Patrones de Diseño: Patrones de Creación. Tema 3-3: Builder

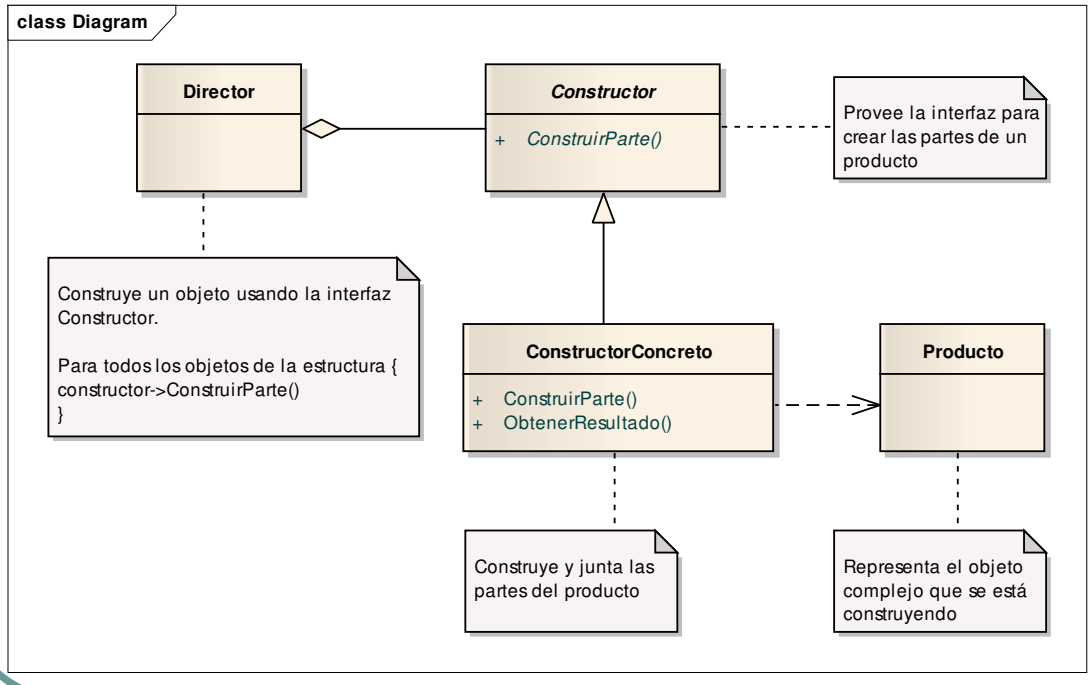
Descripción del patrón

- **Nombre:**
  - Constructor
- **Propiedades:**
  - Tipo: creación
  - Nivel: objeto, componente
- **Objetivo o Propósito:**
  - Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

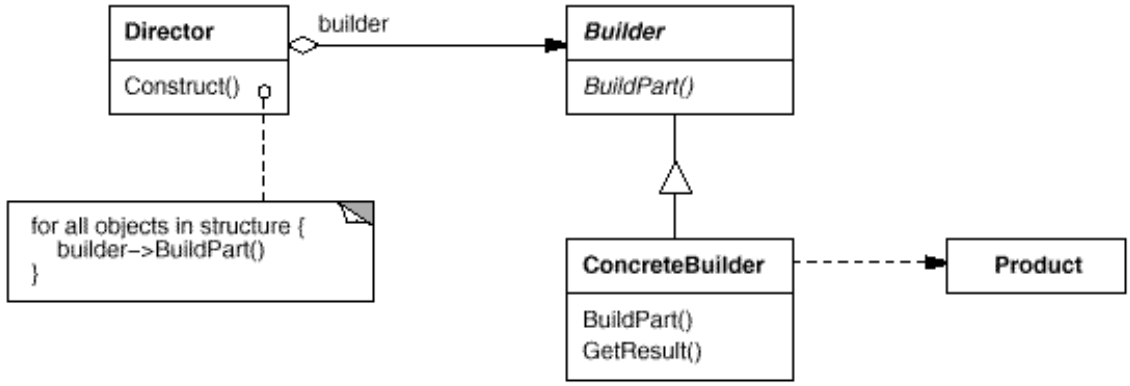
Aplicabilidad

- Use el patrón Builder cuando:
  - Una clase tiene una estructura compleja (si tiene un conjunto variable de objetos relacionados).
  - La clase tiene atributos dependientes entre si. Construcción por etapas.
  - El algoritmo para crear un objeto complejo debiera ser independiente de las partes de que se compone dicho objeto y de cómo se ensamblan.
  - El proceso de construcción debe permitir diferentes representaciones del objeto que está siendo construido.

Estructura



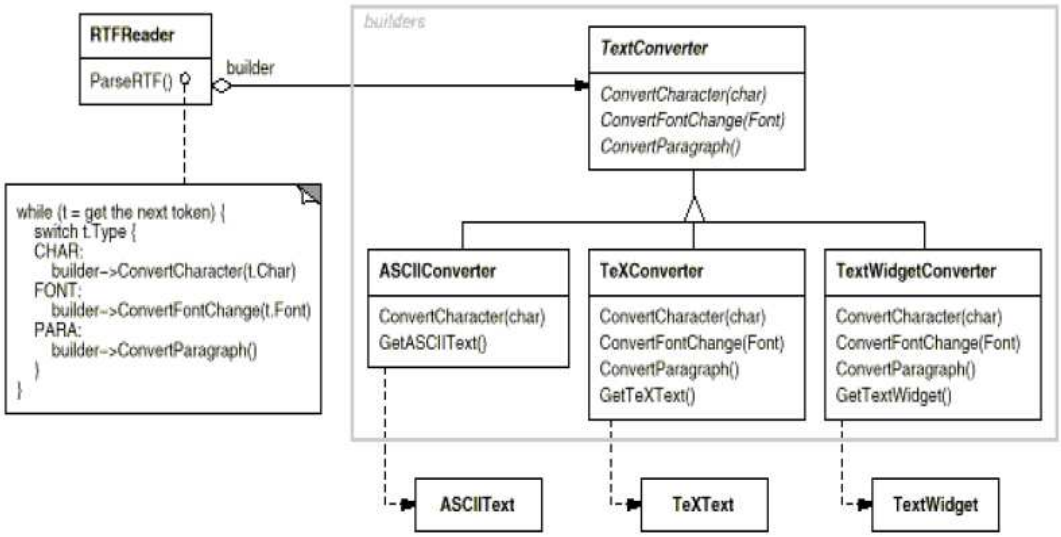
Estructura



Estructura. Participantes

- **Constructor**: Especifica una interfaz para crear las partes de un objeto.
- **ConstructorConcreto:** Implementa la interfaz Constructor para construir y ensamblar las partes del producto. Define la representación a crear. Proporciona un método que devuelve una instancia del producto.
- **Producto:** Representa el objeto complejo en construcción.
- **Director**: Construye un objeto usando la interfaz Constructor.
- **Cliente:** Crea el objeto Director y lo configura con el objeto Constructor deseado.

Estructura. Ejemplo



Estructura. Ejemplo

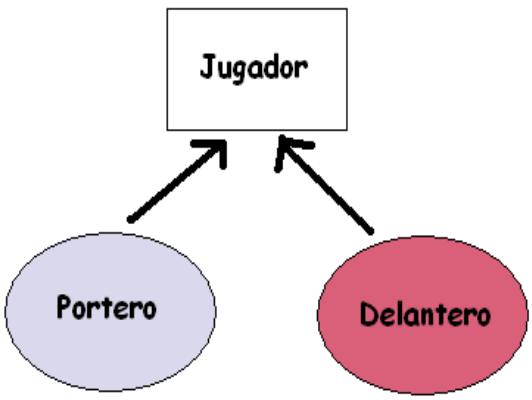
- Supongamos un editor que quiere convertir un tipo de texto (p.ej. RTF) a varios formatos de representación diferentes, y que puede ser necesario en el futuro definir nuevos tipos de representación.
- El lector de RTF (RTFReader) puede configurarse con una clase de Conversor de texto (TextConverter) que convierta de RTF a otra representación.
- A medida que el RTFReader lee y analiza el documento, usa el conversor de texto para realizar la conversión: cada vez que reconoce un token RTF llama al conversor de texto para convertirlo.
- Hay subclases de TextConverter para cada tipo de representación.
- El conversor (*builder*) está separado del lector (*director*): se separa el algoritmo para interpretar un formato textual de cómo se convierte y se representa.

Estructura. Ejemplo

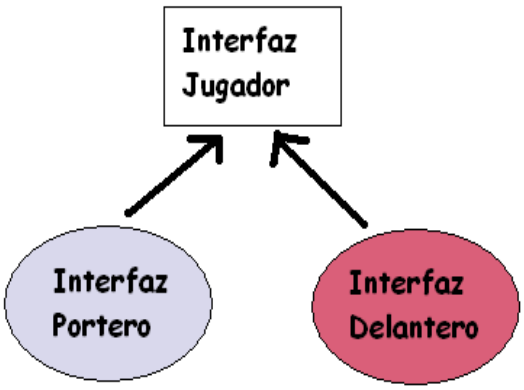
- Identificamos a continuación los elementos del patrón Builder:
  - Constructor: TextConverter.
  - Constructor concreto: ASCIIConverter, TeXConverter, TextWidgetConverter.
  - Producto: ASCIIText, TeXText, TextWidget.
  - Director: RTFReader.

Ejemplo

- **Datos**



- **Representación**



Consecuencias

1. **Permite variar la representación interna de un producto.** El Constructor proporciona al director una interfaz para la construcción del producto, permitiendo la ocultación de su representación, su estructura interna y su ensamblaje. Para cambiar la representación interna del producto se define un nuevo constructor.
2. **Aísla el código de construcción y representación**. Se aumenta la modularidad al encapsular cómo se construyen y se representan los objetos complejos. Los clientes no saben nada de las clases que definen la estructura interna del producto.
3. **Proporciona un control más fino sobre el proceso de construcción.** Se construye el producto paso a paso (los demás patrones lo hacen de una vez) bajo el control del director.

Consecuencias

- Reduce el acoplamiento.
- Permite variar la representación interna de estructuras complejas, respetando la interfaz común de la clase Builder.
- Se independiza el código de construcción de la representación. Las clases concretas que tratan las representaciones internas no forman parte de la interfaz del Builder.
- Cada ConcreteBuilder tiene el código específico para crear y modificar una estructura interna concreta.
- Distintos Directores con distintas utilidades pueden utilizar el mismo ConcreteBuilder.
- Permite un control mayor en el proceso de creación del objeto. El Director controla la creación paso a paso, solo cuando el Builder ha terminado de construir el objeto lo recupera el Director.

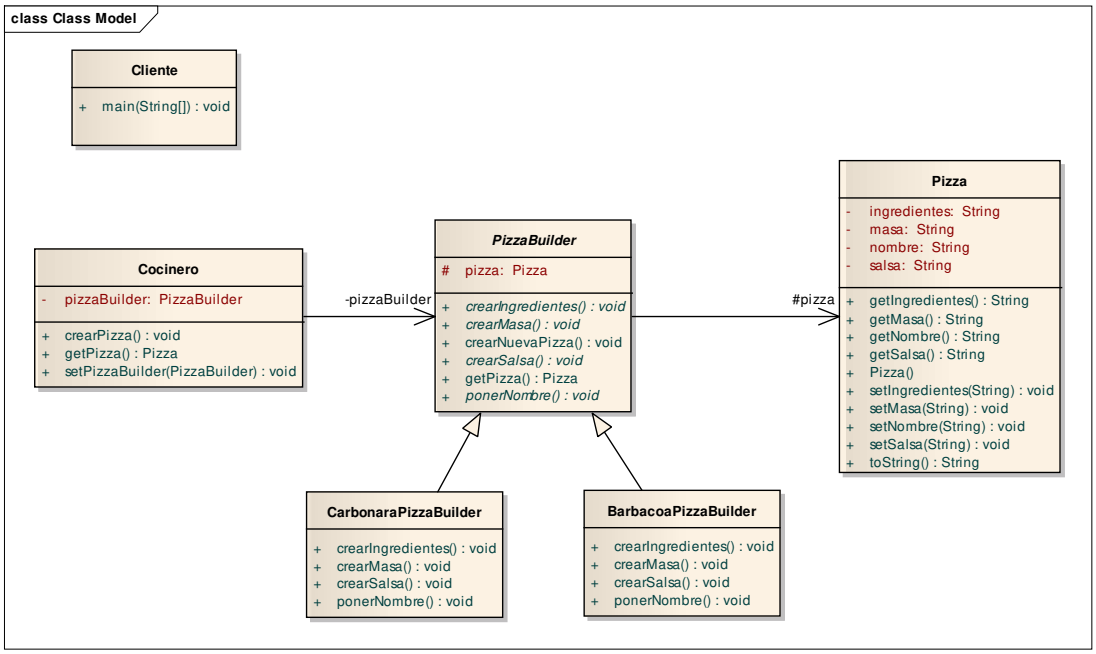
Patrones relacionados

- **Factory Method**. El patrón Builder usa el patrón Factory Method para decidir qué clase concreta instanciar para construir el tipo de objeto deseado.
- **Abstract Factory.** El patrón Abstract Factory se preocupa de lo que se va a hacer, y el Builder, de cómo se va a hacer. Abstract Factory es similar al patrón Builder en que ambos pueden construir objetos complejos. La principal diferencia es que el patrón Builder se centra en construir un objeto complejo paso a paso. Abstract Factory hace énfasis en familias de objetos producto, tanto simples como complejas.
- **Prototype**. El builder se centra en crear paso a paso objetos complejos. Una clase builder es creada por cada clase producto que encapsula la complejidad de crear el objeto. La ventaja del Prototipo es, junto con la necesidad de solo una clase "factory", el hecho de que la lógica de creación está unida directamente con cada clase producto.
- **Composite:** El patrón Builder a menudo se utiliza para producir objetos Composite, porque suelen tener una estructura compleja.

Código de ejemplo

Creación de Pizzas

Código de ejemplo



Estructura. Ejemplo

- Identificamos a continuación los elementos del patrón Builder:
  - Constructor: PizzaBuilder.
  - Constructor concreto: CarbonaraPizzaBuilder, BarbacoaPizzaBuilder.
  - Producto: Pizza.
  - Director: Cocinero.
  - Cliente: Cliente.