Diseño de Software

BUILDER

PATRÓN DE DISEÑO

Eduardo Sancho Selva

¿QUÉ ES BUILDER?

Patrón de diseño **creacional**

Construye objetos complejos **paso a paso**

Permite crear diferentes representaciones de un producto con el mismo código de construcción base

PROBLEMA

Objetos con atributos opcionales o pasos de construcción complejos

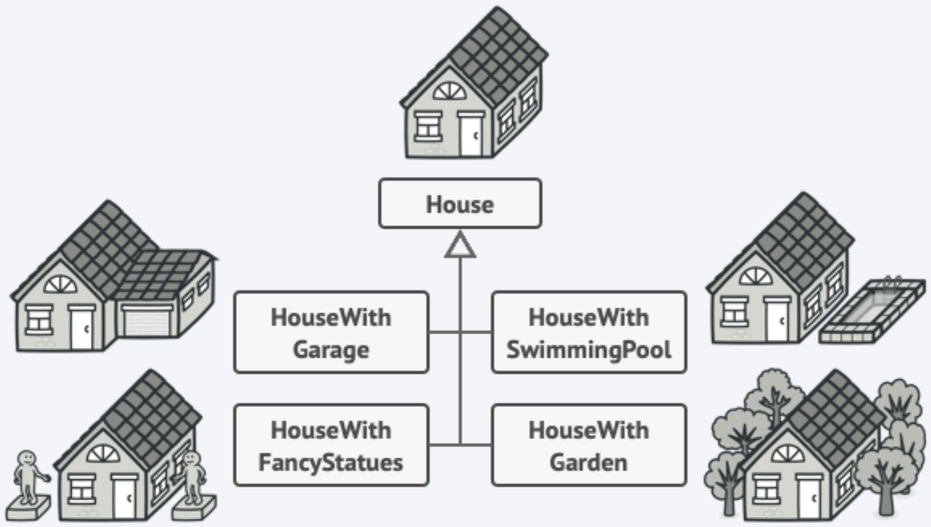
Ejemplo: Objeto Casa

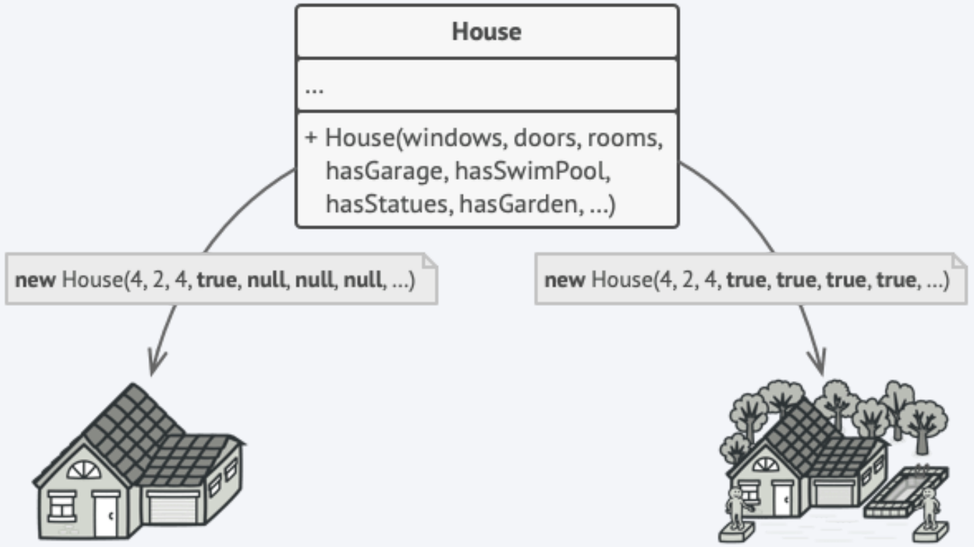
Atributos:

- Paredes
- Puertas
- Ventanas
- Pisos

¿PERO... CUÁL DE TODAS?

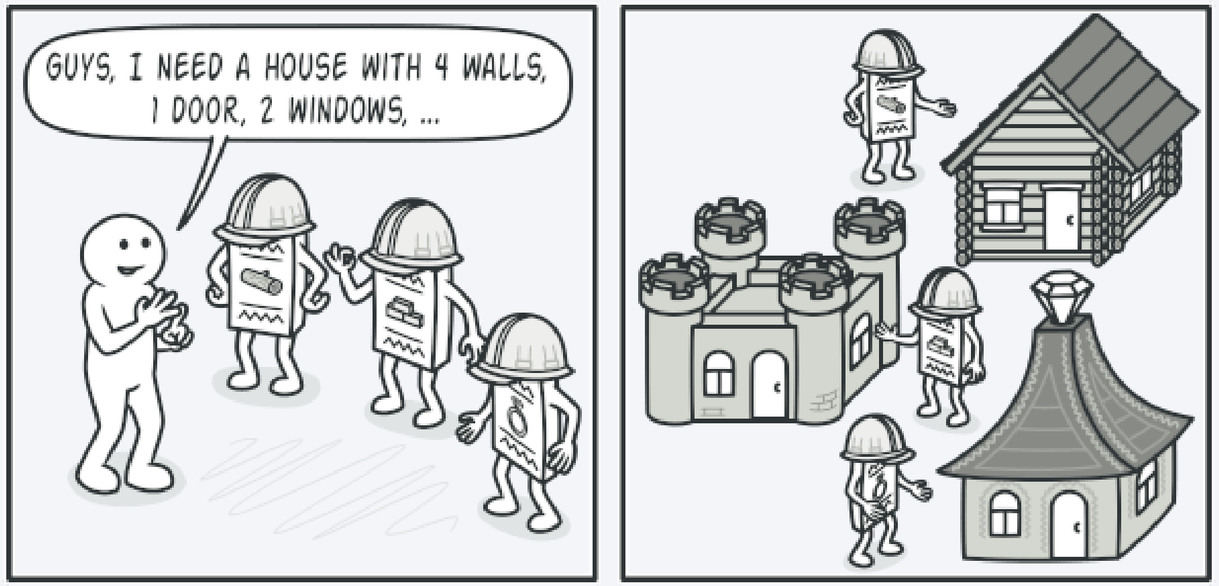
SOLUCIONES (SIN BUILDER)





SOLUCIÓN:

Separación de asuntos



PRODUCTO	BUILDER	BUILDER CONCRETO
class House El objeto complejo que se quiere crear	**class HouseBuilder** Base del constructor Tiene todos los pasos que podrían usarse em una casa Solo se implementan los pasos necesarios	**class WoodenHouseBuilder, StoneHouseBuilder...** Implementaciones concretas, con detalles distintos

EXTRA: DIRECTOR

Trabaja con un constructor
Define el orden de implementación de los pasos de construcción

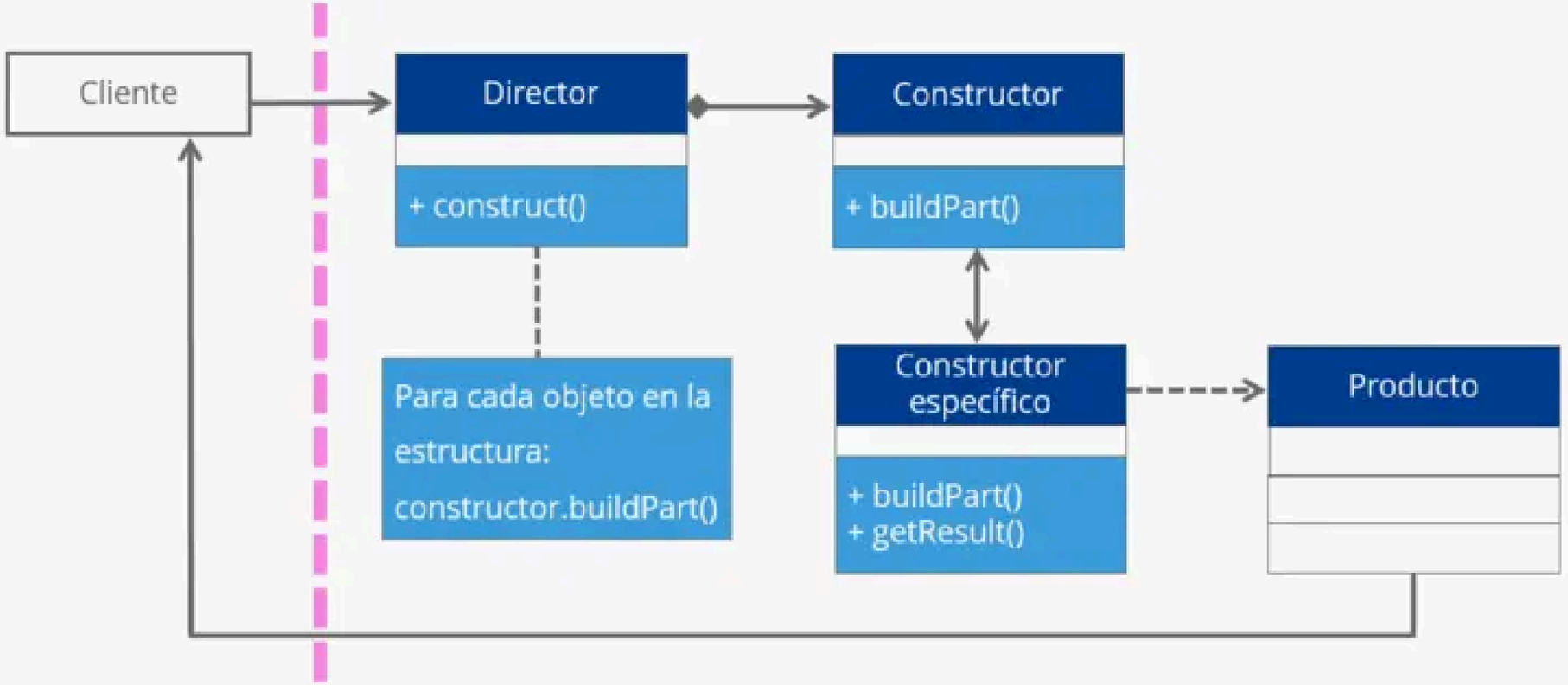
```
class Director is
    // The director works with any builder instance that the
    // client code passes to it. This way, the client code may
    // alter the final type of the newly assembled product.
    // The director can construct several product variations
    // using the same building steps.
    method constructSportsCar(builder: Builder) is
        builder.reset()
        builder.setSeats(2)
        builder.setEngine(new SportEngine())
        builder.setTripComputer(true)
        builder.setGPS(true)

    method constructSUV(builder: Builder) is
        // ...
```

Nuestros Valores

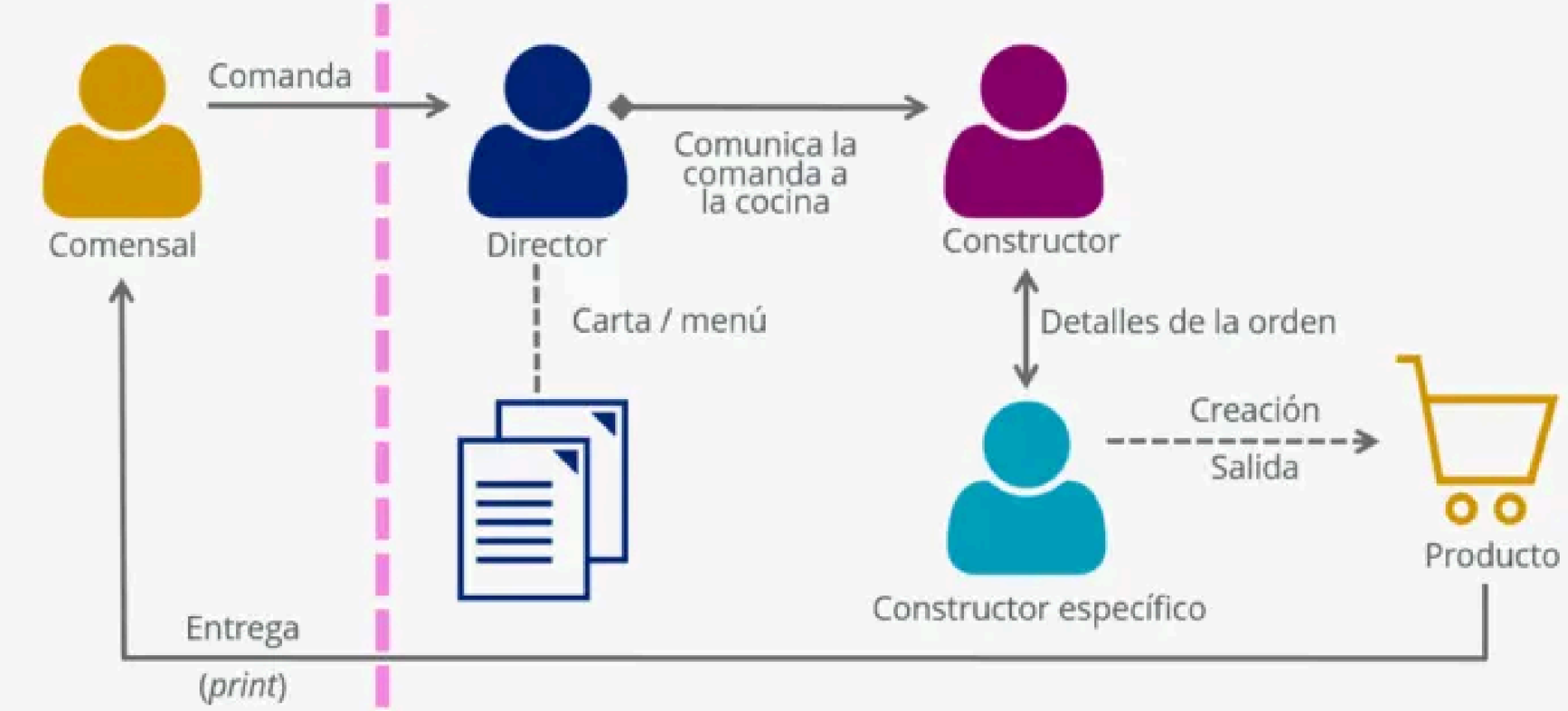
Lorem ipsum dolor sit amet, consectetur risus mollis adipiscing elit. Pellentesque cursus risus ut nulla elementum gravida. Vivamus mollis risus mauris nunc.

Diagrama UML: Builder Pattern



IONOS

La mecánica de Builder Pattern con un ejemplo



IONOS

VENTAJAS

01 Facilita la construcción de objetos complejos que pueden tener parámetros y métodos de implementación muy distintos

02 Claridad y legibilidad de código, al una interfaz de constructor clara y métodos descriptivos

03 SRP: separa la construcción de la representación
OCP: permite agregar variantes del objeto
DIP: dependencia a clase abstracta Builder

DESVENTAJAS

01 Muchas clases pequeñas, lo que puede aumentar la complejidad del código

02 *Overkill* para objetos pequeños y simples, puede agregar complejidad innecesaria

EJEMPLO EN CÓDIGO

Gracias