PATRON ESTRUCTURAL:

COMPOSITE

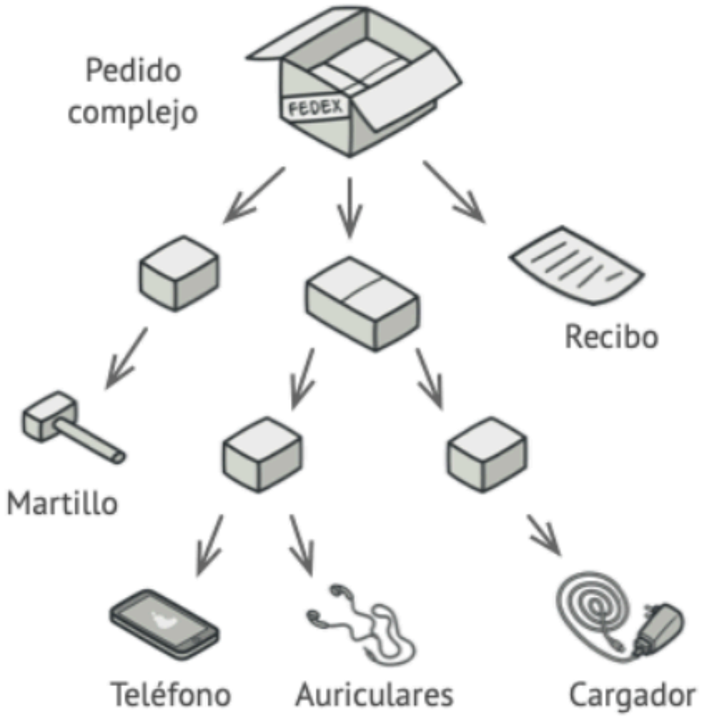
¿QUÉ ES UN PATRÓN DE DISEÑO ESTRUCTURAL?

Los patrones estructurales se centran en cómo se organiza la composición de clases y objetos para formar estructuras más grandes y flexibles.

Facilitar el diseño de estructuras de objetos mediante la organización de las relaciones entre clases y objetos, promoviendo la reutilización de código y simplificación de sistemas.

Ejemplos: Composite, Adapter, Decorator, Proxy, Bridge.

¿QUÉ ES EL PATRÓN COMPOSITE?



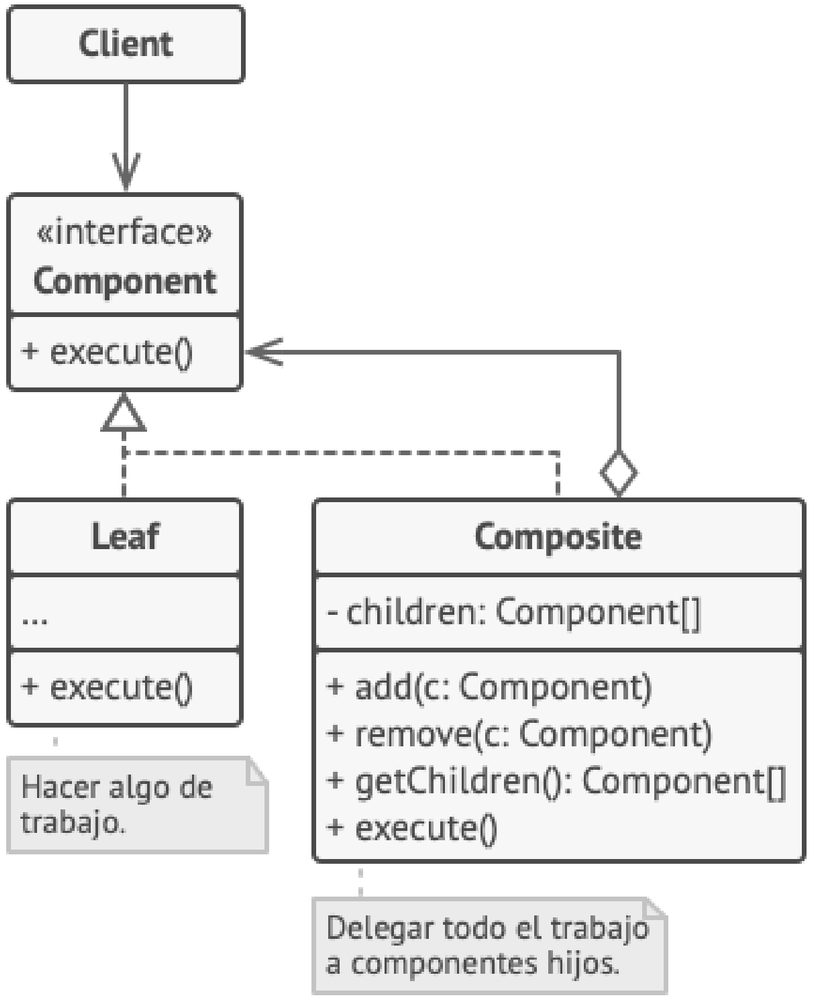
Permite tratar objetos individuales y agrupaciones de objetos de forma uniforme.

Organiza objetos en una estructura de árbol, donde los nodos pueden ser simples (hojas) o compuestos (contenedores).

¿CÓMO FUNCIONA EL PATRÓN COMPOSITE?

Estructura:

- Componente: Define una interfaz común para todos los objetos en la composición.
- Hoja: Representa un objeto individual (sin hijos).
- Compuesto: Representa un objeto que contiene otros objetos (tanto hojas como otros compuestos), que contiene varias subformas.



Funcionamiento:

Cuando un cliente realiza una solicitud (como calcular el precio o mover un gráfico), el objeto compuesto pasa la solicitud a sus subelementos de manera recursiva.
Las hojas procesan directamente la solicitud, mientras que los compuestos la delegan a sus hijos y luego combinan los resultados.

VENTAJAS Y DESVENTAJAS

VENTAJAS

- Simplificación del código cliente: El cliente no necesita preocuparse por la distinción entre objetos simples y compuestos.
- Se pueden agregar nuevos tipos de componentes sin modificar el código existente.

DESVENTAJAS

- La interfaz común puede volverse demasiado abstracta si los componentes tienen funcionalidades muy diferentes.
- Sobrecarga de complejidad: En algunos casos, manejar las relaciones entre objetos compuestos y simples puede volverse complejo.

BENEFICIOS DEL PATRÓN COMPOSITE

UNIFORMIDAD

Permite tratar de manera uniforme tanto a objetos simples como a compuestos sin necesidad de diferenciarlos en el código del cliente.

RECURSIVIDAD NATURAL

Aprovecha la recursividad para realizar operaciones en estructuras de árbol de forma sencilla.

FÁCIL DE EXTENDER

Puedes agregar nuevos tipos de componentes (hojas o compuestos) sin modificar el código existente.

```java
// Interfaz Graphic
interface Graphic {
    void move(int x, int y);
    void draw();
}

// Clase hoja Dot
class Dot implements Graphic {
    private int x;
    private int y;

    public Dot(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public void move(int x, int y) {
        this.x += x;
        this.y += y;
        System.out.println("Dot moved to (" + this.x + ", " + this.y + ")"
```

EJEMPLO

Este diseño permite manejar grupos de objetos gráficos como si fueran un solo objeto, simplificando la manipulación de estructuras complejas.

- Dot: Es una clase hoja que implementa Graphic. Representa un objeto gráfico simple (un punto). Define las coordenadas del punto y métodos para moverlo y dibujarlo.

- CompoundGraphic: Es una clase compuesta que implementa Graphic. Contiene una lista de objetos Graphic (hojas o compuestos) y delega las operaciones move() y draw() a todos sus hijos, creando una estructura recursiva.

GRACIAS

POR SU ATENCIÓN