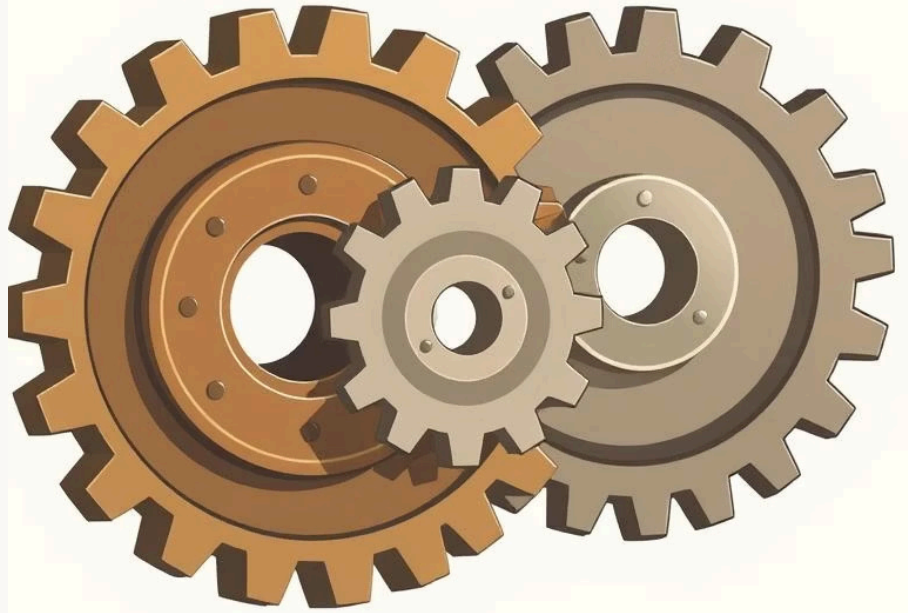
Patrón de Diseño Adapter: Conectando Interfaces Incompatibles

En el mundo del desarrollo de software, la flexibilidad y la reutilización son clave. Pero, ¿qué sucede cuando tienes dos piezas de software que necesitan comunicarse, pero hablan "idiomas" diferentes? Aquí es donde entra en juego el patrón de diseño Adapter.



¿Qué es el Patrón Adapter?

1

Patrón Estructural

Permite que objetos con interfaces incompatibles colaboren. Es un componente fundamental en la arquitectura de software.

2

Actúa como Puente

Traduce la interfaz de una clase a otra esperada por el cliente, permitiendo una comunicación fluida.

3

Ejemplo Cotidiano

Imagina un adaptador de enchufe que permite conectar dispositivos con clavijas diferentes en un mismo tomacorriente.

4

Grandes Beneficios

Mejora la flexibilidad del sistema, fomenta la reutilización de código y cumple el Principio Abierto/Cerrado de SOLID.

El patrón Adapter es esencial para integrar sistemas preexistentes o bibliotecas de terceros que no siguen las mismas convenciones de interfaz.

Ejemplo: Adaptando vehículos con diferentes interfaces

Consideremos un sistema donde necesitamos interactuar con diferentes tipos de vehículos: coches eléctricos y coches de gasolina. Cada uno tiene una forma distinta de "recargar" o "repostar".

```
class CocheElectrico:
    def cargar_bateria(self):
        return "Coche eléctrico cargando batería."

class CocheGasolina:
    def repostar(self):
        return "Coche de gasolina repostando."
```

Para unificarlos bajo una misma interfaz, creamos un adaptador:

```
class AdaptadorVehiculo:
    def __init__(self, vehiculo):
        self._vehiculo = vehiculo

    def recargar(self):
        if hasattr(self._vehiculo, 'cargar_bateria'):
            return self._vehiculo.cargar_bateria()
        elif hasattr(self._vehiculo, 'repostar'):
            return self._vehiculo.repostar()
        else:
            return "Método de recarga no encontrado."
```

Este ejemplo simplifica la interacción con cualquier tipo de vehículo, demostrando cómo el patrón Adapter nos permite trabajar con interfaces incompatibles de manera transparente para el cliente.