Patrón de Diseño Template Method: Definiendo el esqueleto de un algoritmo

El patrón de diseño Template Method es una herramienta poderosa en la programación orientada a objetos que nos permite estructurar algoritmos complejos de manera flexible. Define el esqueleto de un algoritmo en una operación, posponiendo algunos pasos a las subclases. Esto permite a las subclases redefinir ciertos pasos del algoritmo sin cambiar su estructura general. Es una estrategia fundamental para la reutilización de código y la extensibilidad en el desarrollo de software.

- Es un patrón de comportamiento que define la estructura general de un algoritmo en una clase base.
- Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura global.
- Muy usado en frameworks para extender funcionalidades sin duplicar código.
- Ventaja clave: reutilización y flexibilidad al separar el "qué" se hace del "cómo" se hace.

¿Cómo funciona el Template Method?

Estructura general

Imaginemos que queremos procesar diferentes tipos de documentos (PDF, Word, TXT). El proceso general es similar: abrir, leer, procesar y cerrar. Sin embargo, la forma específica de "leer" o "procesar" difiere para cada tipo de documento.

```
class DocumentProcessor:
    def process_document(self):
        self.open_document()
        self.read_document()
        self.process_content()
        self.close_document()

    def open_document(self):
        print("Abriendo documento genérico...")

    def read_document(self):
        raise NotImplementedError("Debe ser
implementado por subclases")

    def process_content(self):
        print("Procesando contenido genérico...")

    def close_document(self):
        print("Cerrando documento genérico.")
```

Ejemplo de implementación

Aquí, `read_document` es el "método plantilla" que las subclases deben implementar. Los demás métodos son "hooks" que pueden ser sobrescritos opcionalmente.

```
class PDFProcessor(DocumentProcessor):
    def read_document(self):
        print("Leyendo contenido de PDF.")

    def process_content(self):
        print("Extrayendo texto e imágenes de PDF.")

class WordProcessor(DocumentProcessor):
    def read_document(self):
        print("Leyendo contenido de documento Word.")

    def process_content(self):
        print("Analizando formato y estructura de Word.")

# Uso del patrón
pdf_doc = PDFProcessor()
print("--- Procesando PDF ---")
pdf_doc.process_document()

word_doc = WordProcessor()
print("\n--- Procesando Word ---")
word_doc.process_document()
```

En este ejemplo, `DocumentProcessor` define el algoritmo general (el método plantilla `process_document`). Las subclases `PDFProcessor` y `WordProcessor` implementan los pasos específicos (`read_document` y `process_content`) de una manera adaptada a su tipo de documento, sin alterar el flujo general definido en la clase base.