Patrones de diseño

# Patrones de creación

Builder

* Puede construir objetos paso a paso, aplazar los pasos de construcción o ejecutar pasos de forma recursiva.
* Puede reutilizar el mismo código para crear varias instancias de objetos.
* *Principio de responsabilidad única* . Puede aislar el código de construcción complejo de la lógica empresarial del producto.

Factory

* Evitas el estrecho acoplamiento entre el creador y los productos concretos.
* *Principio de responsabilidad única* . Puede mover el código de creación del Objeto a un lugar en el programa, lo que facilita el soporte del código.
* *Principio abierto/cerrado* . Puede introducir nuevos tipos de objetos en el programa sin romper el código existente.

Prototype

* Puede clonar objetos sin acoplarlos a sus clases concretas.
* Puede deshacerse de repetir la creación de instancias en favor de la clonación de prototipos prefabricados.
* Puede producir objetos complejos de manera más conveniente.
* Obtiene una alternativa a la herencia cuando se trata de ajustes preestablecidos de configuración para objetos complejos.

## Patrones estructurales

Adapter

* *Principio de responsabilidad única* . Puede separar la interfaz o el código de conversión de datos de la lógica comercial principal del programa.
* *Principio abierto/cerrado* . Puede introducir nuevos tipos de adaptadores en el programa sin romper el código de cliente existente, siempre que funcionen con los adaptadores a través de la interfaz del cliente.

Decorador

* Puede ampliar el comportamiento de un objeto sin crear una nueva subclase.
* Puede agregar o eliminar responsabilidades de un objeto en tiempo de ejecución.
* Puede combinar varios comportamientos envolviendo un objeto en varios decoradores.
* *Principio de responsabilidad única* . Puede dividir una clase monolítica que implementa muchas variantes posibles de comportamiento en varias clases más pequeñas

Facade

* Puede aislar su código de la complejidad de un subsistema.