POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

8. *Builder*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

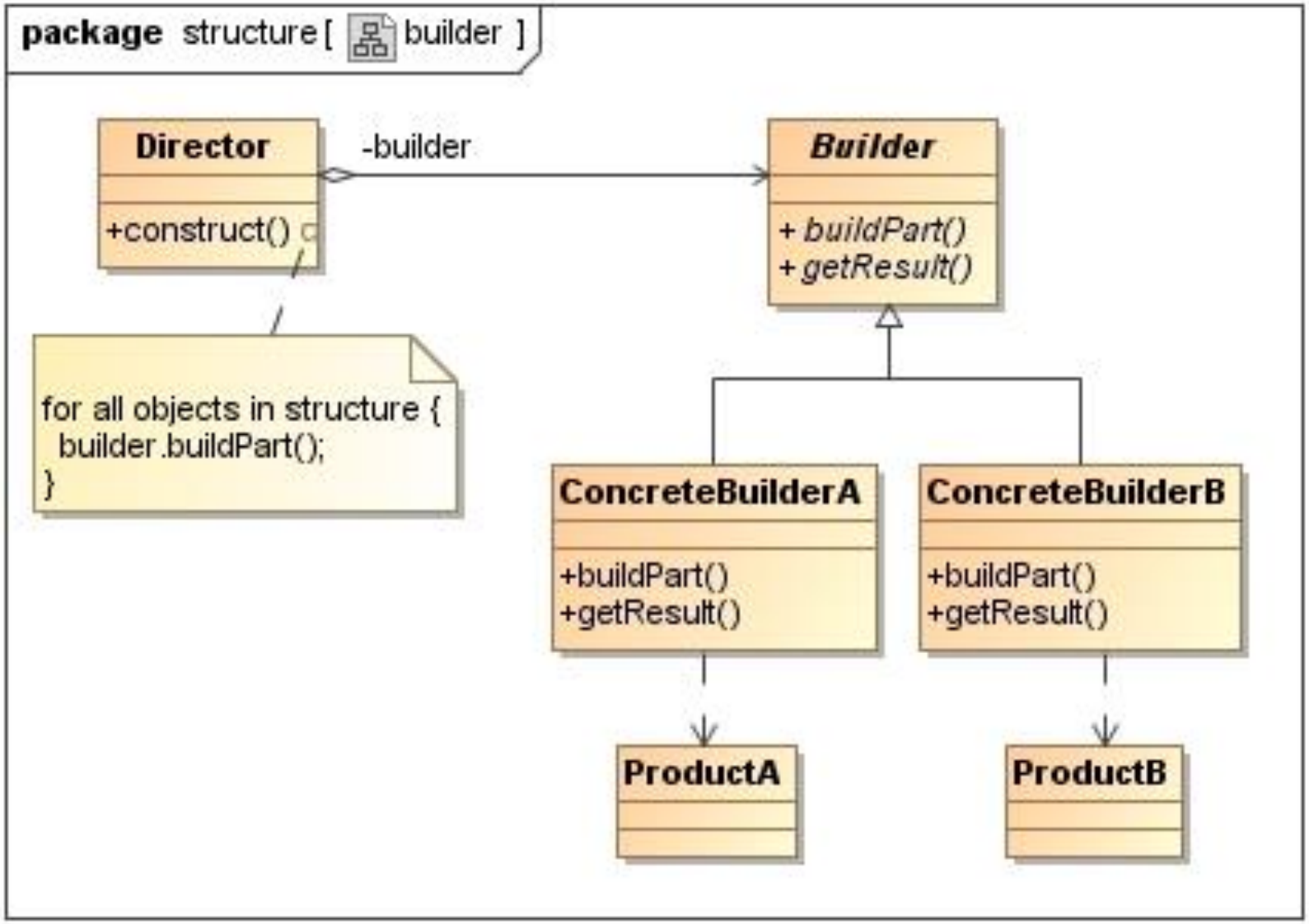
- *Antiguamente, el hombre construía los productos que él mismo producía (prosumidor, sabía hacer de "todo": p.e. pan, pantalones, …) y el intercambio no era fácil (yo quiero lo que tu tienes, tu quieres lo que él tiene y el quiere lo que yo tengo).*
- *Todo era muy complicado ("nadie sabe hacer de todo") e inflexible en sus colaboraciones (sin libre comercio, exportación, importación, …)*
- *Desde que el hombre dejó de ser prosumidor, delega la construcción de sus productos de consumo (p.e. en tiendas, en empresas, …) de tal forma que el mismo proceso de construcción (p.e. ahora, ir de compras) puede crear diferentes productos (pan integral, pantalones largos, …) de forma sencilla y flexible (p.e. nuevos productos en tiendas, …).*

- ***Separa la construcción de un objeto complejo de su representación de tal forma que el mismo proceso de construcción puede crear diferentes representaciones***

1. Problema

- Utilícese cuando:
  - el algoritmo para la creación de un objeto complejo debe ser independiente de las partes que componen el objeto y la forma en que están montadas.
  - el proceso de construcción debe permitir diferentes representaciones para el objeto que se construye

2. Solución


package structure [builder]
Director
-builder
Builder
+construct()
+ buildPart()
+ getResult()
for all objects in structure {
builder.buildPart();
}
ConcreteBuilderA
+buildPart()
+getResult()
ConcreteBuilderB
+buildPart()
+getResult()
ProductA
ProductB

2. Solución

- Por lo general, hay una clases abstracta *Builder* que define una operación para cada componente que un director puede pedirle crear. Las operaciones no hacen nada por defecto. Una clase *Builder* concreta redefine las operaciones de los componentes en los que está interesado para su creación.
- Los constructores construyen sus productos paso a paso. Por tanto, la interfaz de la clase *Builder* debe ser lo suficientemente general como para permitir construir productos por parte de todos los tipos de constructores concretos. A veces podríamos necesitar acceder a las partes del producto que ya fueron construidas. Con las estructuras arbóreas que se crean de abajo a arriba, el constructor devolvería nodos hijos al director, el cual los devolvería al constructor para construir los nodos padre.

2. Solución

- En general, los productos creados por los constructores concretos tiene representaciones tan diferentes que sería de poca ayuda definir una clase padre común para los diferentes productos. Como el cliente suele configurar al director con el ConstructorConcreto adecuado, sabe qué subclase concreta de *Builder* se está usando y puede manejar sus productos en consecuencia.

3. Consecuencias

- Se proporciona al director una interfaz abstracta para la construcción del producto. La interfaz permite al *Builder* ocultar la representación y la estructura interna del producto. También oculta cómo el producto se ensambla. Debido a que el producto se construye a través de una interfaz abstracta, todo lo que tiene que hacer para cambiar la representación interna del producto es definir un nuevo tipo de constructor.
- Mejora la modularidad encapsulando la forma en que un objeto complejo es construido y representado. Los clientes no necesitan saber nada acerca de las clases que definen la estructura interna del producto; estas clases no aparecen en la interfaz del constructor.

3. Consecuencias

- Construye el producto paso a paso bajo el control del director. Sólo cuando el producto está terminado, el director lo recupera desde el constructor. Por lo tanto, la interfaz del constructor refleja el proceso de construcción del producto. Esto le da un mayor control sobre el proceso de construcción y, por lo tanto, de la estructura interna del producto resultante.

4. Relaciones

- ***Singleton*** para un único *Builder*
- ***Composite*** como resultado del *Builder*
- ***Prototype*** y ***Abstract Factory*** para implementar qué componentes debe construir