Patrones de diseño

Los patrones de diseño software se utilizan para resolver problemas comunes que surgen durante el proceso de diseño y desarrollo de software. Estos patrones son la representación de las mejores prácticas y enfoques generales para abordar determinados problemas que surgen en el desarrollo de software.

Los patrones de diseño cuentan con un lenguaje común y una forma de comunicación entre los desarrolladores, que les da la posibilidad de describir y discutir soluciones de diseño de manera más útil y efectiva. Y es que todos ellos están basados en la experiencia de los desarrolladores y expertos en el sector y se documentan para volver a usarlos en situaciones similares.

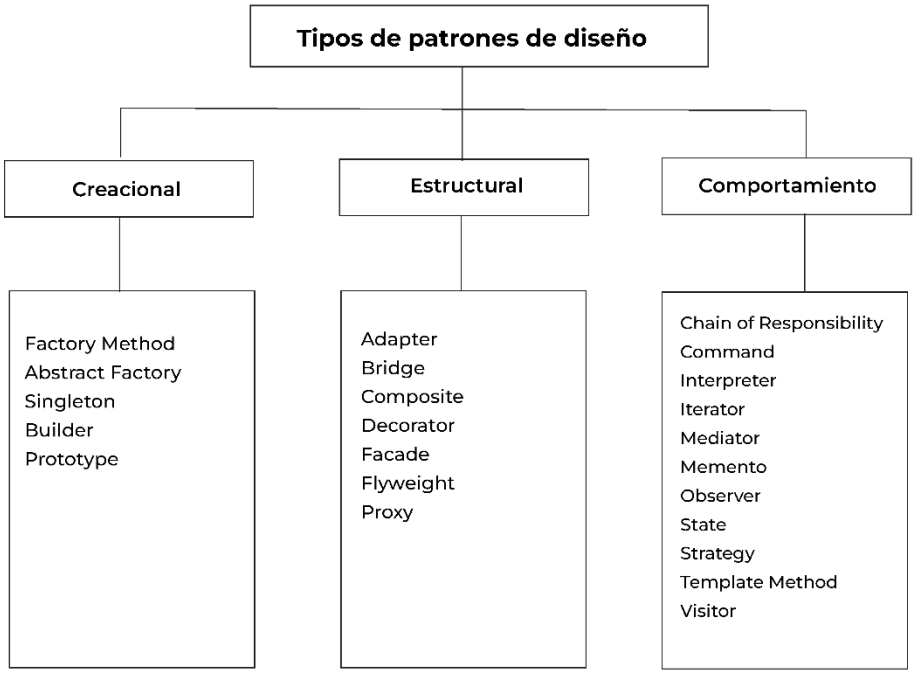
Existen diferentes **tipos de patrones de diseño software**, como los patrones de creación, los patrones de estructuración y los patrones de comportamiento.

Los **patrones de creación** se centran en cómo crear objetos y estructuras de objetos de manera eficiente

Los **patrones de estructuración** se ocupan de organizar y componer clases y objetos.

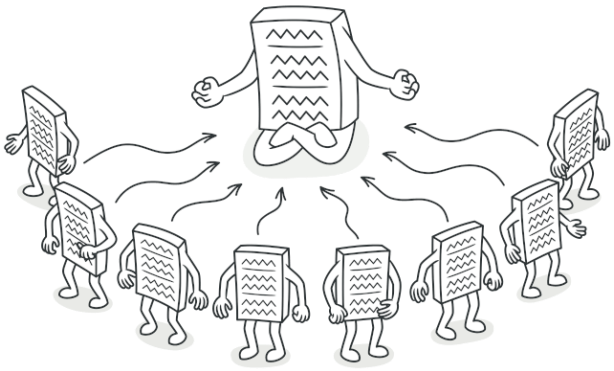
Los **patrones de comportamiento** se refieren a la interacción y comunicación entre objetos.

Utilizar patrones de diseño software puede mejorar la calidad del código, la flexibilidad del diseño y la reutilización de componentes

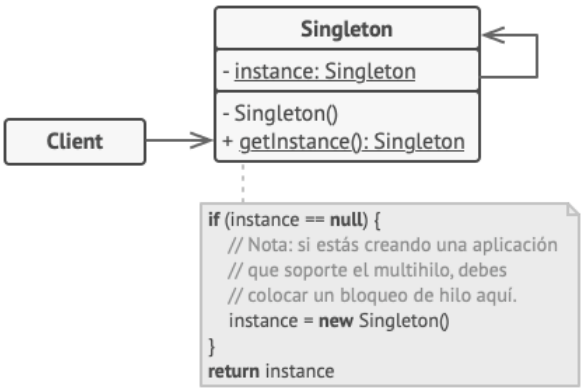


Singleton

Es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.



Estructura



La clase **Singleton** declara el método estático obtenerInstancia que devuelve la misma instancia de su propia clase.

El constructor del Singleton debe ocultarse del código cliente. La llamada al método obtenerInstancia debe ser la única manera de obtener el objeto de Singleton.

Ejemplo en Java

Singleton.java: Singleton

```
package refactoring_guru.singleton.example.non_thread_safe;

public final class Singleton {
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // The following code emulates slow initialization.
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        this.value = value;
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            instance = new Singleton(value);
        }
        return instance;
    }
}
```

DemoSingleThread.java: Código cliente

```
package refactoring_guru.singleton.example.non_thread_safe;

public class DemoSingleThread {
    public static void main(String[] args) {
        System.out.println("If you see the same value, then singleton was reuse
                "If you see different values, then 2 singletons were created (b
                "RESULT:" + "\n");
        Singleton singleton = Singleton.getInstance("FOO");
        Singleton anotherSingleton = Singleton.getInstance("BAR");
        System.out.println(singleton.value);
        System.out.println(anotherSingleton.value);
    }
}
```

OutputDemoSingleThread.txt: Resultados de la ejecución

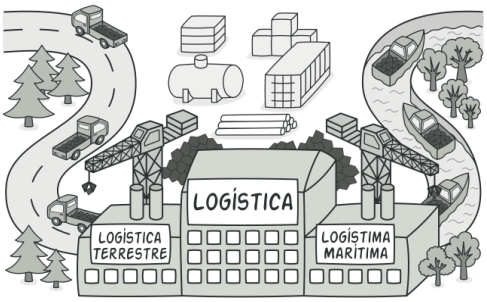
```
If you see the same value, then singleton was reused (yay!)
If you see different values, then 2 singletons were created (booo!!)

RESULT:

FOO
FOO
```

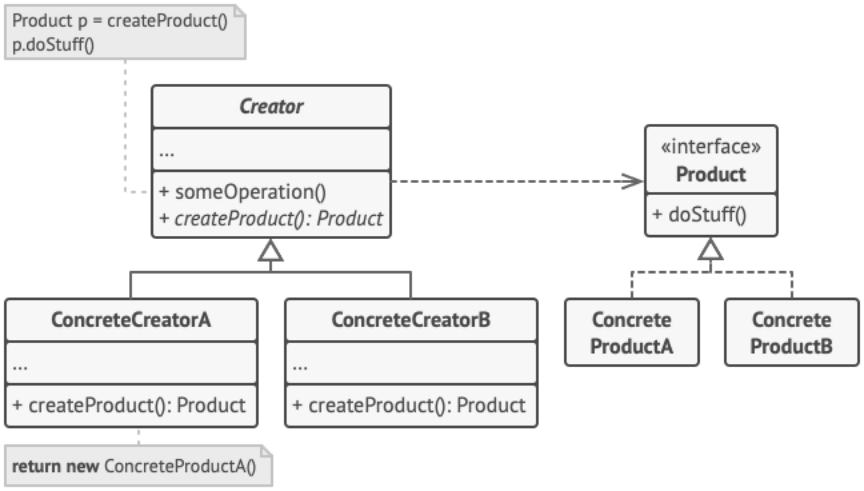
Factory Method

Es un patrón creacional que proporciona una interfaz para crear objetos en una superclase, pero permite a las subclases modificar el tipo de objetos que se crearán.



Es útil cuando se desea delegar la responsabilidad de crear instancias de una clase a una subclase sin especificar la clase concreta.

Estructura



El **Producto** declara la interfaz, que es común a todos los objetos que puede producir la clase creadora y sus subclases.

Los Productos Concretos son distintas implementaciones de la interfaz de producto.

La clase Creadora declara el método fábrica que devuelve nuevos objetos de producto. Es importante que el tipo de retorno de este método coincida con la interfaz de producto.

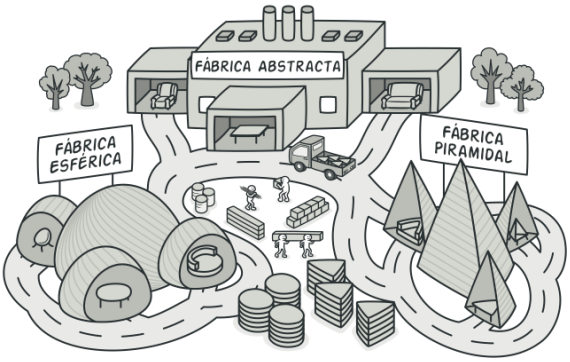
Los Productos Concretos son distintas implementaciones de la interfaz de producto.

Abstract Factory

Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.

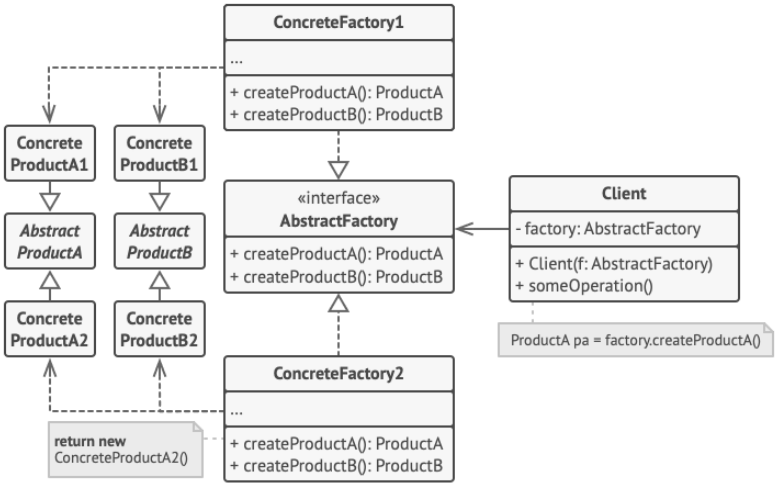
El objetivo principal del Abstract Factory es proporcionar una forma de crear objetos relacionados que pertenezcan a una misma familia o grupo. El patrón Abstract Factory está compuesto de dos niveles de abstracción:

La **fábrica abstracta** define los métodos de fábrica abstractos para cada objeto relacionado dentro de una familia.



La **fábrica concreta** implementa la fábrica abstracta y es responsable de crear y devolver instancias de los objetos concretos correspondientes.

Estructura



Los Productos Abstractos declaran interfaces para un grupo de productos diferentes pero relacionados que forman una familia de productos.

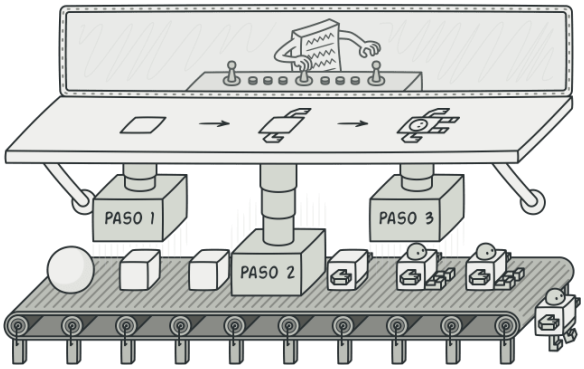
Los Productos Concretos son implementaciones distintas de productos abstractos agrupados por variantes.

Los Productos Concretos son implementaciones distintas de productos abstractos agrupados por variantes.

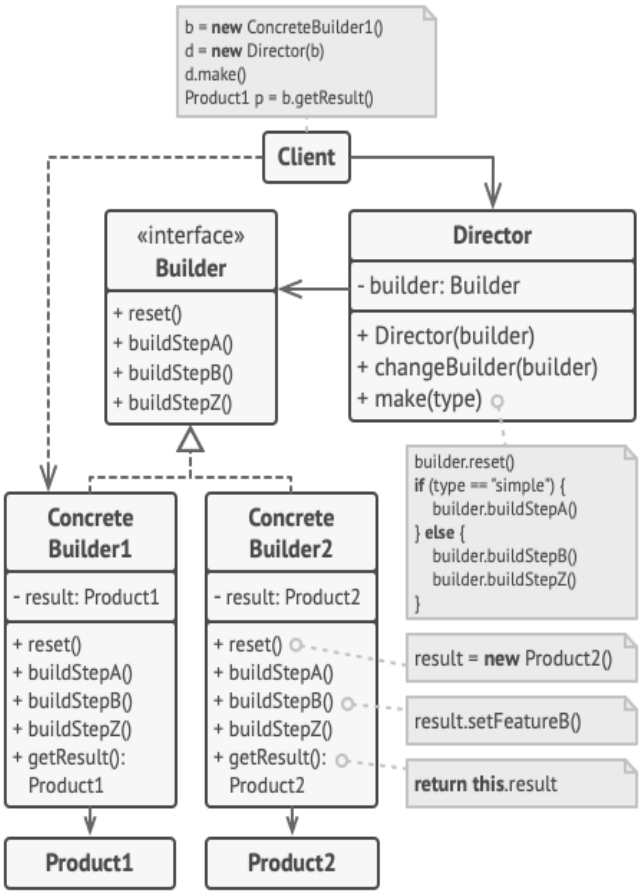
Las Fábricas Concretas implementan métodos de creación de la fábrica abstracta Cada fábrica concreta se corresponde con una variante específica de los productos y crea tan solo dichas variantes de los productos.

Builder

Separa el proceso de construcción de un objeto de su representación final, permitiendo la creación de diferentes representaciones del mismo objeto utilizando el mismo proceso de construcción. El objetivo principal del Builder es abstraer y simplificar el proceso de construcción de objetos complejos.



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.

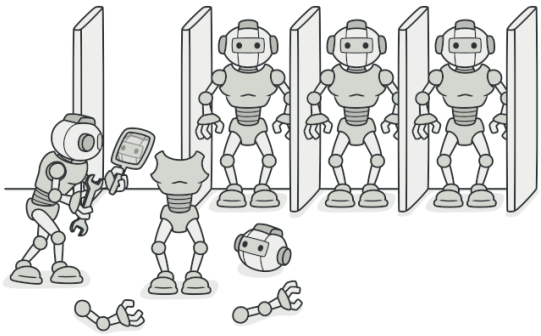
Los Productos son los objetos resultantes. Los productos construidos por distintos objetos constructores no tienen que pertenecer a la misma jerarquía de clases o interfaz.

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.

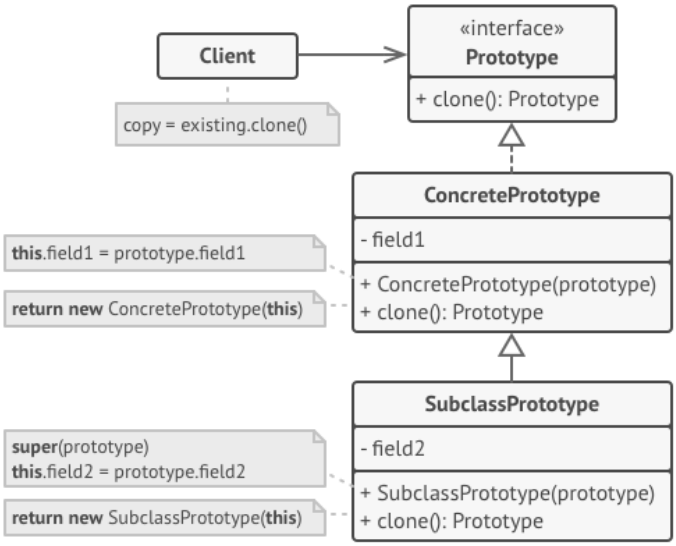
El Cliente debe asociar uno de los objetos constructores con la clase directora.

Prototype

Permite crear nuevos objetos duplicando un objeto existente, llamado prototipo, en lugar de crearlos desde cero. Es decir, permite clonar objetos existentes para crear nuevos objetos con propiedades y estados similares. El objetivo es evitar la creación repetitiva de objetos costosos en términos de rendimiento y recursos.



Estructura



La interfaz Prototipo declara los métodos de clonación. En la mayoría de los casos, se trata de un único método clonar.

La clase Prototipo Concreto implementa el método de clonación

El Cliente puede producir una copia de cualquier objeto que siga la interfaz del prototipo.

Referencias

- Bridge. (2024, 27 mayo). *¿Cuáles son los patrones de diseño software?* The Bridge | Digital Talent Accelerator. https://thebridge.tech/blog/patrones-de-diseno-software
- *Patrones de diseño / Design patterns*. (s. f.). https://refactoring.guru/es/design-patterns