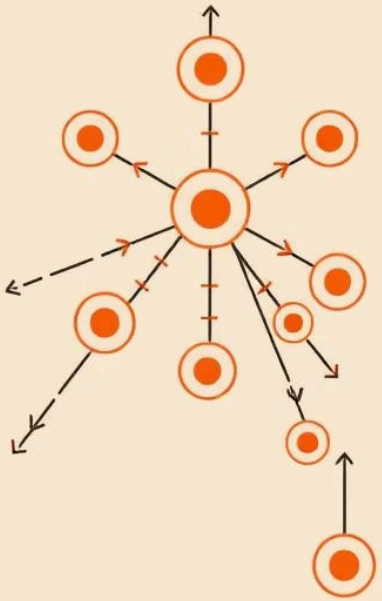

Patrón de Diseño State: Cambia el Comportamiento según el Estado

El Patrón de Diseño State es una herramienta poderosa que permite a un objeto alterar su comportamiento cuando su estado interno cambia. Es como si el objeto se transformara, adaptándose dinámicamente a nuevas situaciones sin necesidad de anidar complejos condicionales "if-else". Esto hace que el código sea más limpio, mantenible y extensible.

Este patrón es ideal para sistemas con múltiples estados y transiciones dinámicas, donde el comportamiento de un objeto depende completamente de su situación actual. Piensa en procesos de pago, flujos de trabajo, o la lógica de una máquina expendedora: todos son escenarios perfectos para el patrón State.

Componentes Clave del Patrón State

Contexto

El "Contexto" es la clase que contiene la referencia al estado actual y delega las peticiones a ese estado. Es el punto de interacción principal para el cliente, que no necesita saber nada sobre los estados concretos.

Estado (Interfaz)

La interfaz "Estado" define un conjunto de métodos que todos los estados concretos deben implementar. Esto asegura que todos los estados manejen las mismas operaciones, pero de maneras diferentes.

Estados Concretos

Los "Estados Concretos" son las implementaciones individuales de la interfaz Estado. Cada clase de estado concreto implementa un comportamiento específico y puede contener lógica para cambiar el estado del Contexto a otro estado concreto.

La belleza de este patrón reside en su adherencia al principio abierto/cerrado: puedes añadir nuevos estados al sistema sin modificar el código existente del Contexto, lo que facilita enormemente la evolución del software.

Ejemplo: Cambio de Estado Dinámico

Imagina una luz inteligente que puede estar encendida, apagada o en modo de ahorro de energía. Su comportamiento (por ejemplo, cómo responde a un botón) cambia según su estado actual.

```
class EstadoLuz:
    def presionar_boton(self, luz):
        pass

class Apagada(EstadoLuz):
    def presionar_boton(self, luz):
        print("Luz: Encendiendo...")
        luz.establecer_estado(Encendida())

class Encendida(EstadoLuz):
    def presionar_boton(self, luz):
        print("Luz: Poniendo en ahorro de energía...")
        luz.establecer_estado(AhorroEnergia())

class AhorroEnergia(EstadoLuz):
    def presionar_boton(self, luz):
        print("Luz: Apagando...")
        luz.establecer_estado(Apagada())

class LuzInteligente:
    def __init__(self):
        self._estado = Apagada()

    def establecer_estado(self, estado):
        self._estado = estado

    def presionar_boton(self):
        self._estado.presionar_boton(self)

# Uso
luz = LuzInteligente()
luz.presionar_boton() # Enciende
luz.presionar_boton() # Ahorro de energía
luz.presionar_boton() # Apaga
luz.presionar_boton() # Enciende de nuevo
```

En este ejemplo, la clase `LuzInteligente` (Contexto) delega el manejo del botón a su `EstadoLuz` actual. Cada estado concreto (`Apagada`, `Encendida`, `AhorroEnergia`) implementa `presionar_boton` de manera diferente y puede cambiar el estado de la luz. Esto demuestra cómo el patrón State simplifica la lógica y facilita la adición de nuevos estados.