Patrón de Diseño Builder

Nombre

- BUILDER (Constructor virtual).

- Es un patrón Creacional.
  - Corresponden a patrones de diseño de software que solucionan problemas de creación de instancias. Nos ayudan a encapsular y abstraer dicha creación.

Intención

- Construir de manera flexible un objeto complejo a partir de otro objeto complejo en una serie de pasos.
- El objetivo es conseguir que la construcción de un objeto compuesto sea independiente de su representación, de manera que la construcción no se vea afectada por el hecho de que cambie su forma de representación.

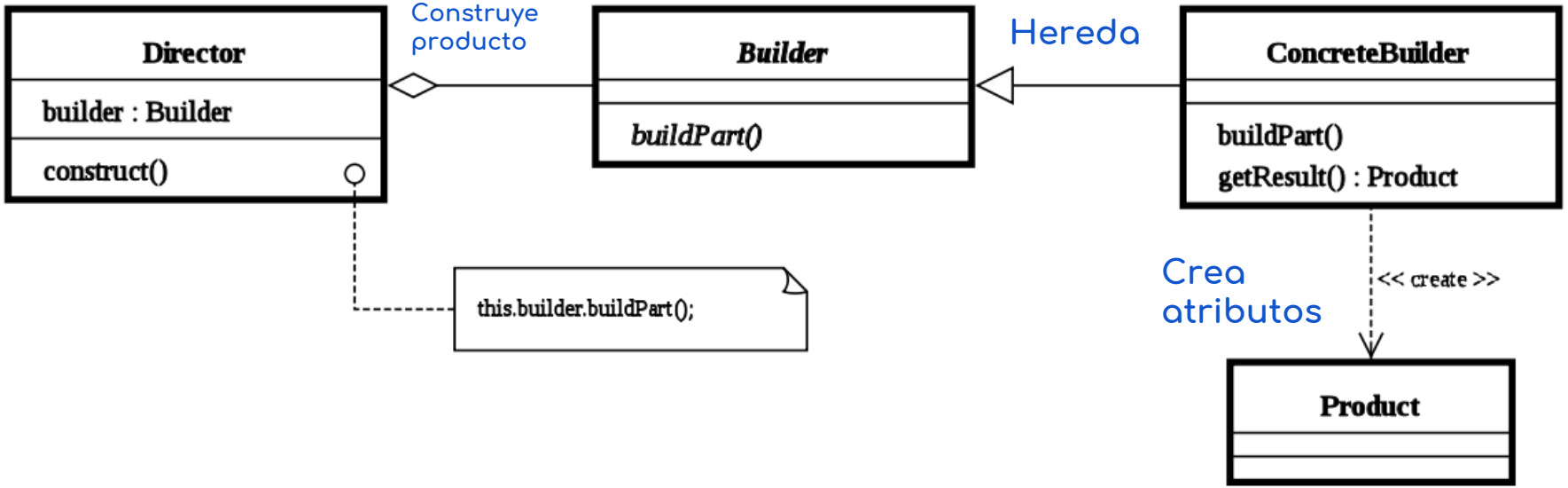
Motivación

- 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 algoritmos para crear diferentes representaciones.

Aplicación

- El patrón Builder se usa cuando:
  - El algoritmo para creación de un objeto complejo debe ser independiente de las partes que conforman el objeto y cómo están ensambladas.
  - El proceso de construcción debe permitir diferentes representaciones del objeto que se construye.

Estructura


Director
builder : Builder
construct()
Construye producto
Builder
buildPart()
Hereda
ConcreteBuilder
buildPart()
getResult() : Product
this.builder.buildPart();
Crea atributos
<< create >>
Product

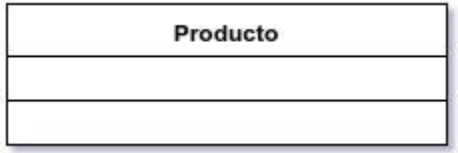
Participantes

- El patrón de diseño Builder se caracteriza por presentar siempre 4 componentes:
  - Clase ‘Producto’.
  - Clase Abstracta (Builder).
  - Clase(s) Concreta(s) (Concrete Builders).
  - Clase ‘Director’.

Clases

‘Producto’:

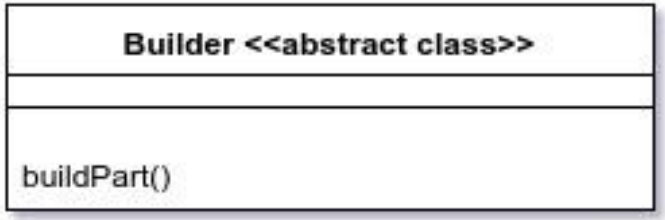
- *Representa el objeto complejo en construcción.*
- *Es donde se crean los atributos del ‘Producto’. Y es donde se crean los métodos ‘set’ de los atributos anteriormente creados.*
- *Incluye las clases que definen las partes componentes, incluyendo interfaces para ensamblar las partes dentro del resultado final.*



Clases

‘Builder’:

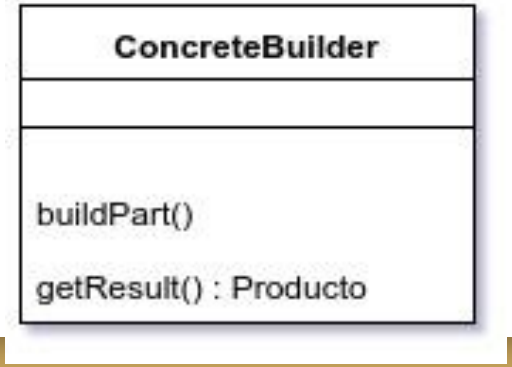
- *Especifica una interfaz abstracta para la creación de partes de un objeto Producto.*
- *Se caracteriza por tener un atributo a un objeto de la clase ‘Producto’. Así mismo, debe tener métodos concretos y métodos abstractos.*
- *Los métodos abstractos asignan valores predeterminados pero no los define, solamente declara el prototipo de estos métodos.*



Clases

‘Concrete Builder’:

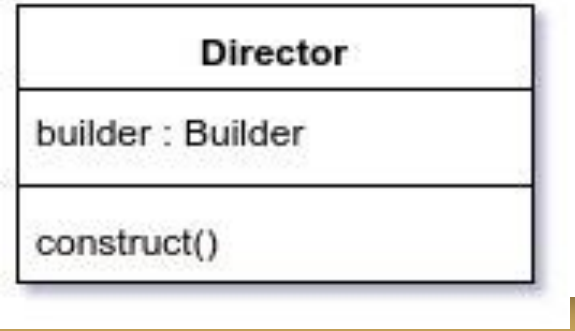
- *Pueden ser varias clases, no tiene límite. Los valores que serán asignados son definidos en esta clase.*
- *Construye y ensambla las partes del producto por implementación de la interfaz Builder.*
- *Define y guarda la ruta de la representación que crea.*
- *Provee una interfaz para recuperación del producto.*



Clases

‘Director’:

- *Es la clase que gestiona el uso de las clases anteriores. Así, desde una clase ‘Principal’, simplemente habrá que hacer uso de la clase ‘Director’ para instanciar objetos de la clase ‘Producto’, dependiendo del ‘ConcreteBuilder’ que se le asigne.*
- *Construye un objeto usando la interfaz Builder.*



Colaboraciones

- 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.

Consecuencias

Ventajas:

- Reduce el acoplamiento.
- Se independiza el código de construcción de la representación.
- Cada 'ConcreteBuilder' tiene el código específico para crear y modificar una estructura interna concreta.
- Distintas clases 'Director' con distintas utilidades, pueden utilizar el mismo "ConcreteBuilder'.
- Permite un mayor control en el proceso de creación del objeto.

Consecuencias

- Permite variar la representación interna de un producto.
- Separa el código de construcción del de representación.

Desventajas:

- Requiere crear un ConcreteBuilder por separado para cada tipo diferente de Producto.

Consecuencias

- Requiere crear un ConcreteBuilder por separado para cada tipo diferente de Producto.
- Requiere que las clases de constructor sean mutables.
- No se garantiza la inicialización de los miembros de datos de la clase.
- La inyección de dependencia puede ser menos compatible.

Ejemplo de código

```
from __future__ import print_function
from abc import ABCMeta, abstractmethod


class Car(object):
    def __init__(self, wheels=4, seats=4, color="Black"):
        self.wheels = wheels
        self.seats = seats
        self.color = color

    def __str__(self):
        return "This is a {0} car with {1} wheels and {2} seats.".format(
            self.color, self.wheels, self.seats
        )
```

Ejemplo de código

```
class Builder:
    __metaclass__ = ABCMeta

    @abstractmethod
    def set_wheels(self, value):
        pass

    @abstractmethod
    def set_seats(self, value):
        pass

    @abstractmethod
    def set_color(self, value):
        pass

    @abstractmethod
    def get_result(self):
        pass
```

Ejemplo de código

```
class CarBuilder(Builder):
    def __init__(self):
        self.car = Car()

    def set_wheels(self, value):
        self.car.wheels = value

    def set_seats(self, value):
        self.car.seats = value

    def set_color(self, value):
        self.car.color = value

    def get_result(self):
        return self.car
```

Ejemplo de código

```
class CarBuilderDirector(object):
    @staticmethod
    def construct():
        builder = CarBuilder()
        builder.set_wheels(8)
        builder.set_seats(4)
        builder.set_color("Red")
        return builder.get_result()


car = CarBuilderDirector.construct()
print(car)
```

Usos conocidos

En la generación de distintos tipos de Sitemaps (Google, HTML...). El ejemplo puede verse en el siguiente enlace:

https://github.com/dparoulek/java-koans/tree/master/src/main/java/com/upgradingdave/koans/builder

*SiteMaps: es un archivo en el que se pueden enumerar las páginas de tu sitio web para informar a Google y a otros motores de búsqueda sobre la organización del contenido del mismo.

Patrones relacionados

- A menudo el patrón Builder construye el patrón Composite.

Fuentes

- https://es.wikipedia.org/
- https://programacion-innata.blogspot.mx/2014/02/patron-de-diseno-builder-incluye.html
- https://informaticapc.com/patrones-de-diseno/builder.php
- https://gist.github.com/pazdera/1121157
- https://en.wikipedia.org/wiki/Builder_pattern#Disadvantages