

METODOLOGÍAS DE PROGRAMACIÓN I

Patrón estructural *Composite*

Situación de ejemplo

- Se posee una clase llamada *Documento* la cual sabe leerse de archivo, guardarse e imprimirse.
- Un documento tiene título y puede contener texto, figuras y tablas.

```
class Documento
    string titulo
    List parrafos
    List figuras
    List tablas

    void leer(filename)
    void escribir(filename)
    void imprimir(impresora)
```



Situación de ejemplo

```
class Documento
    string titulo
    List parrafos
    List figuras
    List tablas

    void leer(filename)
        foreach p in parrafos
            // lectura del párrafo p
        foreach f in figuras
            // lectura de la figura f
        foreach t in tablas
            // lectura de la tabla t
```



Problema

¿Qué sucede con la clase *Documento* si aparecen nuevos requerimientos y un documento puede contener secciones, las cuales a su vez tienen títulos, textos, figuras y tablas?

INDICE GENERAL	
RESUMEN	ii
ABSTRACT	iii
INDICE GENERAL	iv
INDICE DE TABLAS	vii
INDICE DE FIGURAS	viii
INDICE DE GRÁFICOS	x
INDICE DE PANTALLAS	xg
CAPÍTULO I	4
EL PROBLEMA	4
1.1 PLANTEAMIENTO DEL PROBLEMA	4
1.2 JUSTIFICACIÓN E IMPORTANCIA DE LA INVESTIGACIÓN	6



¿Qué sucede con las operaciones de leer, escribir e imprimir?

Situación de ejemplo

```
class Documento
  string titulo
  List parrafos
  List figuras
  List tablas
```

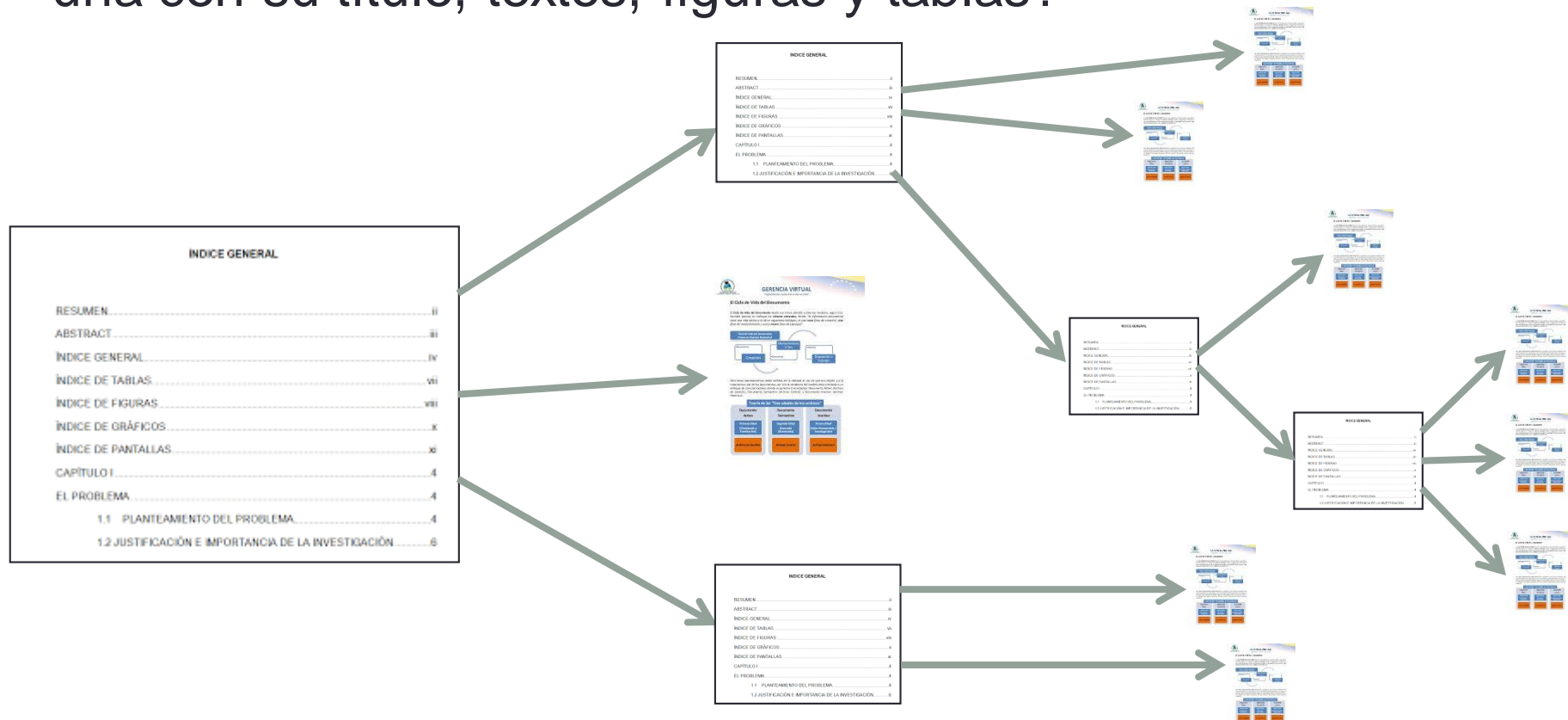
INDICE GENERAL	
RESUMEN.....	ii
ABSTRACT.....	iii
INDICE GENERAL.....	iv
INDICE DE TABLAS.....	vii
INDICE DE FIGURAS.....	x
INDICE DE PANTALLAS.....	xii
CAPITULO I.....	1
EL PROBLEMA.....	1
1.1 PLANTEAMIENTO DEL PROBLEMA.....	1
1.2 JUSTIFICACIÓN E IMPORTANCIA DE LA INVESTIGACIÓN.....	2

```
void leer(filename)
  foreach s in secciones
    foreach p in parrafos
      // lectura del párrafo p
      // de la sección s
    foreach f in figuras
      // lectura de la figura f
      // de la sección s
    foreach t in tablas
      // lectura de la tabla t
      // de la sección s
```



Problema

¿Qué sucede si ahora las secciones pueden tener subsecciones, y estas a su vez más subsecciones cada una con su título, textos, figuras y tablas?



Situación de ejemplo

```
class Documento
```

```
    string titulo
```

```
    List parrafos
```

```
    List figuras
```

```
    List tablas
```

```
void leer(filename)
```

```
    foreach . . .
```

```
        foreach . . .
```

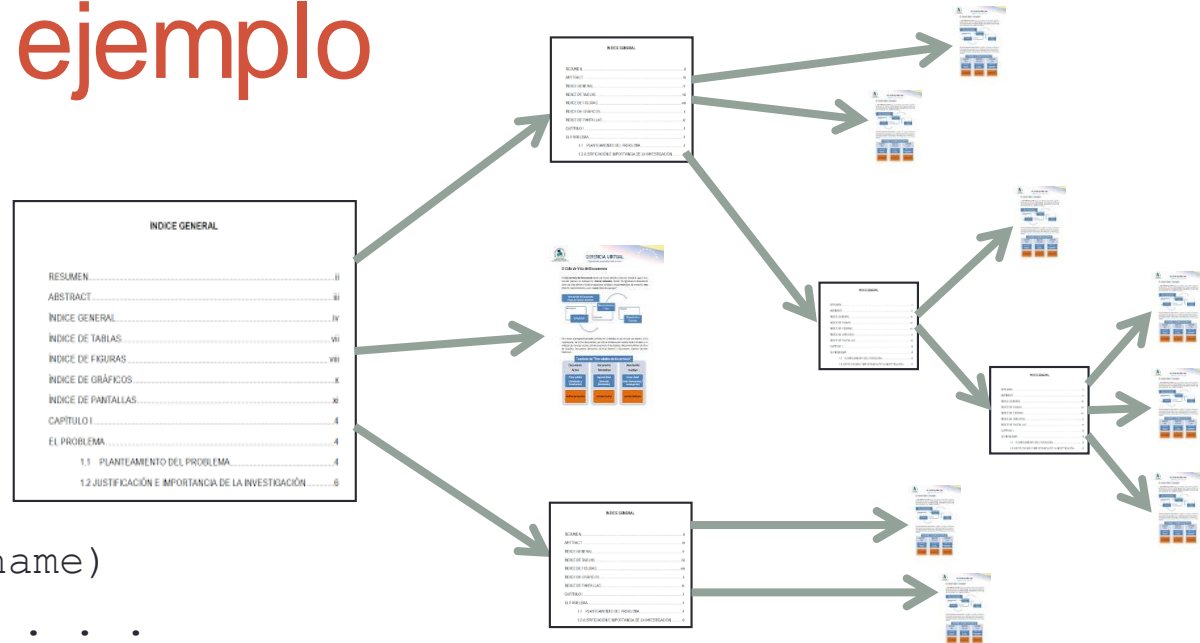
```
            foreach . . .
```

```
                foreach . . .
```

```
                    foreach p in parrafos
```

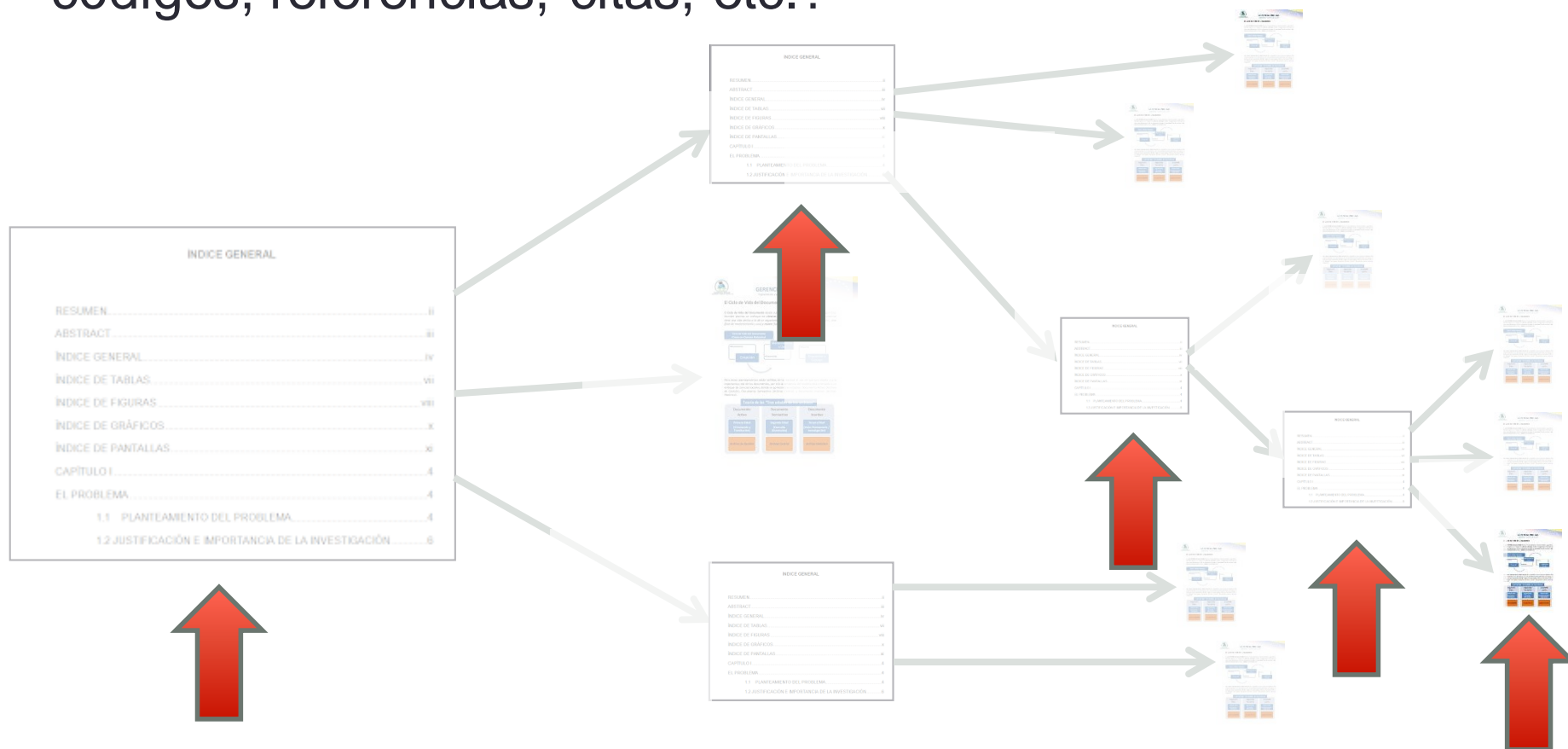
```
                    foreach f in figuras
```

```
                    foreach t in tablas
```



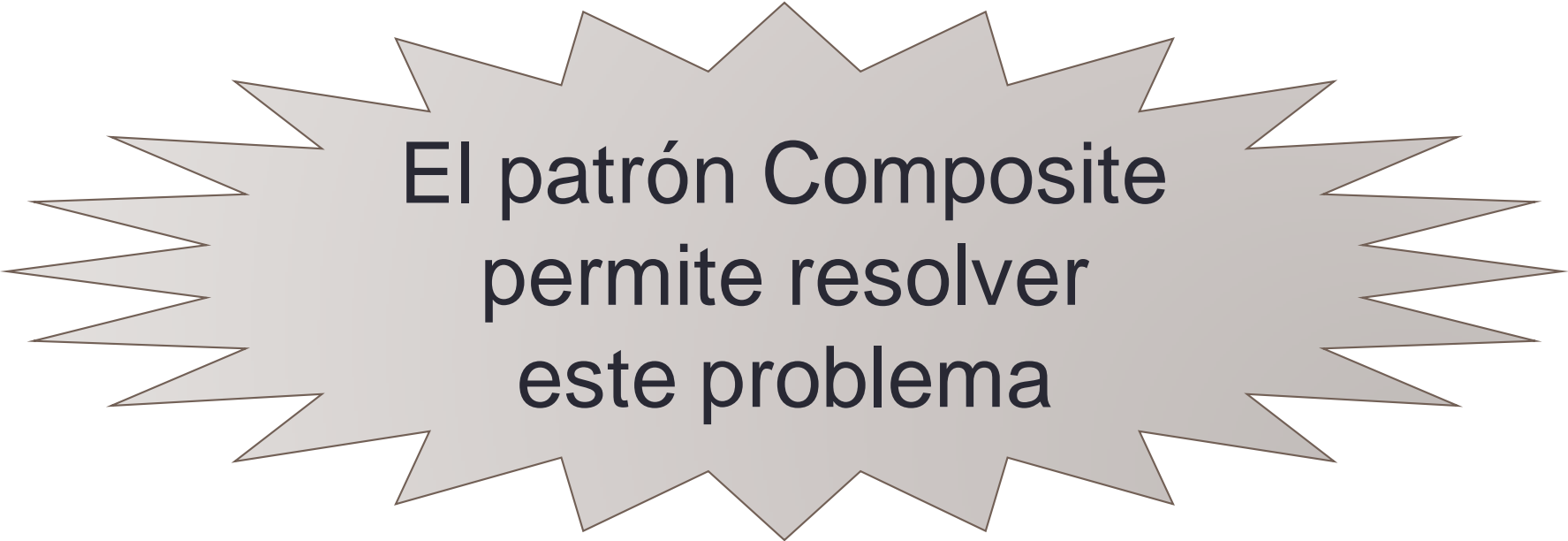
Problema

¿Qué sucede si a futuro además de textos, tablas y figuras aparecen más elementos como ejemplos, ilustraciones, códigos, referencias, citas, etc.?



Motivación

Buscamos un diseño que permita tener un único documento, el cual puede estar dividido en documentos más pequeños y estos estar compuestos por cualquier cantidad de partes y que sea fácilmente extensible.



**El patrón Composite
permite resolver
este problema**

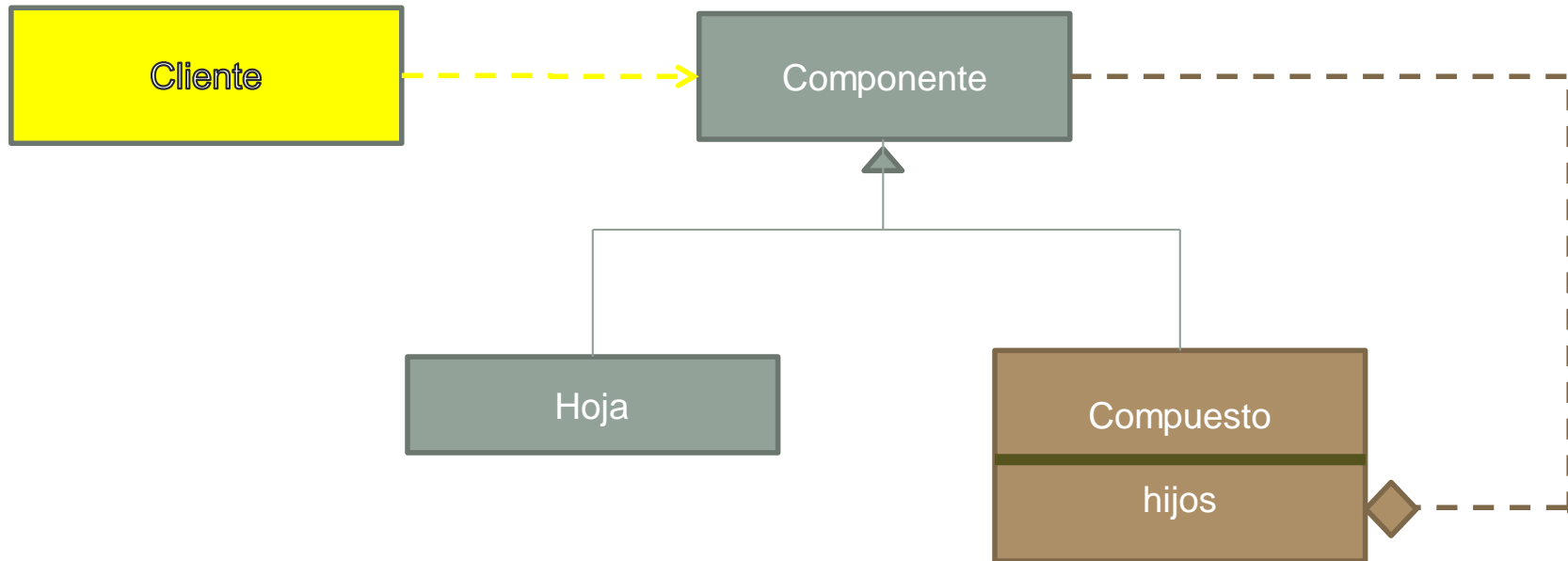
Composite

Propósito: compone objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos

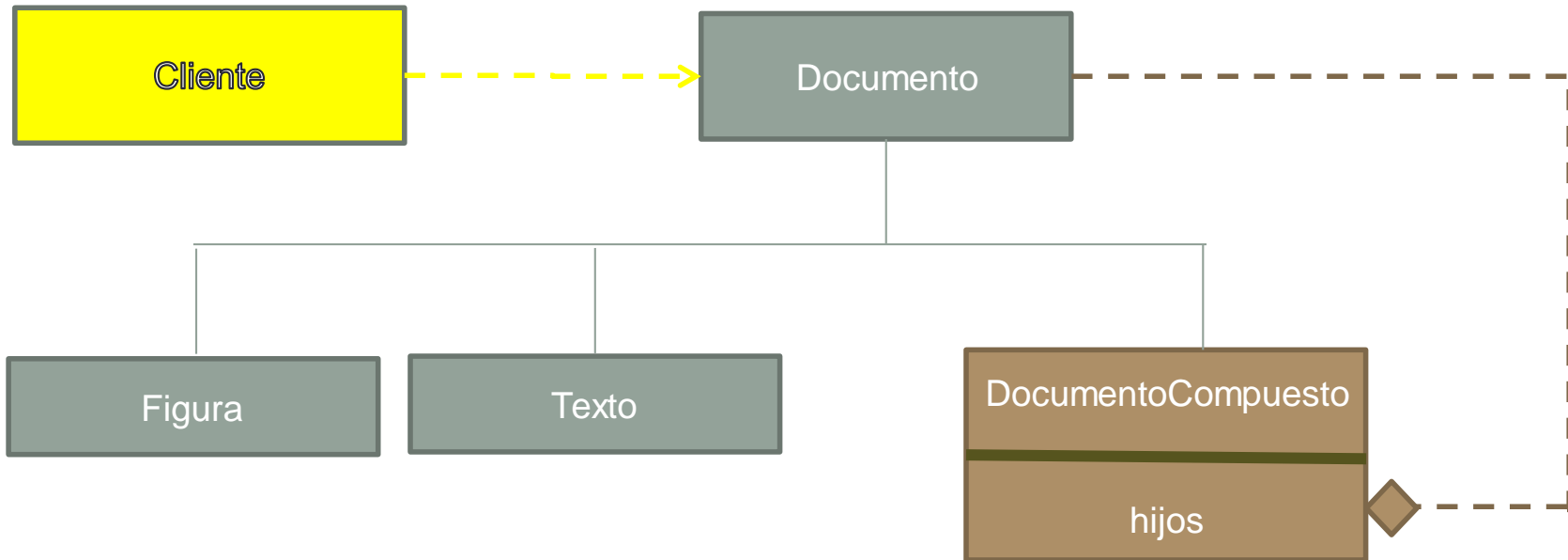
Aplicabilidad: usarlo cuando

- Quiera representar jerarquías de objetos parte-todo.
- Quiera que los clientes sean capaces de obviar las diferencias entre composiciones de objetos y los objetos individuales.
- Quiera que los clientes traten a todos los objetos de la estructura compuesta de manera uniforme.

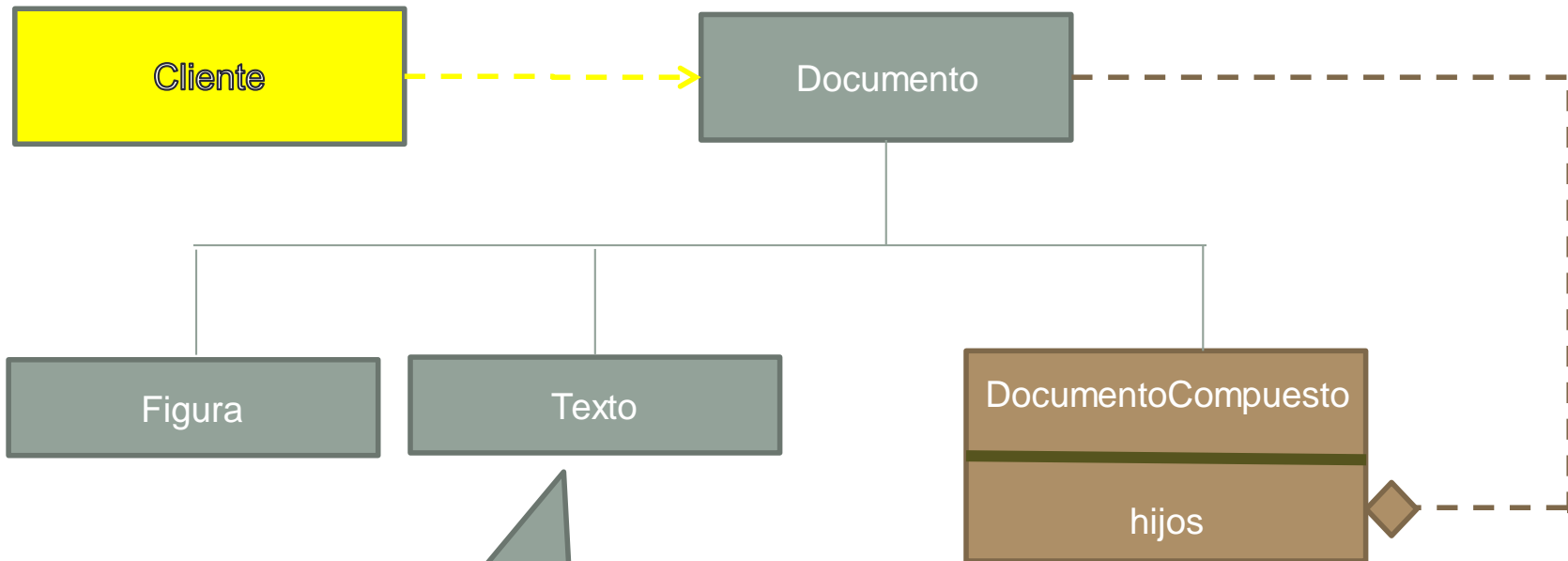
Composite - Estructura



Composite - Estructura

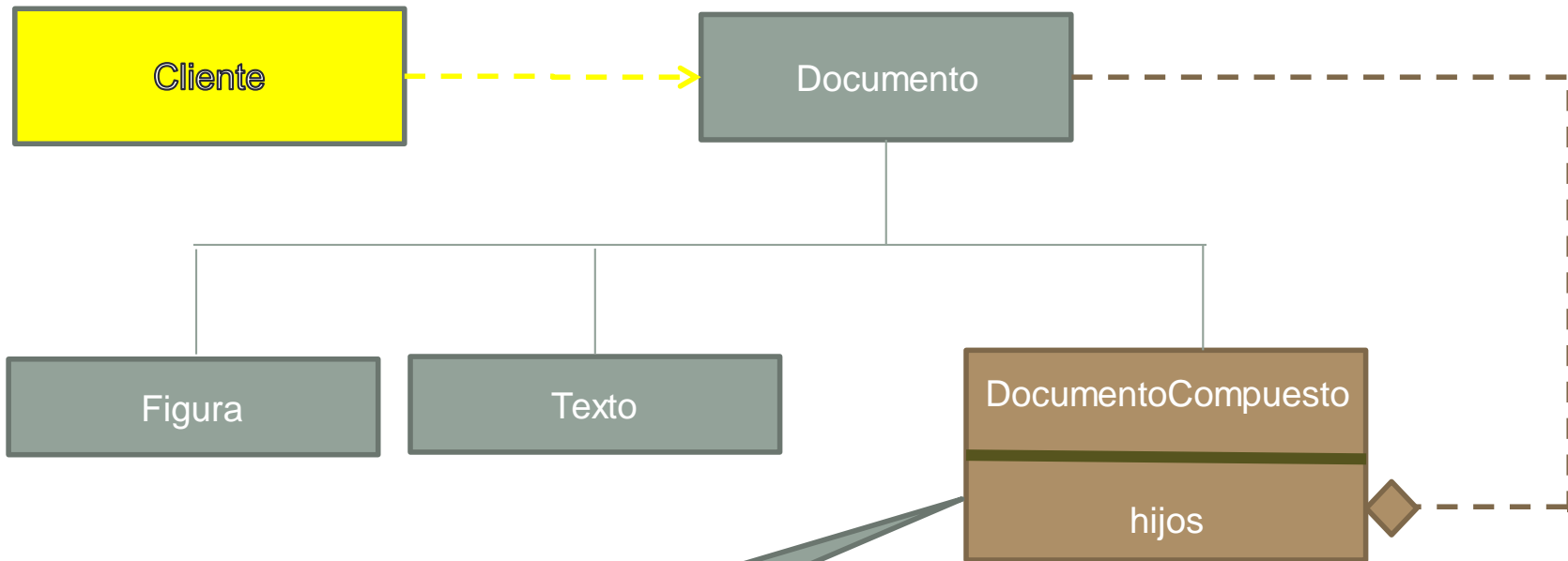


Composite - Estructura



Los "componentes hoja" serán los textos, figuras, tablas, ejemplos, citas todos los demás objetos "sencillos" que aparezcan en un futuro

Composite - Estructura



Un *DocumentoCompuesto* puede estar compuesto por textos, figuras, tablas y otros documentos compuestos

Composite - Implementación

```
abstract class Documento
    string titulo

    abstract void leer(filename)
    abstract void escribir(filename)
    abstract void imprimir(impresora)
    . . .
```

Una clase abstracta (o interface) define todos los métodos que debe realizar cualquier documento

Composite - Implementación

```
class Texto : Documento

    void leer(filename)
        // lectura del texto

    void escribir(filename)
        // escritura del texto

    void imprimir(impresora)
        // impresión del texto
```

Todos los componentes hojas saben como leerse, escribirse e imprimirse.

Composite - Implementación

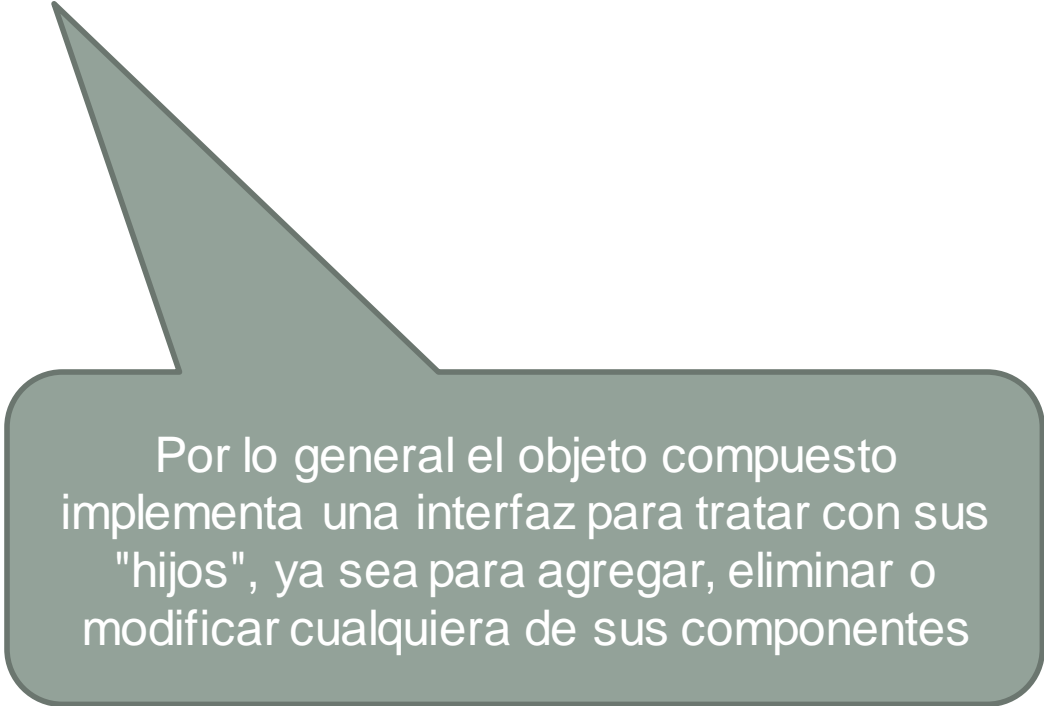
```
class DocumentoCompuesto : Documento
    hijos = new List<Documento>()

    void agregarHijo(Documento d)
        hijos.add(d)
```

Composite - Implementación

```
class DocumentoCompuesto : Documento
    hijos = new List<Documento>()

    void agregarHijo(Documento d)
        hijos.add(d)
```



Por lo general el objeto compuesto implementa una interfaz para tratar con sus "hijos", ya sea para agregar, eliminar o modificar cualquiera de sus componentes

Composite - Implementación

```
class DocumentoCompuesto : Documento
    . . .

    void leer(filename)
        foreach(d in hijos)
            d.leer(filename)
```

Composite - Implementación

```
class DocumentoCompuesto : Documento
    . . .

    void leer(filename)
        foreach(d in hijos)
            d.leer(filename)
```

Por lo general un objeto compuesto solo "transmite" el mensaje recibido a todos sus "hijos" y no hace nada más

Composite – Ventajas

- Define jerarquías formadas por objetos compuestos y primitivos. Los objetos compuestos pueden componerse de objetos, que a su vez pueden ser compuestos.
- Simplifica el cliente. Los clientes pueden tratar de manera uniforme a las estructuras compuestas y a los objetos individuales.
- Facilita añadir nuevos tipos de componentes. Si se añaden nuevas subclases de Compuesto u hoja, estas funcionarán de manera automática con las estructuras y componentes existentes.

Composite – Desventajas

- Puede hacer un diseño demasiado general. La desventaja de este patrón es que hace difícil restringir los componentes de un compuesto.
- A veces es necesario que un compuesto solo tenga ciertos componentes. Composite no garantiza esto.