POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID

Patrón de diseño

BUILDER

Realizado por
Sebastian Zapata Castaño

Dirigido por
Hector Manuel Vanegas Solis

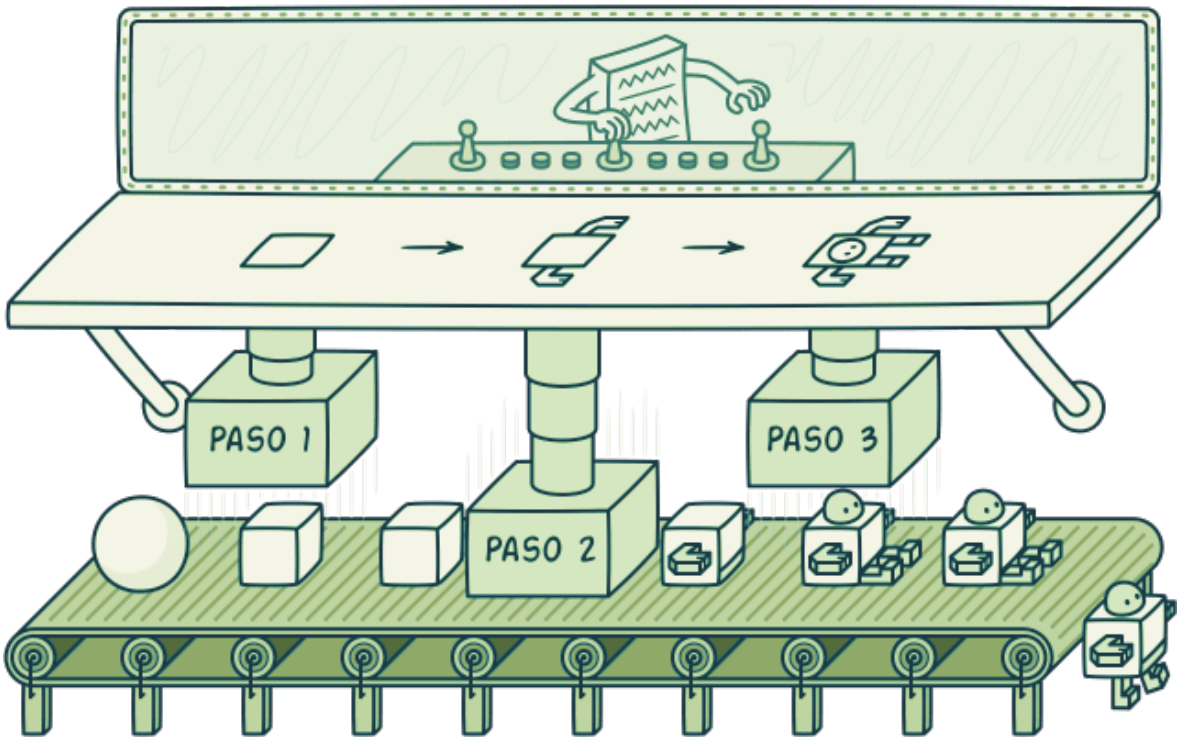
Universidad
Politecnico Colombiano Jaime Isaza Cadavid

Enero, 2022

Descripcion general

El patrón Builder es un tipo de patrón de diseño que se utiliza para resolver tareas de programación en la programación orientada a ojetos. Los patrones Builder (o Constructor) facilitan el proceso de programación para los desarrolladores porque no tienen que volver a diseñar cada paso una y otra vez como un proceso de programa.

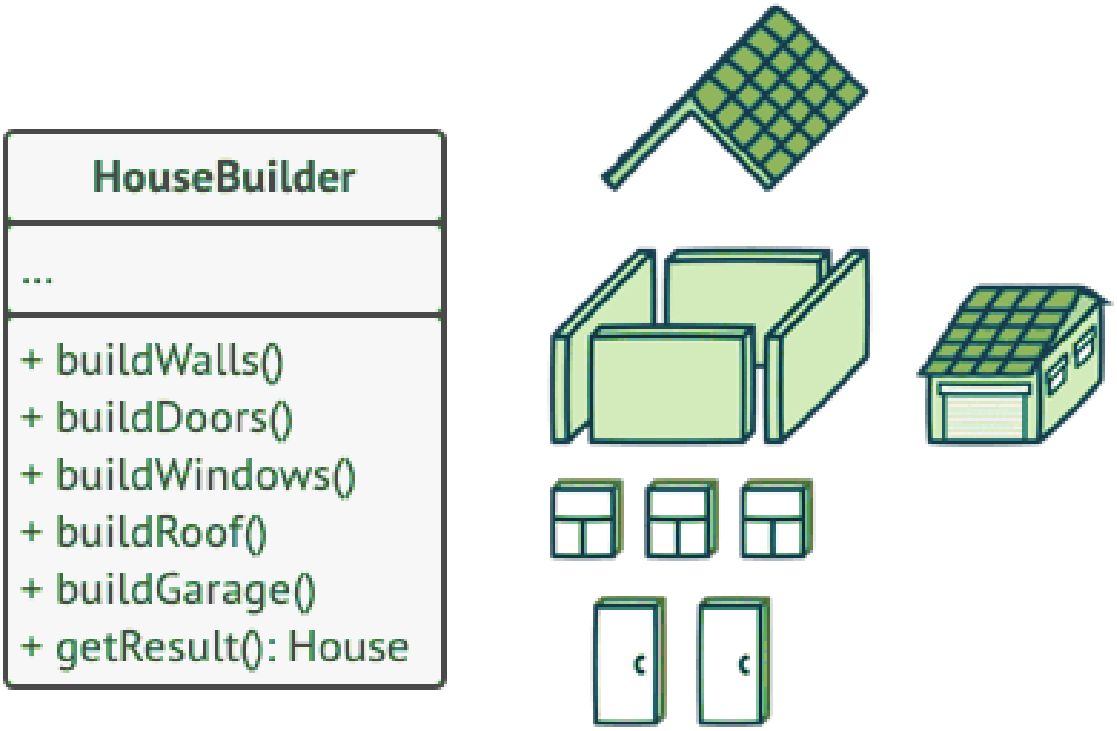
En lugar de un replanteamiento paso a paso, podrían usar una solución establecida. Los elementos del software se basan en el libro Design Pattern: Elements of Reusable ObjectOriented Software publicado en 199 por cuatro desarrolladores de software estadounidenses conocidos como Gang of Four, o GoF para abreviar. En este tutorial, lo guiaremos a través de los aspectos clave del patrón de diseño Builder e incluimos un ejemplo de trabajo.



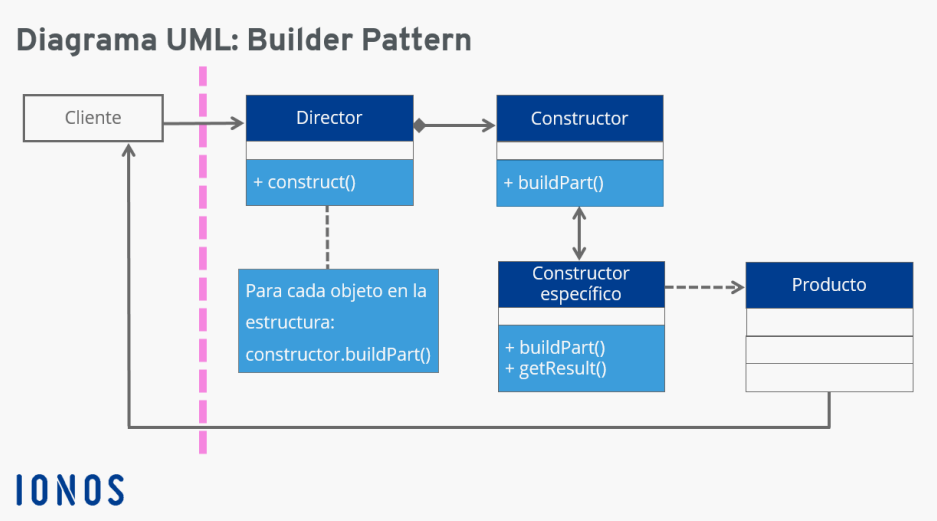
¿Que soluiona este patrón de diseño?

Un objeto complejo que requiere una inicialización laboriosa y paso a paso de muchos campos y objetos anidados. Por lo general, este código de inicialización está enterrado en un constructor gigante con una gran cantidad de parámetros.

Pensemos en construir una casa, lógico que debemos construir las paredes techo, suelo y demás... pero si queremos instalar cosas nuevas o que sea mas luminosa ¿que haríamos? para no llenarnos de sub clases el patrón Builder sugiere que saques el código de construcción del objeto de su propia clase y lo coloques dentro de objetos independientes llamados constructores.

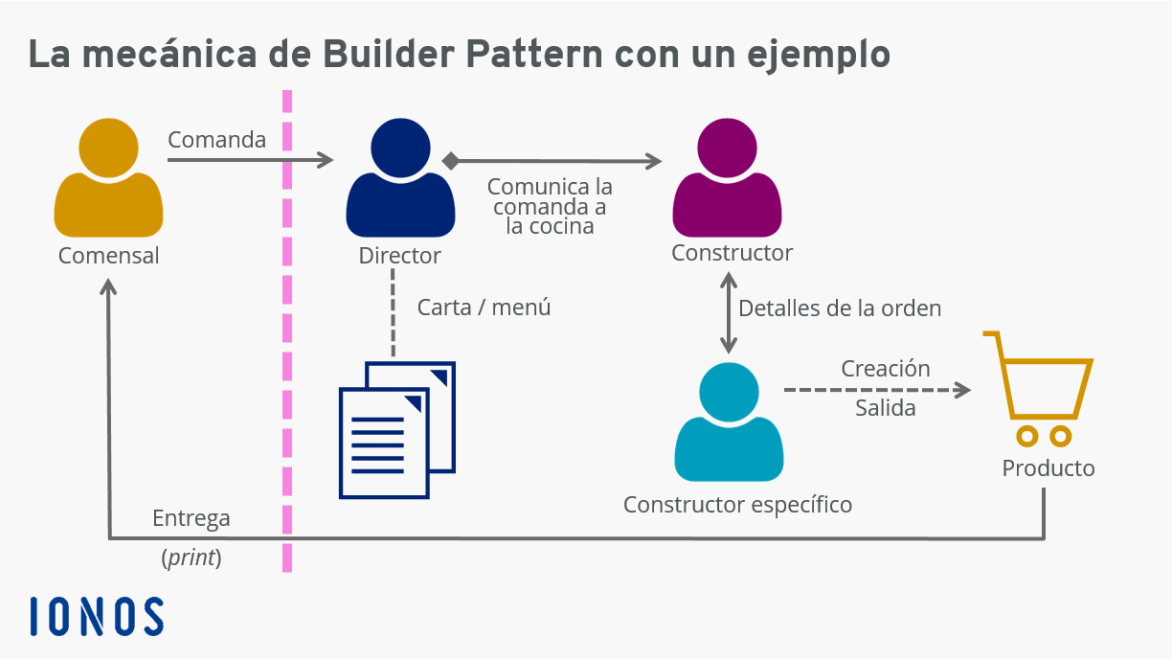


Builder como diagrama UML



¿Como se utiliza este patrón de diseño?

Una forma de ilustrar el patrón de diseño de Builder es tomar el ejemplo de un restaurante donde un cliente hace un pedido. Después de recibir el pedido el chef preparará. Todo el proceso hasta la entrega real se lleva a cao .[en]tre bambalinas"; el cliente no ve lo que sucede en la cocina y solo otiene los resultados (imprime en lenguaje de programación).



Ventajas y desventajas de Builder

0.0.1. Ventajas del patrón de diseño Builder

La compilación y el renderizado (salida) se cominan por separado. Las representaciones internas del constructor están ocultas de la implementación. Estas nuevas representaciones se pueden integrar fácilmente utilizando clases específicas del constructor. El proceso de construcción está claramente controlado por el gerente. Si se necesitan camios se pueden realizar sin consultar al cliente.

0.0.2. Desventajas del patrón de diseño Builder

El patrón Builder incluye un vínculo estrecho entre el producto el constructor específico y las capas del proceso de diseño por lo que puede ser difícil realizar camios en el proceso suyacente. La construcción de ojetos requiere el conocimiento de su uso y entorno específicos. El uso de patrones familiares como el patrón de diseño Builder puede llevar a los desarrolladores a pasar por alto soluciones más simples y elegantes. En última instancia muchos desarrolladores consideran que este es uno de los patrones de diseño menos importantes.

Ejemplo en codigo del patron de diseño Builder

El objeto, es decir, el menú, está vacío al principio. Una vez se hace la comanda, se añade contenido.

```
1  public class Menu {
2     private String starter = "No entrante";
3     private String maincourse = "No plato principal";
4     private String dessert = "No postre";
5     private String drink = "No bebida";
6     public void setentrante (String starter) {
7             this.starter = starter;
8     }
9     public void setplatoprincipal (String maincourse) {
10            this.maincourse = maincourse;
11    }
12    public void setpostre(String dessert) {
13            this.dessert = dessert;
14    }
15    public void setbebida(String drink) {
16            this.drink = drink;
17    }
18    public void print() {
19            System.out.println(
20                    "¡Tu comida esta lista! " + "\n" +
21                    " - Entrante: " + starter +
22                    " - Plato principal: " + maincourse +
23                    " - Postre: " + dessert +
24                    " - Bebida: " + drink);
25    }
26 }
```

El director proporciona el “ambiente” necesario para que un plato pueda ser elaborado (o construido) para el cliente. Este ambiente es accesible para todos los clientes. El cliente solo se comunica con el director para que la preparación como tal esté oculta:

```
public class MattsRestaurant {
   private MenuBuilder menuBuilder;
   public void setBuilder(MenuBuilder menuBuilder) {
           this.menuBuilder = menuBuilder;
   }
   public Menu buildMenu(){
           menuBuilder.buildStarter();
           menuBuilder.buildMainCourse();
           menuBuilder.buildDessert();
           menuBuilder.buildDrink();
           return menuBuilder.build();
   }
}
```

Luego, entra en escena el constructor. En nuestro ejemplo, sería el cocinero jefe:

```
public abstract class MenuBuilder {
   Menu menu = new Menu();
   abstract void buildStarter();
   abstract void buildMainCourse();
   abstract void buildDessert();
   abstract void buildDrink();
   Menu build()
{
           return menu;
   }
}
```

El constructor específico, en este caso el cocinero, construye (es decir, cocina) los componentes individuales del plato de la comanda. Para ello, ignora (override) los artículos del menú "abstract":

```
public class MenuOfTheDayBuilder extends MenuBuilder {
   @Override
   public void buildStarter() {
           burger.setStarter("Sopa de calabaza");
   }
   @Override
   public void buildMainCourse() {
           burger.setMainCourse("Filete a la plancha con patatas");
   }
   @Override
   public void buildDessert() {
           burger.setDessert("Helado de vainilla");
   }
   @Override
   public void buildDrink() {
           burger.setDrink("Vino tinto");
   }
}
```

Por último, se resumen los componentes individuales del plato y se realiza la

entrega al cliente, es decir, “se imprime”.

```
public class Main {
   public static void main(String[] args) {
           MattsRestaurant mattsRestaurant = new MattsRestaurant();
           menuRestaurant.setBuilder(new MenuOfTheDayBuilderBuilder());
           buildMenu(menuRestaurant);
           menuRestaurant.setBuilder(new SpecialMenuBuilder());
           buildMenu(menuRestaurant);
   }
   private static void buildMenu(MattsRestaurant mattsRestaurant) {
           MenuOfTheDay menu = mattsRestaurant.buildMenu();
           menu.print();
   }
}
```

Referencias

- Builder. (n.d.). Refactoring.guru. Retrieved January 15, 2022, from https://refactoring.guru/es/design-patterns/builder
- ¿Qué es el patrón de diseño Builder? (n.d.). IONOS Digitalguide. Retrieved January 15, 2022, from https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-de-diseno-builder/