Clase 4 - Patrones de diseño

Un patron proporciona una descripcion abstracta de un problema de diseño y como lo resuelve. Cada patron describe un problema que ocurre una y otra vez en nuestro entorno, asi como la solucion a ese problema, de tal modo que se pueda aplicar esta solucion un millon de veces sin hacer lo mismo dos veces. Las soluciones se expresan en terminos de objetos e interfaces. En general, un patron tiene cuatro elementos esenciales:

1. Nombre del patron: Permite describir con una o dos palabras un problema de diseño junto con sus soluciones y consecuencias.
2. El problema: Describe cuando aplicar el patron. Explica el problema y su contexto. Puede describir problemas completos de diseño (por ejemplo como representar algoritmos como objetos). Aveces, el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patron
3. La solucion: Describe los elementos que constituyen el diseño, sus relaciones,responsabilidades y colaboraciones. La solucion no describe un diseño o una implementacion en concreto, sino que un patron es como una plantilla que puede aplicarse en muchas situaciones diferentes.
4. Las consecuencias: Son los resultados asi como las ventajas e inconvenientes de aplicar el patron. Las consecuencias en el software suelen referirse entre espacio y tiempo.

Aspectos importantes:

Un patron, es un par problema-solucion
Los patrones tratan con problemas recurrentes y buenas soluciones (probadas) a esos problemas.
La solucion es suficientemente generica para poder aplicarse de diferentes maneras.

Patrones que vimos en las teorias:

Patrón Adapter:

Permite que dos clases con interfaces incompatibles trabajen juntas. El Adapter, actua como un puente entre la clase existente y una nueva interfaz que se necesita.

🧠 ¿Cómo funciona?

El Adapter:

- **Recibe llamadas** en una interfaz conocida.

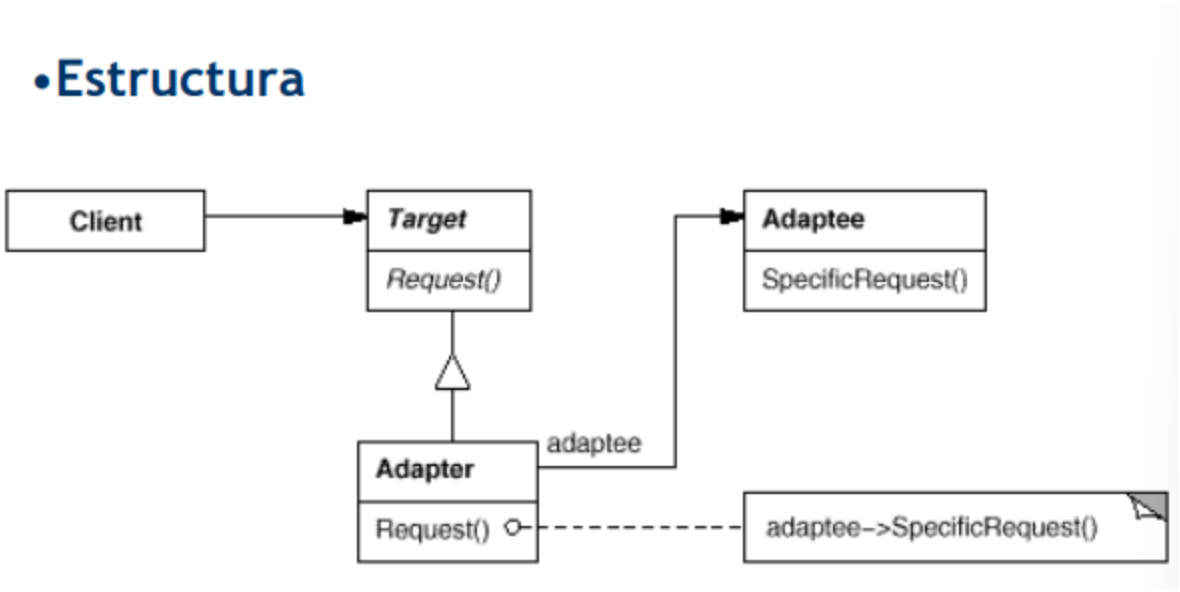
- **Traduce esas llamadas** a una forma que la clase existente entienda.
- **Devuelve los resultados**, si es necesario, en el formato que se espera.

Intencion:

Convertir la interfaz de una clase en otra que el cliente espera.
Adapter permite que ciertas clases con interfaces incompatibles puedan trabajar en conjunto.

Aplicabilidad:

Use Adapter cuando quiera usar una clase existente y su intefaz no es compatible con lp que precisa,



Target: Define la interfaz especifica que usa el cliente.
Client: Colabora con objetos que satisfacen la interfaz de Target
Adaptee: Define una interfaz que precisa ser adaptada
Adapter: Adapta la interfaz del Adaptee a la interfaz del Target,

Colaboraciones:

Los objetos Client llaman a las operaciones en la instancia del Adapter.
A su vez, el Adapter llama a las operaciones definidas en el Adaptee

Patron Template Method

Template Method permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar la estructura del algoritmo.

🧠 ¿En qué consiste?

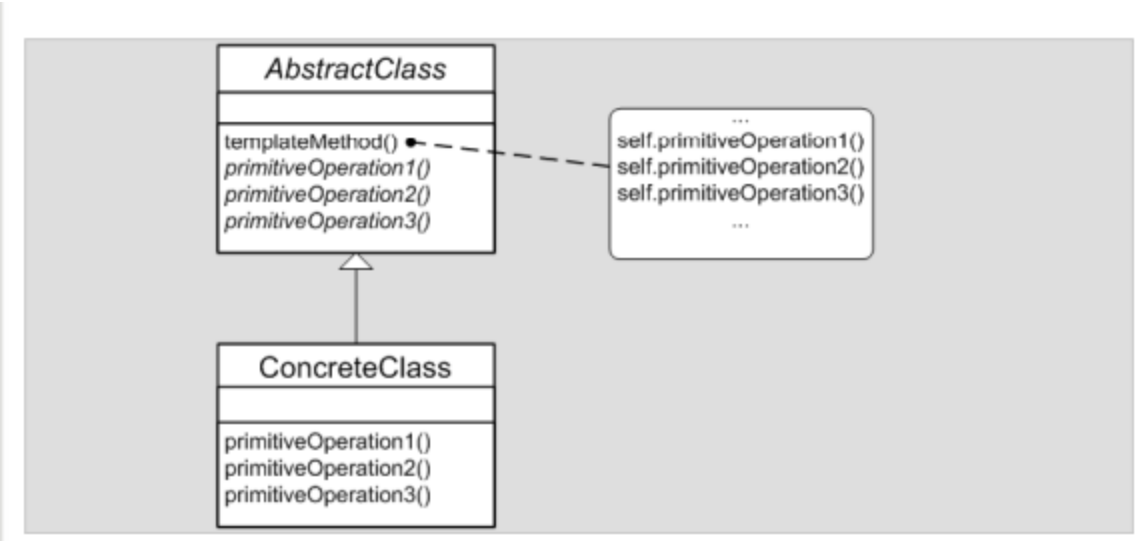
La idea es tener una **clase abstracta** que define un método que representa la **plantilla** del algoritmo. Este método llama a otros métodos (algunos pueden estar implementados y otros ser abstractos), y las **subclases personalizan** esos métodos según lo necesiten.

📌 ¿Cuándo usar Template Method?

- Cuando varios algoritmos comparten pasos similares pero difieren en algunos detalles.
- Cuando querés evitar duplicar código entre clases que siguen un proceso similar.

Aplicabilidad: Usar template method

- Para evitar duplicar codigo
- Para controlar las extensiones que pueden hacer las subclases
- Para implementar las partes invariantes de un algoritmo una vez y dejar que las subclases implementen los aspectos que varian.



Participantes:

- Implementa un metodo que contiene el esqueleto de un algoritmo (el template method)-
- Declara operaciones primitivas abstractas que las subclases concretas deben definir para implementar los pasos de un algoritmo.

Colaboraciones:

- ConcreteClass confia en que la superclase implemente las partes invariantes del algoritmo.

Consecuencias:

- Tecnica fundamental para reusar codigo.

- Lleva a tener inversion de control (la superclase llama a las operaciones definidas en las subclases)