerárquica que se puede recorrer de manera recursiva. Por lo que ayuda a realizar operaciones en profundidad en la estructura.
- **Flexibilidad**: Facilita la creación de objetos compuestos, lo que significa que se puede combinar objetos en una estructura compleja de manera flexible. Brinda la posibilidad de agregar o eliminar objetos y composiciones de objetos sin cambiar el código del cliente.
- **Simplifica el código del cliente**: Los clientes pueden interactuar con la estructura compuesta sin necesidad de conocer la diferencia entre objetos individuales y compuestos. Esto simplifica el código del cliente y lo hace más genérico.

Diagrama de clases:

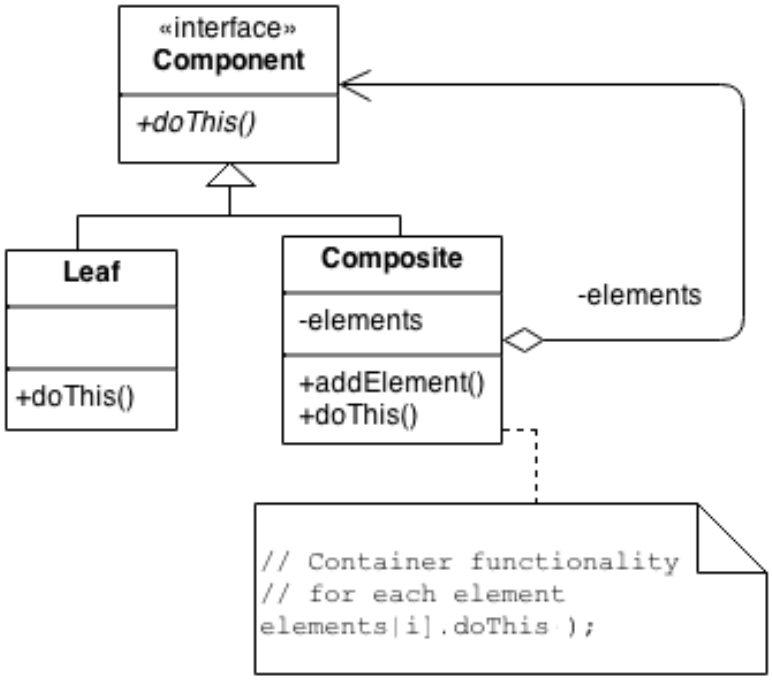
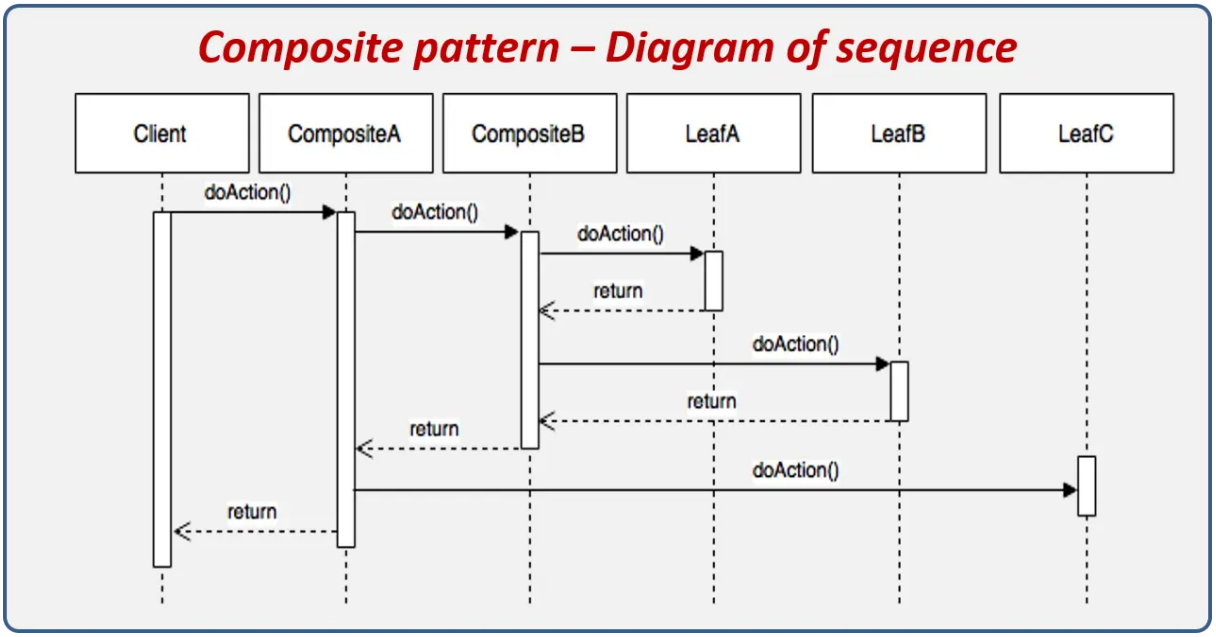


Diagrama de secuencia:



Participantes:

- Component: Se puede definir como una interfaz o clase abstracta y en esta se describen las operaciones y estructura común para todos los objetos en la jerarquía, ya sean elementos simples, o compuestos.
- Leaf: Definen el comportamiento y representan los elementos básicos o individuales dentro de la jerarquía. Debido a esto, las hojas no cuentan con hijos y en la mayoría de los casos, son las que realizan gran parte del trabajo en conjunto con otras hojas ya que no tienen a quien más delegar. Son del tipo Component, por lo que implementan sus comportamientos.
- Composite: Al igual que las hojas, también implementan la interfaz Component y sus comportamientos. Representan composiciones de objetos, por lo que tienen sub-elementos / hijos los cuales pueden ser hojas u otros contenedores. Debido a esto, también proporcionan operaciones específicas relacionadas a la administración de hijos, tales como agregar, eliminar, obtener su lista.

  Por otro lado, al implementar las operaciones de Component, realizan una delegación del trabajo a sus sub-elementos que tienen asociados para realizarlo y unificar el trabajo en un todo.
- Client: Es el usuario el cual interactúa y hace uso de la estructura jerárquica de objetos. Para esto, lo hace mediante la interfaz Component de tal manera que trata a todos los objetos (hojas y contenedores) de la misma manera, es decir uniformemente.

Aplicabilidad:

- Cuando se tenga que implementar una estructura de objetos con forma de árbol.
- Cuando se quiere que el cliente pueda ignorar la diferencia entre composiciones de objetos y los objetos individuales. El o los clientes tratarán a todos los objetos en la estructura del composite de forma uniforme

Consecuencias:

Las consecuencias del patrón Composite son:

- Genera que el código del cliente sea simple. Trabaja tanto las composiciones como los objetos individuales de la misma forma. Al cliente no le importa si está tratando con una hoja o un componente, lo que facilita el código.
- Favorece el principio de abierto cerrado, pues hace que sea más fácil añadir nuevos tipos de componentes. Pues al agregar una hoja o componente, la estructura del patrón trabaja exactamente igual a como se creó con anterioridad.
- Puede hacer que el diseño sea demasiado general. Al poder agregar tan fácilmente nuevos componentes, hace que sea más complicado restringir los componentes de la composición. Si se quiere restringir esta, crea la obligación de hacer comprobaciones durante la ejecución para crear estas restricciones.
- Puede resultar difícil proporcionar una interfaz común para clases cuya funcionalidad difiere demasiado. En algunos casos, tendrás que generalizar en exceso la interfaz componente, provocando que sea más difícil de comprender.
- Define una jerarquía de clases que consiste de objetos primitivos y composición de objetos.
- Permite trabajar con estructuras de árbol complejas con mayor comodidad, pues utiliza el polimorfismo y la recursión.