POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

5. *Abstract Factory*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

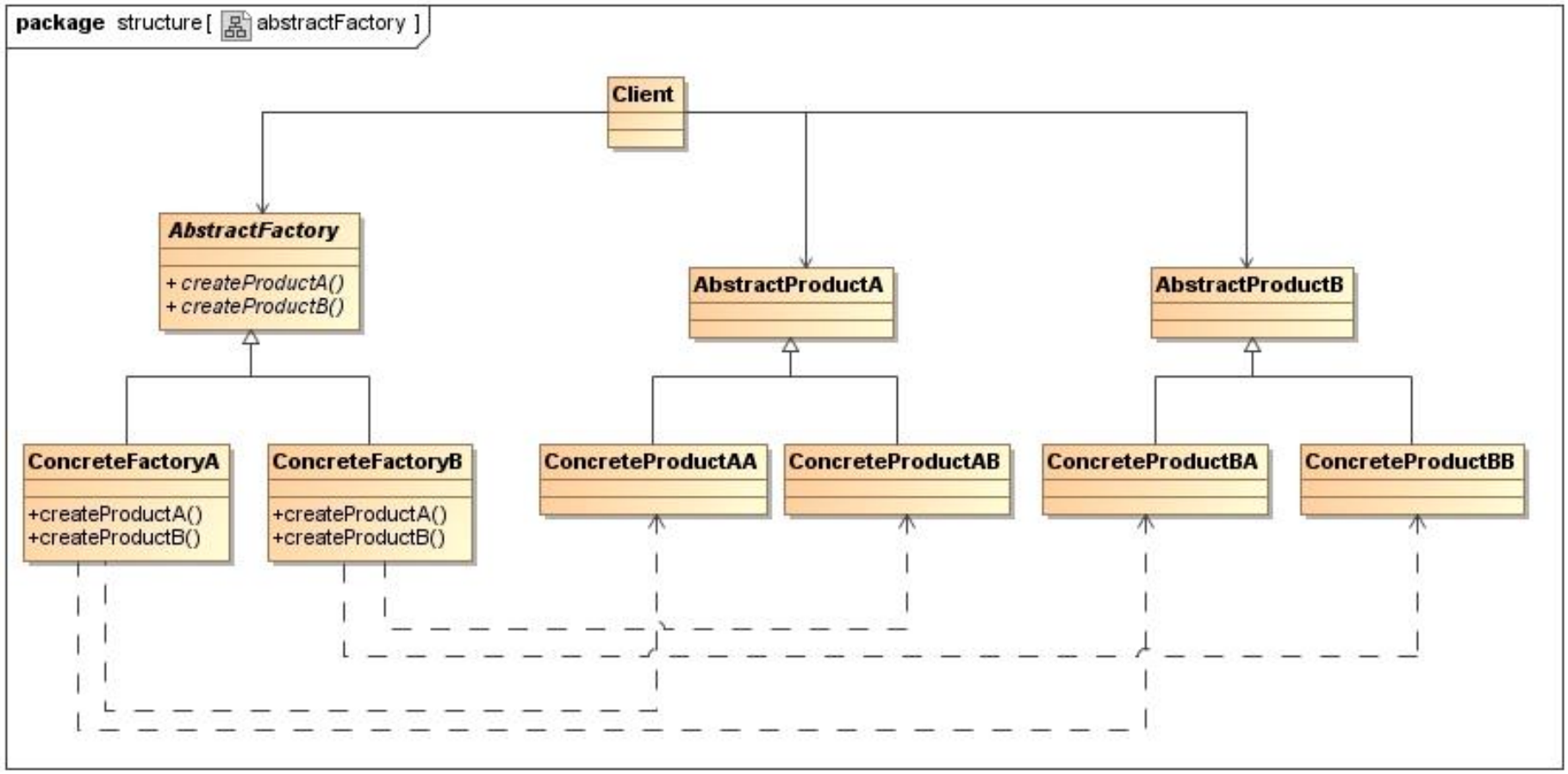
- *Un cirujano realiza una intervención siguiendo un algoritmo y dependiendo del lugar (quirófano, selva, …), la realizará con unos instrumentos u otros (bisturí vs navaja, pulsómetro vs contando pulsaciones, …*
- *Si la explicación del cirujano incluye la adquisición del instrumental en cualquier lugar, ésta complica y tiende a crecer más y más mezclando dos asuntos innecesariamente.*
- *Si la explicación de cirujano incluye la solicitud de instrumental abstracto (algo cortante, contador de pulsaciones, …) y otros explican por separado cómo se obtienen los instrumentos dependiendo lugar ninguna explicación será compleja ni tendiendo a crecer*

- ***Provee una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas***

1. Problema

- Úsese cuando:
  - Un sistema debe ser independiente de cómo se crean, componen y representan sus productos
  - Un sistema debe ser configurado con una familia de productos de entre varias
  - Una familia de objetos producto relacionados está diseñada para ser usada conjuntamente, y es necesario hacer cumplir esta restricción
  - Quiere proporcionar una biblioteca de clases de productos, y sólo quiere revelar sus interfaces, no sus implementaciones

2. Solución


package structure [abstractFactory]
Client
AbstractFactory
+ createProductA()
+ createProductB()
AbstractProductA
AbstractProductB
ConcreteFactoryA
+createProductA()
+createProductB()
ConcreteFactoryB
+createProductA()
+createProductB()
ConcreteProductAA
ConcreteProductAB
ConcreteProductBA
ConcreteProductBB

2. Solución

- Definiendo una clase *FabricaDeProductos* que declara una interfaz para crear cada tipo básico de producto. También hay una clase abstracta para cada tipo de producto, y las subclases concretas implementan útiles para un estándar concreto de interfaz de usuario.
  - La interfaz *FabricaDeProductos* tiene una operación que devuelve un nuevo objeto para cada clase abstracta de producto. Los clientes llaman a estas operaciones para obtener instancias de productos, pero no son conscientes de las clases concretas que están usando. De esta manera los clientes son independientes de la interfaz de usuario. En otras palabras, los clientes no tienen que atarse a una clase concreta, sino sólo a una interfaz definida por una clase abstracta.

2. Solución

- *AbstractFactory* solo declara una interfaz para crear productos. Se deja a las subclases *ConcreteFactory* el crearlos realmente. El modo más común de hacer esto es definiendo un método de fabricación para cada producto.
- *AbstractFactory* por lo general define una operación diferente para cada tipo de producto que puede producir. Los tipos de producto está codificados en las signaturas de las operaciones. Añadir un nuevo tipo de producto requiere cambiar la interfaz de *AbstractFactory* y todas las clases que dependen de ella.

2. Solución

- Un diseño más flexible, aunque menos seguro, es añadir un parámetro a las operaciones que crean los objetos. Este parámetro especifica el tipo de objeto a ser creado. Podría tratarse de un identificador de clase, un entero, una cadena de texto o cualquier otra cosa que identifique el tipo de producto. De hecho, con este enfoque, *AbstractFactory* solo necesita una única operación "Hacer" con un parámetro que indique el tipo de objeto a crear.

2. Solución

- Podemos aplicarla en C++ sólo cuando todos los objetos tienen la misma clase abstracta o cuando los objetos producto pueden ser convertidos con seguridad al tipo concreto por el objeto que lo solicita. Pero incluso cuando no es necesaria la conversión de tipos, todavía subyace un problema inherente: todos los productos se devuelven al cliente con la misma interfaz abstracta que el tipo de retorno. El cliente no podrá por tanto distinguir o hacer suposiciones seguras acerca de la clase de un producto. En caso de que los clientes necesiten realizar operaciones específicas de las subclases, éstas no estarán accesibles a través de la interfaz abstracta. Aunque el cliente podría hacer una conversión al tipo de una clase hija, eso no siempre resulta viable o seguro, porque la conversión de tipo puede fallar. Éste es el inconveniente típico de una interfaz altamente flexible y extensible.

3. Consecuencias

- El patrón *Abstract Factory* ayuda a controlar las clases de objetos que crea una aplicación. Como una fábrica encapsula la responsabilidad y el proceso de creación de objetos producto, aísla a los clientes de las clases de implementación. Los clientes manipulan las instancias a través de sus interfaces abstractas. Los nombre de las clases producto quedan aisladas en la implementación de la fábrica concreta; no aparecen en el código cliente.
- La clase de una fábrica concreta sólo aparece una vez en una aplicación – cuando se crea-. Esto facilita cambiar la fábrica concreta que usa una aplicación. Como una fábrica abstracta crea una familia completa de productos, toda la familia de productos cambia de una vez.

3. Consecuencias

- Cuando se diseñan objetos producto en una familia para trabajar juntos, es importante que una aplicación use objetos de una sola familia a la vez. *Abstract Factory* facilita que se cumpla esta restricción.
- Ampliar las fábricas abstractas para producir nuevos tipo de productos no es fácil. Esto se debe a que la interfaz *AbstractFactory* fija el conjunto de productos que se pueden crear. Permitir nuevos tipos de productos requiere ampliar la interfaz de la fábrica, lo que a su vez implica cambiar la clase *AbstractFactory* y todas sus subclases.

4. Relación

- ***Singleton*** para una aplicación que sólo necesita una instancia de una *FabricaConcreta* por cada familia de productos.
- ***Factory Method*** para que una fábrica concreta especifique sus productos. Si bien esta implementación es sencilla, requiere una nueva subclase fábrica concreta para cada familia de productos, incluso aunque las familias de productos difieran ligeramente.
- ***Prototype*** para el caso de que sea posible tener muchas familias de productos. La fábrica concreta se inicializa con una instancia prototípica de cada producto de la familia, y crea un nuevo producto clonando su prototipo. El enfoque basado en prototipos elimina la necesidad de una nueva clase de fábrica concreta para cada nueva familia de productos.

5. Comparación

- ***Abstract Factory** vs **Builder**.*
  - Se parecen en que 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 hincapié en familias de objetos "producto" (simples o complejos).
  - *Builder* devuelve el producto como paso final, mientras que *Abstract Factory* lo devuelve inmediatamente.
- ***Abstract Factory** vs **Facade** .*
  - *Abstract Factory* también puede ser una alternativa a *Facade* para ocultar clases específicas de una plataforma proporcionando una interfaz para crear el subsistema de objetos de forma independiente a otros subsistemas.