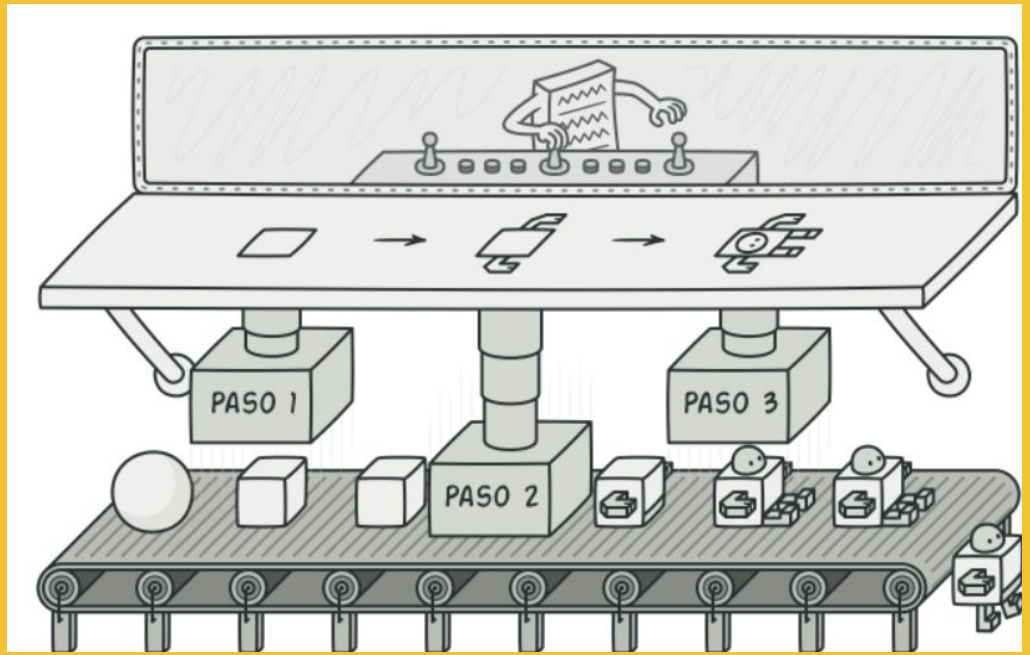
Patrón de diseño Builder

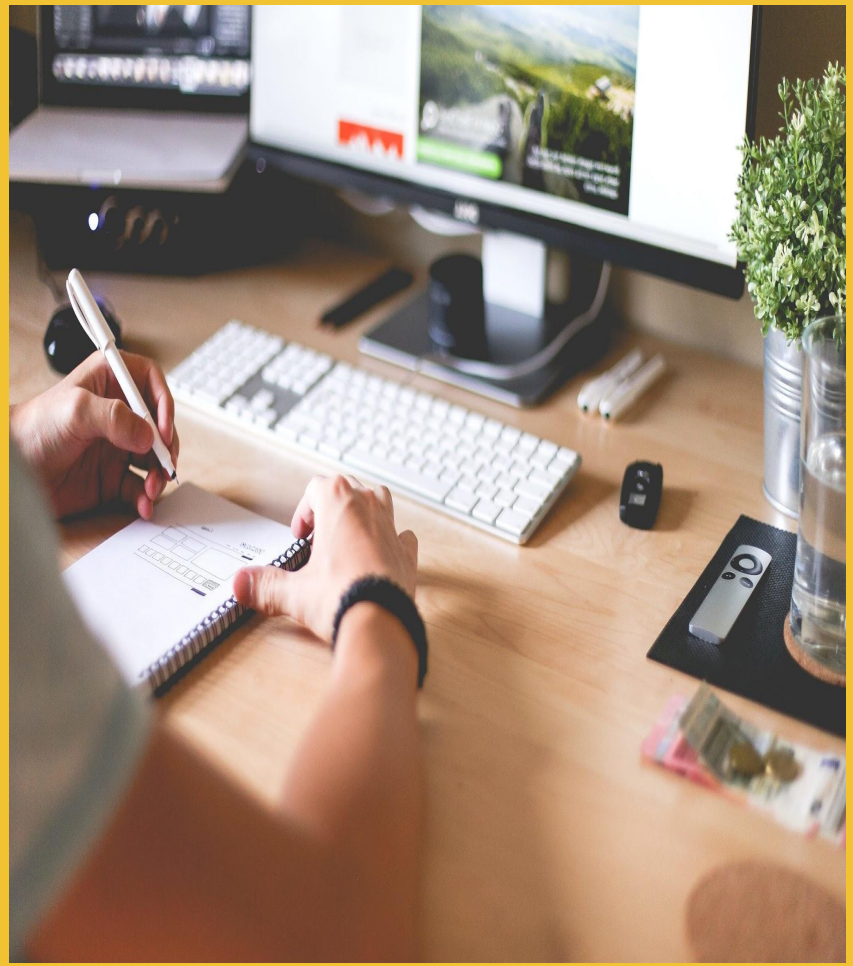
Jordano Escalante



- Los patrones de diseño buscan solucionar problemas típicos en la forma en la que se diseña un programa o funcionalidad.

- Buscan también la reutilización del código.

- Generalmente resuelven problemas con un alto grado de complejidad.



- También buscan establecer soluciones comunes para problemas comunes, de manera que un problema (como lo es la creación de muchos objetos complejos) tengan una solución conocida que pueda implementarse independientemente del contexto.

- Evitar la creación de la rueda (no hace falta volver a idear soluciones nuevas para cada problema).

- Establecer un estándar.

Plantillas

Cada patrón cuenta con una plantilla que es común en estructura con otros patrones, solamente cambia su contenido, esto con el fin de mantener un estándar que facilite el uso y comprensión de los patrones.

Nombre

Clasificación.

Motivación.

Usos.

Estructura.

Participantes.

Colaboración entre participante.

Consecuencias.

Implementación.

Código.

Ejemplos de uso.

Patrones relacionados.

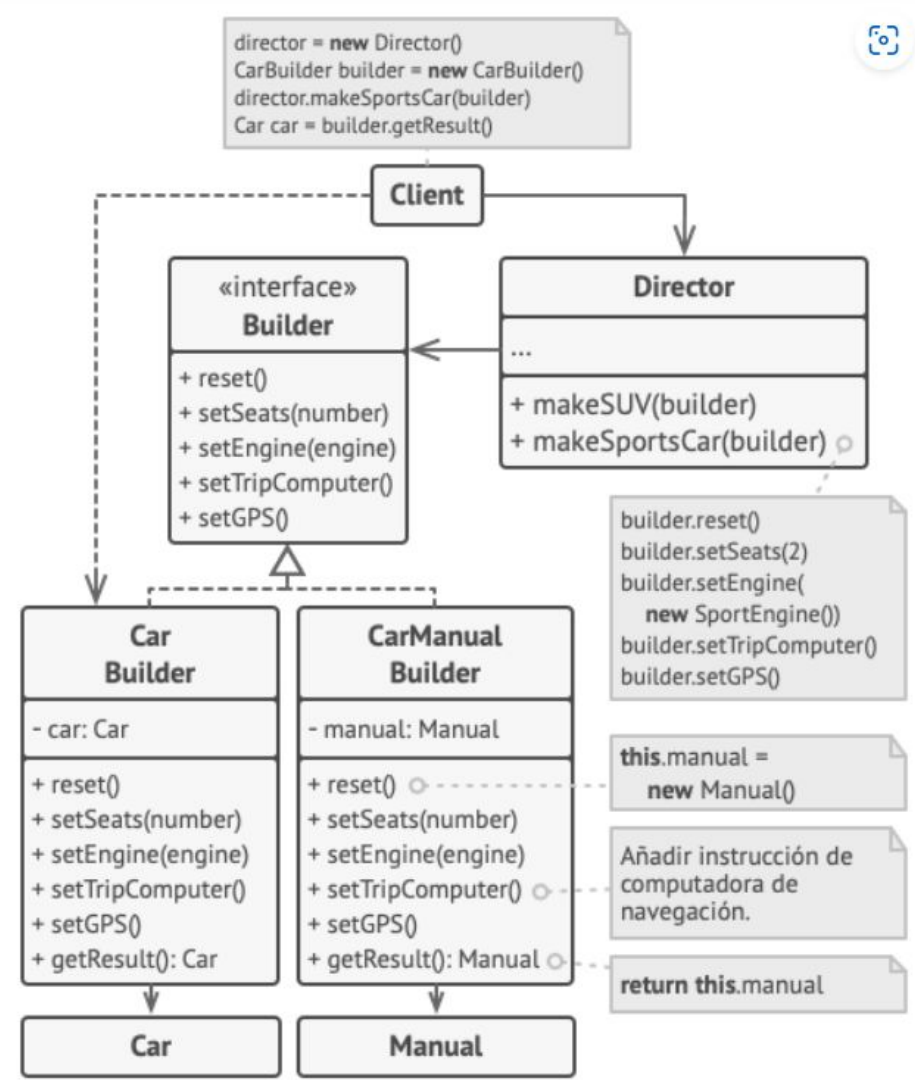
Partes del Patrón Builder

Director: Utiliza la interfaz del constructor para crear un objeto siguiendo los pasos en el orden específico que se requiere.

Builder: Provee la funcionalidad para crear el objeto complejo en el orden apropiado.

Constructor específico: Contiene los detalles de implementación del cómo se construye el objeto.

Producto: Es el objeto concreto creado.



UML

Director: Crear los objetos llamando al builder respectivo.

Builder: Es la interface que define el orden en que se deben crear los elementos del objeto.

Builder concreto: Define los detalles de implementación.

Objeto: Producto.

¿Cuando usar el patrón?

Es conveniente utilizarlo en casos donde se deben crear varias instancias de objetos, usando el builder se ahorra tener que construir cada objeto parámetro por parámetro, también es conveniente usarlo en casos donde los constructores tienen un gran número de atributos.

Class Pizza	**VeggiePizza**
pepperoni	no
queso	yes
salsa	yes

Ventajas

- Reutilización de código.
- Simplificación en la creación de objetos.
- Estandarización de procesos.
- Single responsibility.

DesVentajas

- Nivel de complejidad alto.
- Alta cohesión que puede dificultar el realizar cambios.


"That's all Folks!"