¡Hola Mundo!

Si eres programador o estás despegando en el desarrollo informático y la programación, tarde o temprano te toparás con el término **patrones de diseño de software**. El desarrollo de software es un proceso complejo que requiere un enfoque estructurado y eficiente para lograr resultados óptimos.

En este contexto, los patrones de diseño de software se han convertido en una herramienta invaluable para los desarrolladores, brindando soluciones probadas y comprobadas para problemas comunes de diseño.

¿Qué es un patrón de diseño?

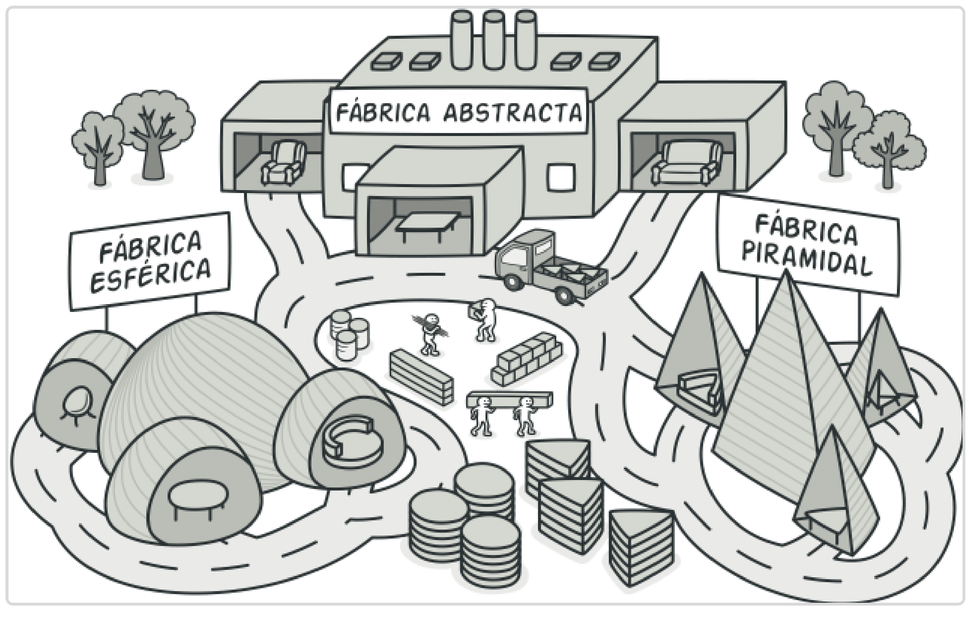
Son elementos reutilizables creados para resolver problemas comunes, es decir que con su aplicación podremos corregir diferentes problemas que presenta nuestro código de una manera segura, estable y testeada. Pueden clasificarse por su propósito:

- Los patrones **creacionales** proporcionan mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización de código existente.

- Los patrones **estructurales** explican cómo ensamblar objetos y clases en estructuras más grandes a la vez que se mantiene la flexibilidad y eficiencia de la estructura.

- Los patrones de **comportamiento** se encargan de una comunicación efectiva y la asignación de responsabilidades entre objetos.

Fábrica Abstracta



Próposito

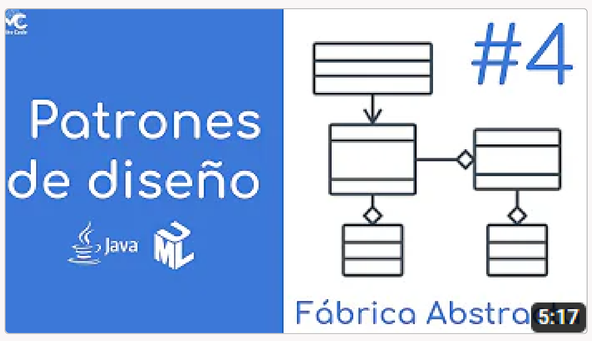
Es un patrón de diseño creacional que nos permite producir familias de objetos relacionados sin especificar sus clases concretas.

Pros ✓

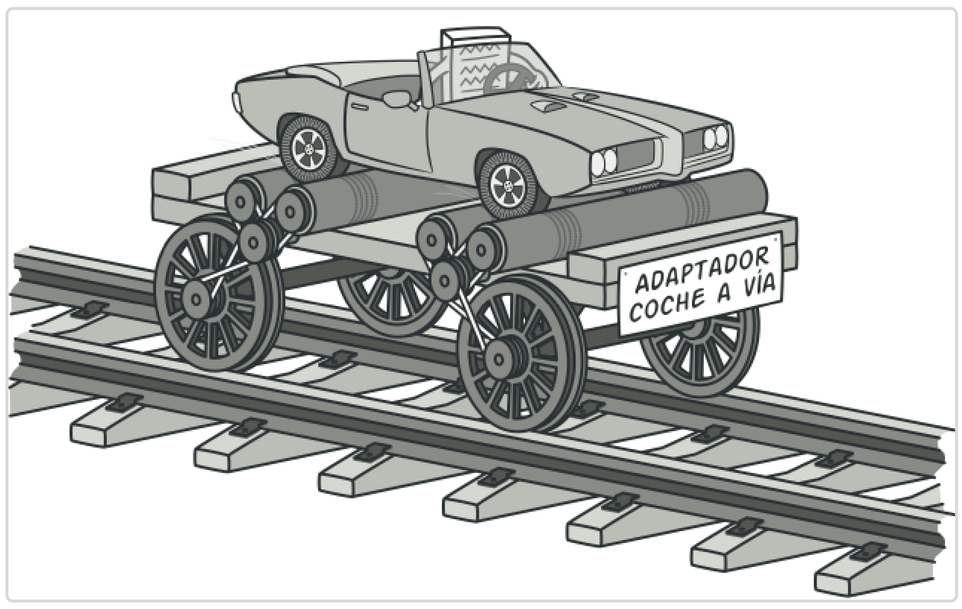
- Puedes tener la certeza de que los productos que obtienes de una fábrica son compatibles entre sí.
- Evitas un acoplamiento fuerte entre productos concretos y el código cliente.

Contras ✗

- Puede ser que el código se complique más de lo que debería, ya que se introducen muchas nuevas interfaces y clases junto al patrón.



Adaptador

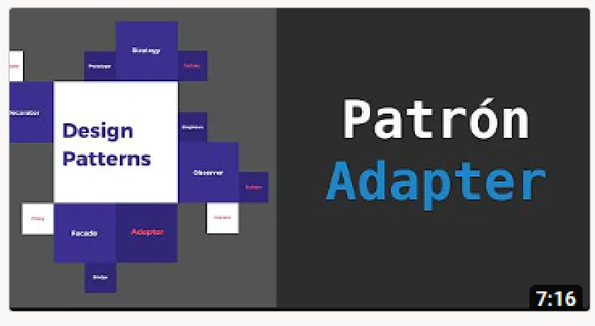


Próposito

Es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.

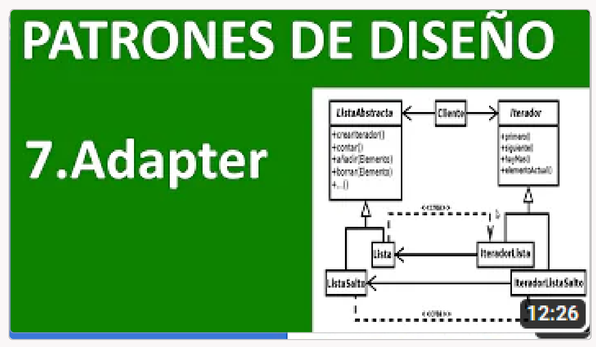
Pros ✓

- Principio de responsabilidad única. Puedes separar la interfaz o el código de conversión de datos de la lógica de negocio primaria del programa.



Contras ✗

- La complejidad general del código aumenta, ya que debes introducir un grupo de nuevas interfaces y clases.



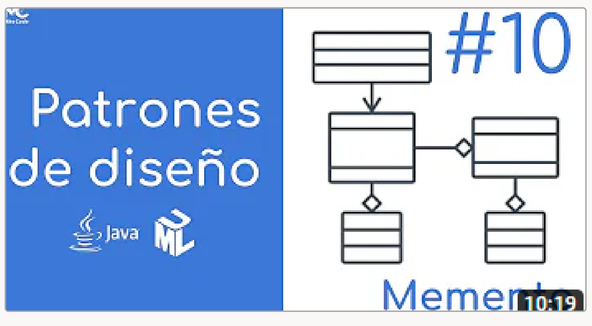
Recuerdo

Próposito

Es un patrón de diseño de comportamiento que te permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.

Pros ✓

- Puedes simplificar el código de la originadora permitiendo que la cuidadora mantenga el historial del estado de la originadora.



Contras ✗

- La aplicación puede consumir mucha memoria RAM si los clientes crean mementos muy a menudo.