Patrón de Diseño Visitor: Separando Algoritmos de Estructuras de Objetos

El patrón Visitor es una técnica de diseño que permite añadir nuevas operaciones a una estructura de objetos sin modificar las clases de esos objetos. Esto es ideal para sistemas con jerarquías complejas donde se necesitan múltiples operaciones distintas, resolviendo el problema de modificar clases en producción y manteniendo el principio abierto/cerrado.

Extensibilidad

Permite añadir nuevas operaciones fácilmente sin alterar el código existente de la estructura de objetos.

Separación Clara

Mantiene los algoritmos separados de las estructuras de objetos en las que operan, mejorando la modularidad.

Cumple el Principio

Abre para extensión, cerrado para modificación, un pilar fundamental del diseño de software robusto.

Ejemplo: Visitor para Componentes Concretos

Imaginemos una estructura de documentos con componentes como Títulos y Párrafos. Queremos añadir funcionalidades como exportar a diferentes formatos (HTML, Markdown) sin modificar las clases originales.

Estructura Base

```
class Component:
    def accept(self, visitor):
        pass

class Title(Component):
    def __init__(self, text):
        self.text = text
    def accept(self, visitor):
        visitor.visit_title(self)

class Paragraph(Component):
    def __init__(self, text):
        self.text = text
    def accept(self, visitor):
        visitor.visit_paragraph(self)
```

Implementación del Visitor

```
class DocumentVisitor:
    def visit_title(self, title):
        pass
    def visit_paragraph(self, paragraph):
        pass

class HtmlExportVisitor(DocumentVisitor):
    def visit_title(self, title):
        return f"<h1>{title.text}</h1>"
    def visit_paragraph(self, paragraph):
        return f"<p>{paragraph.text}</p>"
```

Uso del Patrón Visitor

```
document_components = [
    Title("Mi Documento"),
    Paragraph("Este es un párrafo de ejemplo."),
    Paragraph("Otro párrafo importante aquí.")
]

html_exporter = HtmlExportVisitor()
for component in document_components:
    print(component.accept(html_exporter))
```

Este ejemplo ilustra cómo el `HtmlExportVisitor` "visita" cada componente (`Title` y `Paragraph`) para generar la salida HTML, sin que las clases de los componentes necesiten conocer los detalles de cómo se exportan.