Politécnico Colombiano Jaime Isaza Cadavid

Facultad de Ingeniería

Builder

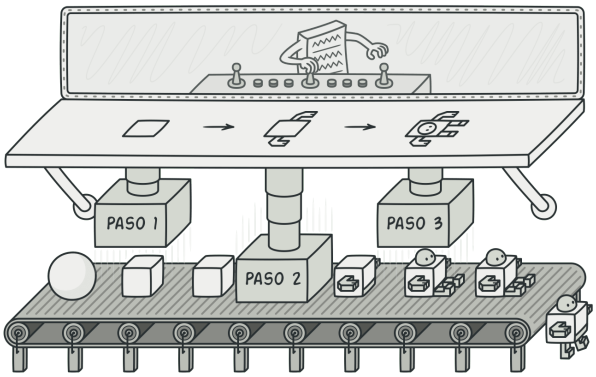
Diseño de software

Autor: Claudia
Apellidos: Gil Sánchez

15 de enero de 2022

Introducción

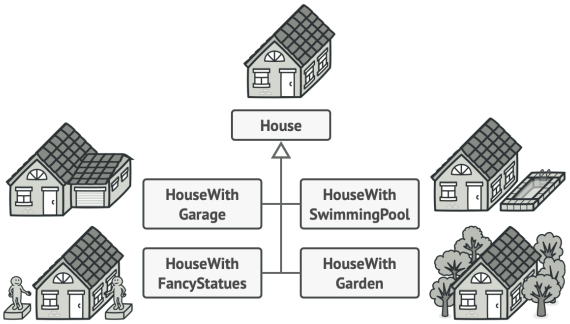
Como patrón de diseño, el patrón builder se usa para permitir la creación de una variedad de objetos complejos desde un objeto fuente. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.



Problema

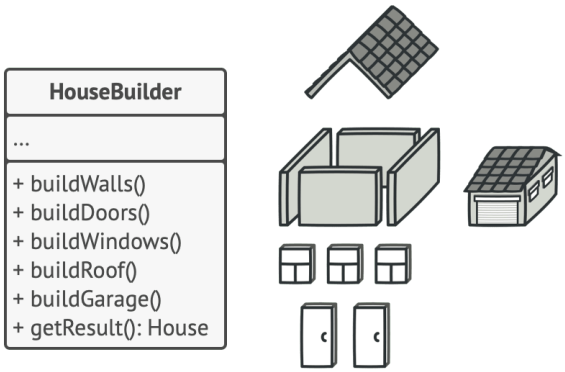
Imagina un objeto complejo que requiere una inicialización laboriosa, paso a paso, de muchos campos y objetos anidados. Normalmente, este código de inicialización está sepultado dentro de un monstruoso constructor con una gran cantidad de parámetros. O, peor aún: disperso por todo el código cliente.

Crear una subclase por cada configuración posible de un objeto puede complicar demasiado el programa. Un constructor con un montón de parámetros tiene su inconveniente: no todos los parámetros son necesarios todo el tiempo.



Solución

El patrón Builder te permite construir objetos complejos paso a paso. El patrón Builder no permite a otros objetos acceder al producto mientras se construye. Los distintos constructores ejecutan la misma tarea de formas distintas. La clase directora sabe qué pasos de construcción ejecutar para lograr un producto que funcione.

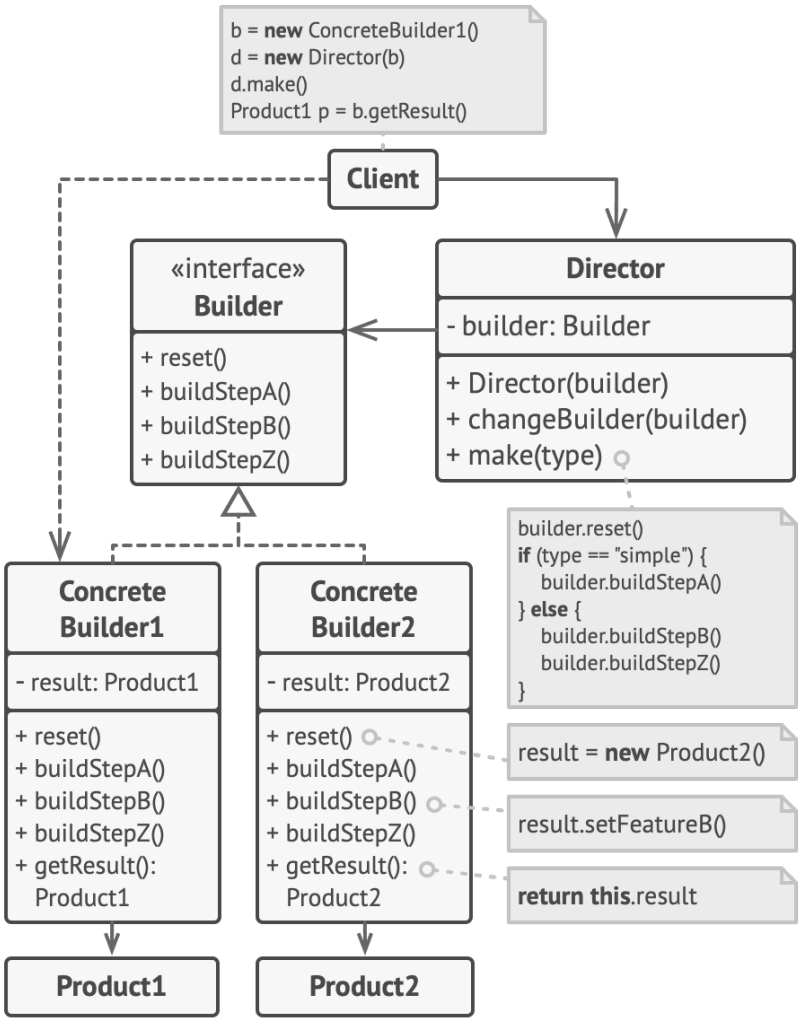


Clase directora

El cliente sólo necesita asociar un objeto constructor con una clase directora, utilizarla para iniciar la construcción, y obtener el resultado del objeto constructor. Puedes ir más lejos y extraer una serie de llamadas a los pasos del constructor que utilizas para construir un producto y ponerlas en una clase independiente llamada directora. La clase directora define el orden en el que se deben ejecutar los pasos de construcción, mientras que el constructor proporciona la implementación de dichos pasos.

1. Estructura

   - La interfaz Constructora declara pasos de construcción de producto que todos los tipos de objetos constructores tienen en común.
   - Los Constructores Concretos ofrecen distintas implementaciones de los pasos de construcción. Los constructores concretos pueden crear productos que no siguen la interfaz común.
   - Los Productos son los objetos resultantes. Los productos construidos por distintos objetos constructores no tienen que pertenecer a la misma jerarquía de clases o interfaz.
   - La clase Directora define el orden en el que se invocarán los pasos de construcción, por lo que puedes crear y reutilizar configuraciones específicas de los productos.
   - El Cliente debe asociar uno de los objetos constructores con la clase directora. Normalmente, se hace una sola vez mediante los parámetros del constructor de la clase directora, que utiliza el objeto constructor para el resto de la construcción. No obstante, existe una solución alternativa para cuando el cliente pasa el objeto constructor al método de producción de la clase directora. En este caso, puedes utilizar un constructor diferente cada vez que produzcas algo con la clase directora.



Cómo implementarlo

Existe una manera alternativa, en la que el objeto constructor se pasa directamente al método de construcción del director.

El resultado de la construcción tan solo se puede obtener directamente del director si todos los productos siguen la misma interfaz. Antes de que empiece la construcción, el cliente debe pasar un objeto constructor al director.

El código cliente crea tanto el objeto constructor como el director.

Crea una clase constructora concreta para cada una de las representaciones de producto e implementa sus pasos de construcción. El director utiliza el objeto constructor para el resto de la construcción. La razón por la que este método no se puede declarar dentro de la interfaz constructora es que varios constructores pueden construir productos sin una interfaz común.

No olvides implementar un método para extraer el resultado de la construcción. No obstante, si trabajas con productos de una única jerarquía, el método de extracción puede añadirse sin problemas a la interfaz base.

Relaciones con otros patrones

Puedes utilizar Builder al crear árboles Composite complejos porque puedes programar sus pasos de construcción para que funcionen de forma recursiva. Muchos diseños empiezan utilizando el Factory Method (menos complicado y más personalizable mediante las subclases) y evolucionan hacia Abstract Factory, Prototype, o Builder (más flexibles, pero más complicados).

Builder se enfoca en construir objetos complejos, paso a paso. Abstract Factory devuelve el producto inmediatamente, mientras que Builder te permite ejecutar algunos pasos adicionales de construcción antes de extraer el producto.

1. El uso del patrón Builder está muy extendido en las principales bibliotecas Java:
   - java.lang.StringBuilderappend() (unsynchronized)
   - java.lang.StringBufferappend() (synchronized)
   - java.nio.ByteBufferput() (también en CharBuffer, ShortBuffer, IntBuffer, LongBuffer, FloatBuffer y DoubleBuffer)
   - javax.swing.GroupLayout.GroupaddComponent()
   - Todas las implementaciones java.lang.Appendable
2. Identificación: El patrón Builder se puede reconocer por una clase, que tiene un único método de creación y varios métodos para configurar el objeto resultante. A menudo, los métodos del Builder soportan el encadenamiento (por ejemplo, algúnBuilder-¿establecerValorA(1)-¿establecerValorB(2)-¿crear()).

Ejemplo del patrón de diseño Builder

```java
package io.github.picodotdev.pattern.builder;

public class Usuario {

    private String email;
    private String nombre;
    private String apellidos;
    private String telefono;
    private String direccion;

    private Usuario() {
    }

    Usuario(UsuarioBuilder builder) {
        if (builder.getEmail() == null) {
            throw new IllegalArgumentException("email es requerido");
        }
        this.email = builder.getEmail();
        this.nombre = builder.getNombre();
        this.apellidos = builder.getApellidos();
        this.telefono = builder.getTelefono();
        this.direccion = builder.getDireccion();
    }
}
```

```java
package io.github.picodotdev.pattern.builder;

public class UsuarioBuilder {

    private String email;
    private String nombre;
    private String apellidos;
    private String telefono;
    private String direccion;

    public UsuarioBuilder() {
    }

    public UsuarioBuilder email(String email) {
        this.email = email;
        return this;
    }

    public UsuarioBuilder nombre(String nombre, String apellidos) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        return this;
    }

    public UsuarioBuilder telefono(String telefono) {
        this.telefono = telefono;
        return this;
    }

    public UsuarioBuilder direccion(String direccion) {
        this.direccion = direccion;
        return this;
    }
```

```
34
35      public Usuario build() {
36          return new Usuario(this);
37      }
38
39      // Getters
40      public String getEmail() {
41          return email;
42      };
43
44      public String getNombre() {
45          return nombre;
46      };
47
48      public String getApellidos() {
49          return apellidos;
50      };
51
52      public String getTelefono() {
53          return telefono;
54      };
55
56      public String getDireccion() {
57          return direccion;
58      };
59  }
```

Su uso sería de la siguiente manera algo más autoexplicativa y legible que la opción de usar constructores.

```
1  package io.github.picodotdev.pattern.builder;
2
3  public class Main {
4
5    public static void main(String[] args) {
6      Usuario usuario = new UsuarioBuilder()
7          .email("nombre.apellido@gmail.com")
8          .nombre("Nombre", "Apellido")
9          .telefono("555123456")
10         .direccion("c\\ Rue el Percebe 13").build();
11   }
12 }
```

Referencias

Builder. (n.d.). Refactoring.guru. Retrieved January 15, 2022, from https://refactoring.guru/es/design-patterns/builder.

picodotdev. (2015, September 27). Ejemplo del patrón de diseño Builder. Blog Bitix. https://picodotdev.github.io/blog-bitix/2015/09/ejemplo-del-patron-de-diseno-builder/.