Patrón Flyweight: Optimización de Objetos Compartidos

El patrón Flyweight es un patrón estructural fundamental que tiene como objetivo principal la reducción drástica del consumo de memoria. Lo logra al permitir que una gran cantidad de objetos compartan un estado común, evitando así la duplicación innecesaria de datos en sistemas con alta redundancia. Es particularmente beneficioso en aplicaciones que manejan una multitud de objetos con características idénticas o muy similares, donde la creación de instancias individuales para cada uno resultaría en un uso ineficiente de los recursos.

La clave del Flyweight radica en la separación inteligente del estado de un objeto en dos categorías distintas:

- **Estado Intrínseco**: Esta parte del estado es compartida y permanece constante entre múltiples objetos. Representa la información que no cambia de una instancia a otra y, por lo tanto, puede ser almacenada y reutilizada (ej., color de un coche, modelo de un vehículo).
- **Estado Extrínseco**: Este estado es único y variable para cada objeto individual. Es la información que diferencia a una instancia de otra y debe ser proporcionada por el cliente al usar el flyweight (ej., matrícula de un coche, nombre del propietario).

El beneficio central de este patrón es que evita la creación de objetos duplicados, reutilizando eficientemente las instancias existentes y, por ende, optimizando el uso de la memoria y mejorando el rendimiento general del sistema.

Componentes Clave del Patrón Flyweight

Flyweight

Es la interfaz o clase abstracta que declara el método que permite al flyweight recibir y actuar sobre el estado extrínseco. Los objetos flyweight concretos implementan esta interfaz y contienen el estado intrínseco compartido.

Flyweight Factory

Responsable de gestionar y proveer las instancias de flyweights. Esta fábrica mantiene un pool de flyweights existentes y asegura que no se creen duplicados. Si un flyweight solicitado ya existe, lo devuelve; de lo contrario, crea una nueva instancia y la almacena para futuras solicitudes.

Cliente

Es el componente que utiliza los flyweights. El cliente mantiene el estado extrínseco de cada objeto y pasa esta información a los métodos del flyweight cuando es necesario. Es crucial que el cliente sepa diferenciar y gestionar el estado extrínseco.

Para ilustrar el patrón, consideremos un sistema de gestión de vehículos. Aquí, las características comunes a muchos coches, como el modelo (ej. "Sedán", "SUV") y el color (ej. "Rojo", "Azul"), representan el **estado intrínseco**.

Por otro lado, la matrícula y el propietario de cada coche son ejemplos del **estado extrínseco**, ya que son únicos para cada vehículo y deben ser proporcionados por el cliente al interactuar con las instancias flyweight.

Ejemplo: Gestión de Coches con Flyweight

A continuación, se presenta un ejemplo práctico en Python para gestionar una gran cantidad de objetos de coches utilizando el patrón Flyweight, demostrando cómo se optimiza la memoria al compartir datos intrínsecos como el modelo y el color.

```
class CocheFlyweight:
    def __init__(self, modelo, color):
        self._modelo = modelo
        self._color = color

    def mostrar_info(self, matricula, propietario):
        print(f"Coche: Modelo {self._modelo}, Color {self._color}, "
              f"Matrícula {matricula}, Propietario {propietario}")

class CocheFactory:
    _flyweights = {}

    def get_flyweight(self, modelo, color):
        key = (modelo, color)
        if key not in self._flyweights:
            print(f"Creando nuevo Flyweight para {modelo} {color}")
            self._flyweights[key] = CocheFlyweight(modelo, color)
        return self._flyweights[key]

if __name__ == "__main__":
    factory = CocheFactory()

    # Creación de coches, reutilizando flyweights cuando sea posible
    coche1 = factory.get_flyweight("Sedán", "Rojo")
    coche1.mostrar_info("ABC-123", "Juan Pérez")

    coche2 = factory.get_flyweight("SUV", "Azul")
    coche2.mostrar_info("DEF-456", "María Gómez")

    coche3 = factory.get_flyweight("Sedán", "Rojo") # Reutiliza el flyweight existente
    coche3.mostrar_info("GHI-789", "Carlos Ruiz")

    coche4 = factory.get_flyweight("Sedán", "Negro")
    coche4.mostrar_info("JKL-012", "Ana López")

    print("\nNúmero total de Flyweights únicos creados:", len(factory._flyweights))
```

En este código, `CocheFlyweight` guarda el estado intrínseco (modelo, color), mientras que `CocheFactory` asegura la reutilización. Al ejecutar, veremos que a pesar de crear cuatro "coches", solo se instancian tres `CocheFlyweight` únicos, demostrando la eficiencia en memoria del patrón Flyweight.