Composite Design Pattern

Miguel Angel Martinez Rodriguez
Arquitectura de Software, Ingeniería de Sistemas, Facultad de Ingeniería en Ciencias de la Computación y Telecomunicaciones.
Miguel_Angel-M@outlook.com

Abstract

.

1. Introducción

En esta ocasión tratare de explicar el patrón de diseño Compuesto (Composite Design Pattern). Este patrón de diseño denota una estructura de árbol, que se compone de partes complicadas y relacionadas. Existen muchas variaciones en la estructura de datos del árbol, pero a veces es necesario un árbol en el que tanto las ramas como las hojas del árbol deben tratarse de manera uniforme. El patrón compuesto le permite componer objetos en una estructura de árbol para representar la jerarquía de la parte completa, lo que significa que puede crear un árbol de objetos que está hecho de diferentes partes, pero que puede tratarse como una gran cosa. Composite permite a los clientes tratar objetos individuales y composiciones de objetos de manera uniforme, esa es la intención del Patrón Compuesto. Pero antes de profundizar en los detalles de un ejemplo, veamos algunos detalles más sobre el Patrón compuesto.

2. Patrón de diseño Compuesto (Composite Design Pattern)

El patrón compuesto es un patrón de diseño de partición y describe un grupo de objetos que se tratan de la misma manera que una sola instancia del mismo tipo de objeto. La intención de un compuesto es "componer" objetos en estructuras de árbol para representar jerarquías de parte completa. Le permite tener una estructura de árbol y solicitar a cada nodo en la estructura de árbol que realice una tarea.

- Según lo descrito por Gof, “Componga objetos en la estructura de árbol para representar jerarquías de parte completa. Composite permite al cliente tratar objetos individuales y composiciones de objetos de manera uniforme”.
- Cuando se trata de datos estructurados en árbol, los programadores a menudo tienen que discriminar entre un nodo hoja y una rama. Esto hace que el código sea más complejo y, por lo tanto, propenso a errores. La solución es una interfaz que permite tratar objetos complejos y primitivos de manera uniforme.
- En la programación orientada a objetos, un compuesto es un objeto diseñado como una composición de uno o más objetos similares, todos exhibiendo una funcionalidad similar. Esto se conoce como una relación "tiene-a" entre objetos.

El concepto clave es que puede manipular una sola instancia del objeto tal como manipularía un grupo de ellos. Las operaciones que puede realizar en todos los objetos compuestos a menudo tienen una relación de denominador menos común.

El patrón compuesto tiene cuatro participantes:

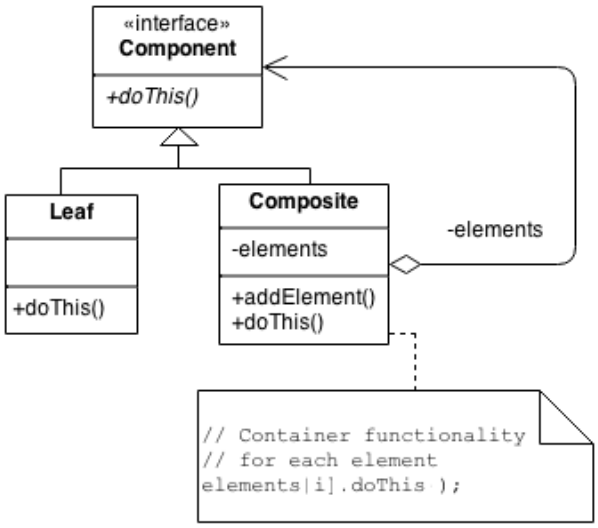
- **Componente**: el componente declara la interfaz para los objetos en la composición y para acceder y administrar sus componentes secundarios. También implementa un comportamiento predeterminado para la interfaz común a todas las clases, según corresponda.
- **Hoja**: la hoja define el comportamiento de los objetos primitivos en la composición. Representa objetos de hoja en la composición.
- **Compuesto**: Compuesto almacena componentes secundarios e implementa operaciones relacionadas con niños en la interfaz del componente.
- **Cliente**: el cliente manipula los objetos en la composición a través de la interfaz del componente.

3. Clasificación del patrón

Composite es un patrón de diseño estructural que le permite componer objetos en estructuras de árbol y luego trabajar con estas estructuras como si fueran objetos individuales.

4. Estructura

Compuestos que contienen componentes, cada uno de los cuales podría ser un compuesto.



Puede usarse por ejemplo en:

Menús que contienen elementos de menú, cada uno de los cuales podría ser un menú.

Directorios que contienen archivos, cada uno de los cuales podría ser un directorio.

Contenedores que contienen elementos, cada uno de los cuales podría ser un contenedor.

5. Observaciones

Composite y Decorator tienen diagramas de estructura similares, lo que refleja el hecho de que ambos se basan en la composición recursiva para organizar un número abierto de objetos.

El compuesto se puede atravesar con Iterator. El visitante puede aplicar una operación sobre un Compuesto. Composite podría usar la Cadena de responsabilidad para permitir que los componentes accedan a propiedades globales a través de su padre. También podría usar Decorator para anular estas propiedades en partes de la composición. Podría usar Observer para vincular una estructura de objeto a otra y State para permitir que un componente cambie su comportamiento a medida que cambia su estado.

Compuesto puede permitirle componer un Mediador a partir de piezas más pequeñas a través de una composición recursiva.

Decorator está diseñado para permitirle agregar responsabilidades a los objetos sin subclasificar. El enfoque de Composite no está en el adorno sino en la representación. Estas intenciones son distintas pero complementarias. En consecuencia, Composite y Decorator a menudo se usan en concierto.

6. Aplicabilidad

Use el patrón compuesto cuando tenga que implementar una estructura de objeto con forma de árbol.

El patrón compuesto le proporciona dos tipos de elementos básicos que comparten una interfaz común: hojas simples y contenedores complejos. Un contenedor puede estar compuesto de hojas y otros contenedores. Esto le permite construir una estructura de objetos recursivos anidados que se asemeja a un árbol.

Use el patrón cuando desee que el código del cliente trate los elementos simples y complejos de manera uniforme.

Todos los elementos definidos por el patrón compuesto comparten una interfaz común. Con esta interfaz, el cliente no tiene que preocuparse por la clase concreta de los objetos con los que trabaja.

7. Pros y contras

Pro: Puede trabajar con estructuras arbóreas complejas de manera más conveniente: utilice el polimorfismo y la recursividad para su ventaja.

Pro: Principio abierto / cerrado. Puede introducir nuevos tipos de elementos en la aplicación sin romper el código existente, que ahora funciona con el árbol de objetos.

Contra: Puede ser difícil proporcionar una interfaz común para las clases cuya funcionalidad difiere demasiado. En ciertos escenarios, necesitaría sobre generalizar la interfaz del componente, haciendo que sea más difícil de comprender.

8. Ejemplo

Imaginemos un sistema de organización de proyecto, el cual puede estar compuesto de trabajos a realizar el mismo también puede estar compuesto por tareas, siento este la unidad mínima.

La siguiente imagen ilustra la estructura de un paquete.

8.1. Diagrama de clases dinámico