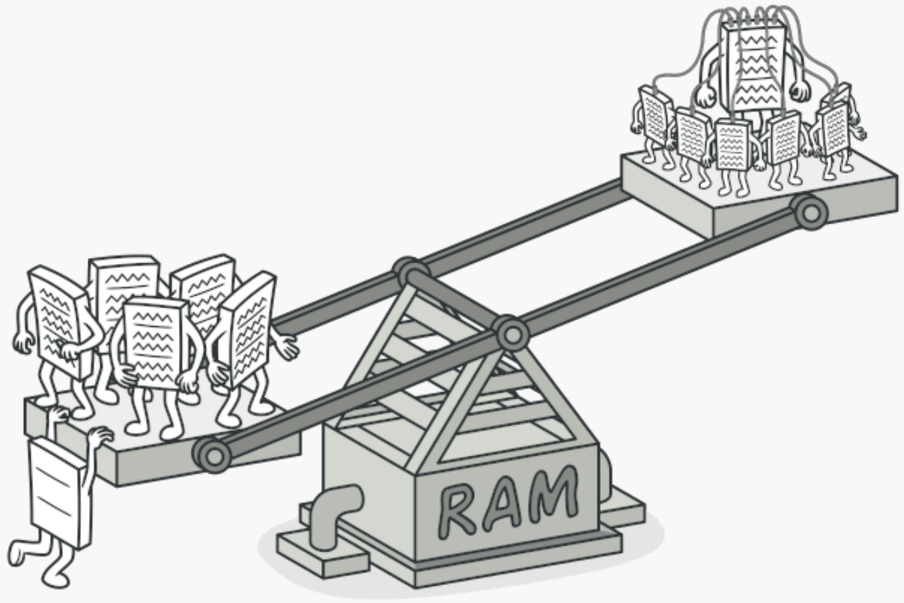
# Patron Estructural: Flyweight




Roger Arjona

# Patrones de Diseño

Solución general repetible a un problema que ocurre comúnmente en el diseño de software.

# Criticas de hacia los Patrones

- Los patrones compensan las características faltantes del lenguaje.
- Los patrones repiten las buenas practicas.
- Los patrones pueden conducir a una sobre-ingeniería/ingenieria excesiva.

# ¿Como utilizar los patrones?

- Los patrones se utilizan mejor para comunicar que estás siguiendo un enfoque de diseño bien comprendido. (**Cuando los entiendes bien**)
- No implementar un patrón si el lenguaje tiene una solución directa.
- No intentes adaptar todo en términos de patrones.Usa un patron unicamente si es la solucion mas elegante.
- Usa un patron unicamente si es la solucion mas elegante.
- No tengas miedo de crear nuevos patrones.

# Flyweight

Optimizar el uso de la memoria y mejorar el rendimiento en aplicaciones que utilizan una gran cantidad de objetos.

# Cuando aplicarlo:

01

### Millones de Objetos

Se necesita generar una cantidad enorme de objetos similares.

02

### RAM limitada

Se consume toda la RAM disponible de un dispositivo objetivo.

03

### Información Repetida

Los objetos contienen estados duplicados que se pueden extraer y compartir.

# Para comprenderlo ...

1: 100 años de soledad, Gabriel Garcia Marquez, "Muchos años después, frente al pelotón de fusilamiento ...", Porrua, Edicion de Bolsillo, 100
2: 100 años de soledad, Gabriel Garcia Marquez, "Muchos años después, frente al pelotón de fusilamiento ...", Alfaguara, Edicion Especial, 100
3: 100 años de soledad, Gabriel Garcia Marquez, "Muchos años después, frente al pelotón de fusilamiento ...", Santillana, Edicion de Lujo, 150
4: 100 años de soledad, Gabriel Garcia Marquez, "Muchos años después, frente al pelotón de fusilamiento ...", Gandhi, Edicion de Bolsillo, 80
5: 100 años de soledad, Gabriel Garcia Marquez, "Muchos años después, frente al pelotón de fusilamiento ...", Porrua, Edicion Especial, 110

09

# Implementación

Del papel al Código.

01

**Divide los campos de una clase que se convertirá en flyweight en dos partes.**

Separar lo **Intrínseco** de lo **Intrínseco**

## Intrínseco

- Nombre
- Contenido
- Autor

## Extrínseco

- Editorial
- Edición
- Precio

02

## Deja los campos que representan el estado intrínseco en la clase, pero asegúrate de que sean inmutables.

Deben llevar sus valores iniciales únicamente dentro del constructor.

```
class LibroFlyweight:
    """ Flyweight que representa el contenido compartido de un libro (Intrinseco). Es inmutable debido a
que no hay un metodo que actualice los valores"""

    def __init__(self, nombre, autor, contenido):
        self.nombre = nombre
        self.autor = autor
        self.contenido = contenido

class Libro:
    """ Clase de libro que mantiene su estado único (extrínseco). """
    def __init__(self, editorial, edicion, precio, nombre, autor, contenido):
        self.editorial = editorial
        self.portada = portada
        self.precio = precio
        self.flyweight = LibroFlyweight(nombre, autor, contenido)
  return go(f, seed, [])
}
```

03

## Repasa los métodos que utilizan campos del estado extrínseco. Para cada campo utilizado en el método, introduce un nuevo parámetro y utilízalo en lugar del campo.

Esto se hace cuando se refactoriza código para implementar el patrón.

```python
class LibroFlyweight:

    def __init__(self, nombre, autor, contenido):
        ...

    def mostrar_contenido_en_pagina(self, pagina_actual):
        # Aquí se utiliza la 'pagina_actual' para mostrar el contenido
        print(f"Mostrando contenido de la página {pagina_actual}: {self.contenido[pagina_actual]}")
```

04

## Opcionalmente, crea una clase fábrica para gestionar el grupo de objetos flyweight, buscando uno existente antes de crear uno nuevo

Una vez que la fábrica esté en su sitio, los clientes sólo deberán solicitar objetos flyweight a través de ella.

```
class LibroFactory:
    """ Fábrica Flyweight para crear o reutilizar instancias de LibroFlyweight. """
    _libros_flyweight = {}

    @classmethod
    def obtener_flyweight(cls, contenido):
        if contenido not in cls._libros_flyweight:
            cls._libros_flyweight[contenido] = LibroFlyweight(contenido)
        return cls._libros_flyweight[contenido]
```

# Pasemos al Código

Porque nada podría salir mal cuando ejecutas código en vivo

1 x 385.8 MB

Flyweight

4 x 358.8 MB

Consumo elevado de RAM

# Ventajas

01

### Ahorro de RAM

Puedes ahorrar mucha RAM, siempre que tu programa tenga toneladas de objetos similares.

# Desventajas

## Uso de CPU

01

Puede que estés cambiando RAM por ciclos CPU cuando deba calcularse de nuevo parte de la información de contexto cada vez que alguien invoque un método flyweight.

## Codigo

02

El código se complica mucho. Los nuevos miembros del equipo siempre estarán preguntándose por qué el estado de una entidad se separó de tal manera.

# ¿Aplica en Python?

Yes, no, maybe. I don’t know
Can you repeat the question?

# In Real Life

01

**Videojuegos**

02

**Edición Gráfica**

03

**Sistemas Operativos**

# Páginas Web

01

**Paginación**

02

**Lazy Loading**

03

**Componentes**

Sigue un principio similar: minimizar el uso de recursos (en este caso, memoria y tiempo de renderizado en el navegador) al limitar la cantidad de datos procesados y mostrados simultáneamente.

# Dudas y Preguntas

No por favor, yo se lo pregunte a ChatGPT