|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | Diseño | | |
| AREA | Software | REGIMEN | Semestral |

**Tema: Patrones de diseño**

**Builder**

Permite la creación de un objeto complejo, a partir de una variedad de partes que contribuyen individualmente a la creación y ensamblación del objeto mencionado.

Hace uso de la frase "divide y conquistarás".

Por otro lado, centraliza el proceso de creación en un único punto, de tal forma que el mismo proceso de construcción pueda crear representaciones diferentes.  
  
Los objetos que dependen de un algoritmo tendrán que cambiar cuando el algoritmo cambia. Por lo tanto, los algoritmos que estén expuestos a dicho cambio deberían ser separados, permitiendo de esta manera reutilizar dichos algoritmos para crear diferentes representaciones.  
  
Se debe utilizar este patrón cuando sea necesario:

* Independizar el algoritmo de creación de un objeto complejo de las partes que constituyen el objeto y cómo se ensamblan entre ellas.
* Que el proceso de construcción permita distintas representaciones para el objeto construido, de manera dinámica.

Esta patrón debe utilizarse cuando el algoritmo para crear un objeto suele ser complejo e implica la interacción de otras partes independientes y una coreografía entre ellas para formar el ensamblaje. Por ejemplo: la construcción de un objeto Computadora, se compondrá de otros muchos objetos, como puede ser un objeto PlacaDeSonido, Procesador, PlacaDeVideo, Gabinete, Monitor, etc.

Diagrama UML

A diagram of a building

Description automatically generated

Producto: representa el objeto complejo a construir.  
Builder: especifica una interface abstracta para la creación de las partes del Producto. Declara las operaciones necesarias para crear las partes de un objeto concreto.  
ConcreteBuilder: implementa Builder y ensambla las partes que constituyen el objeto complejo.  
Director: construye un objeto usando la interfaz Builder. Sólo debería ser necesario especificar su tipo y así poder reutilizar el mismo proceso para distintos tipos.  
  
El Cliente crea el objeto Director y lo configura con el objeto Builder deseado.  
El Director notifica al constructor cuándo una parte del Producto se debe construir.  
El Builder maneja los requerimientos desde el Director y agrega partes al producto.  
El Cliente recupera el Producto desde el constructor.  
  
Ejemplo  
  
El objetivo del ejemplo es poder crear un objeto Auto (este sería nuestro producto). El auto se compondrá de varios atributos que lo componen: motor, marca, modelo y cantidad de puertas. En nuestro ejemplo, el auto no se compone de muchos objetos complejos. De hecho, se compone de sólo 4 objetos relativamente sencillos. Esto es para poder hacer entendible la propuesta del Builder y no perderse en los objetos que lo componen. Queda en la imaginación del lector la posibilidad de trabajar con ejemplos más complejos. Yo particularmente usé mucho este patrón cuando trabajé con archivos.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

Siguiendo con nuestro ejemplo, definimos nuestro Builder llamado AutoBuilder. El Builder define al menos dos cosas: un método para devolver el Producto (el auto en nuestro caso) y los métodos necesarios para la construcción del mismo.

A screenshot of a computer code

Description automatically generated

Serán los ConcreteBuilders los encargados de colocarle la lógica de construcción de cada Auto en particular. En nuestro caso, tendremos dos ConcreteBuilder: FiatBuildery FordBuilder. Recordemos que, en nuestro ejemplo, son clases que construyen objetos muy sencillos con datos hardcodeados para facilitar el aprendijaze del patrón en sí.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Por último, realizaremos el Director. Lo primero que debe hacerse con esta clase es enviarle el tipo de auto que se busca construir (Ford, Fiat, etc). Luego, al llamar al método constructAuto(), la construcción se realizará de manera automática.

A screenshot of a computer code

Description automatically generated

La invocación desde un cliente sería:

A screenshot of a computer program

Description automatically generated

Consecuencias

* Permite variar la representación interna de un producto.
* El Builder ofrece una interfaz al Director para construir un producto y encapsula la representación interna del producto y cómo se juntan sus partes.
* Si se cambia la representación interna basta con crear otro Builder que respete la interfaz.
* Separa el código de construcción del de representación.
* Las clases que definen la representación interna del producto no aparecen en la interfaz del Builder.
* Cada ConcreteBuilder contiene el código para crear y juntar una clase específica de producto.
* Distintos Directores pueden usar un mismo ConcreteBuilder.
* Da mayor control en el proceso de construcción.
* Permite que el Director controle la construcción de un producto paso a paso.
* Sólo cuando el producto está acabado lo recupera el director del builder.