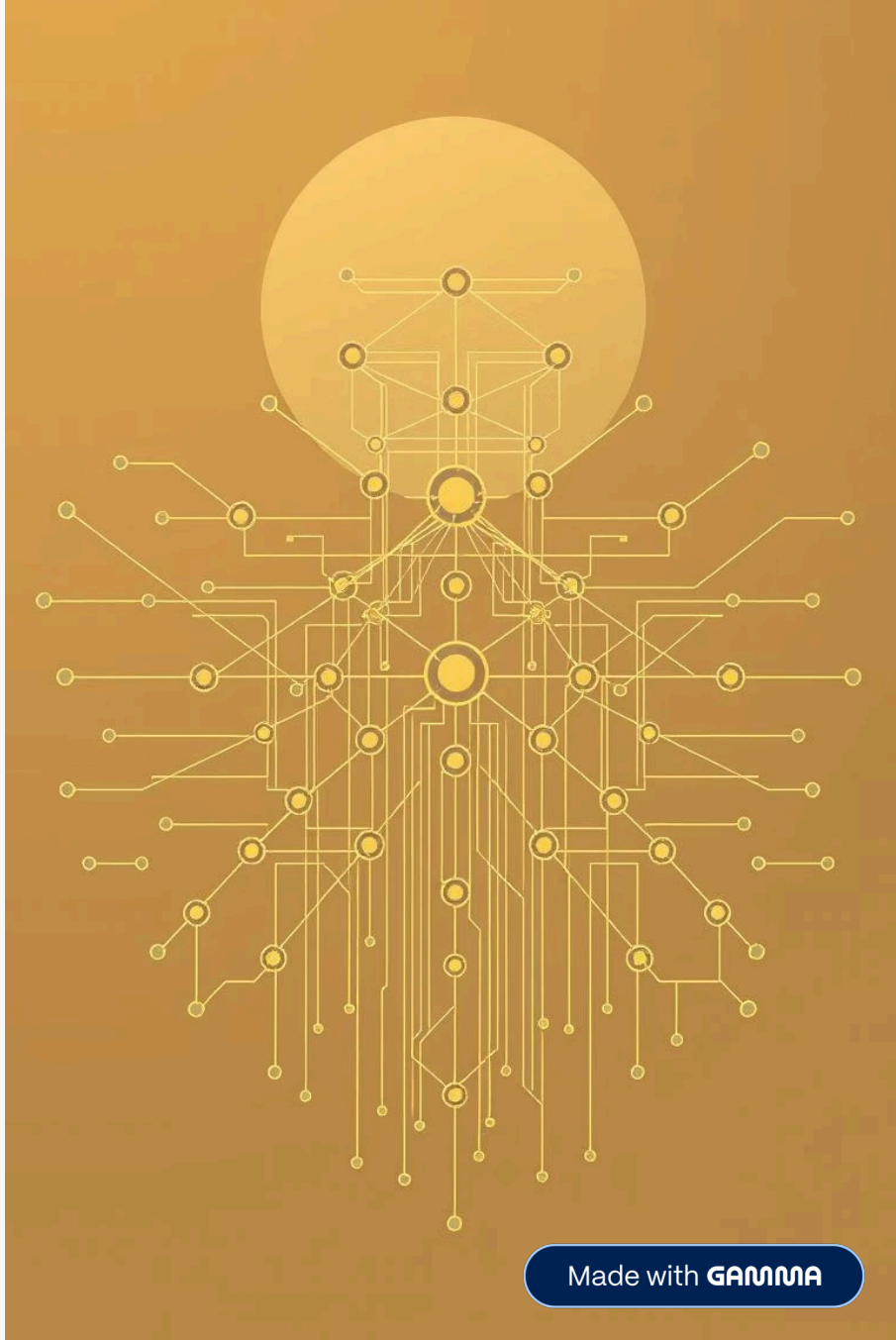
Patrón de Diseño Composite: Simplificando estructuras jerárquicas

El patrón Composite es una poderosa herramienta de diseño que permite tratar objetos individuales y grupos de objetos de manera uniforme. Es ideal para representar estructuras en forma de árbol, donde los nodos pueden ser tanto elementos terminales (hojas) como contenedores que a su vez pueden tener otros elementos (compuestos). Esta capacidad unificada facilita enormemente la gestión de jerarquías complejas, ya sean menús de aplicaciones, gráficos de escenas o sistemas organizativos de una empresa, simplificando el código y mejorando la flexibilidad.



Componentes clave del patrón Composite

1

Componente (Componente)

Define la interfaz común para hojas y compuestos. Asegura que todos los elementos en la jerarquía puedan ser tratados de la misma manera, ya sean objetos individuales o colecciones.

2

Hoja (Leaf)

Representa los objetos terminales de la jerarquía. No tienen hijos y implementan directamente las operaciones definidas por el Componente.

3

Compuesto (Composite)

Es un contenedor que puede incluir tanto Hojas como otros Compuestos. Delega las operaciones a sus hijos, permitiendo que las estructuras complejas actúen como objetos individuales.

Visualmente, se puede pensar en un árbol: las ramas serían los Compuestos que contienen otras ramas o Hojas, y las Hojas serían los elementos individuales. Todos ellos, ramas y hojas, responden de forma uniforme a las mismas acciones.

Ejemplo: implementación básica

A continuación, se muestra una implementación sencilla del patrón Composite en Python, representando un sistema de archivos donde tanto archivos como carpetas pueden ser manejados de forma similar.

```
from abc import ABC, abstractmethod

class Componente(ABC):
    @abstractmethod
    def mostrar_contenido(self):
        pass

class Archivo(Componente):
    def __init__(self, nombre):
        self.nombre = nombre

    def mostrar_contenido(self):
        return f"Archivo: {self.nombre}"

class Carpeta(Componente):
    def __init__(self, nombre):
        self.nombre = nombre
        self.hijos = []

    def agregar(self, componente):
        self.hijos.append(componente)

    def eliminar(self, componente):
        self.hijos.remove(componente)

    def mostrar_contenido(self):
        contenido = f"Carpeta: {self.nombre}\n"
        for hijo in self.hijos:
            contenido += f"  - {hijo.mostrar_contenido()}\n"
        return contenido

# Uso del patrón
documentos = Carpeta("Documentos")
documentos.agregar(Archivo("informe.pdf"))
documentos.agregar(Archivo("plan.docx"))

imagenes = Carpeta("Imágenes")
imagenes.agregar(Archivo("foto1.jpg"))
imagenes.agregar(Archivo("paisaje.png"))

raiz = Carpeta("Mi PC")
raiz.agregar(documentos)
raiz.agregar(imagenes)
raiz.agregar(Archivo("readme.txt"))

print(raiz.mostrar_contenido())
```

Este código demuestra cómo los objetos `Archivo` (Hojas) y `Carpeta` (Compuestos) implementan la misma interfaz `Componente`, permitiendo que el sistema trate una carpeta de la misma manera que un archivo al recorrer su contenido.