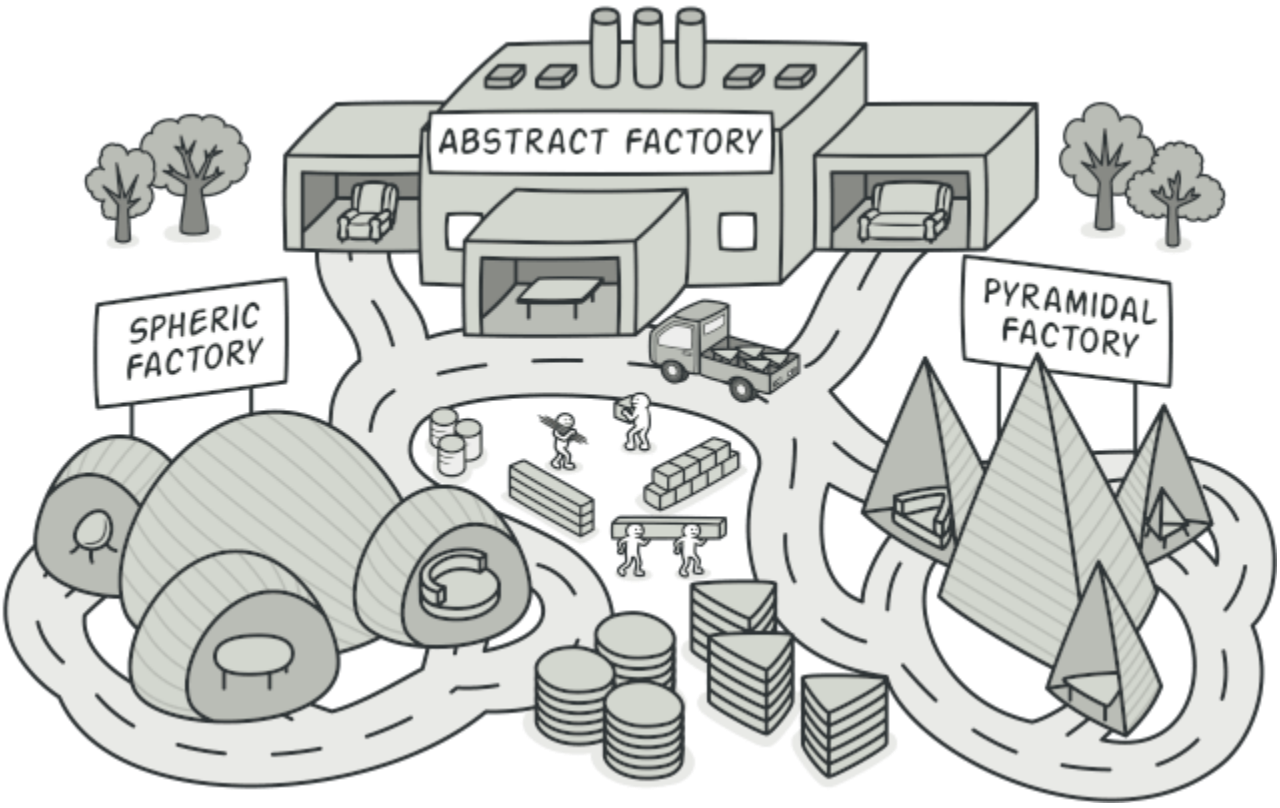
Patrón Abstract Factory

Abstract Factory es un patrón de diseño creacional que le permite producir familias de objetos relacionados sin especificar sus clases concretas.

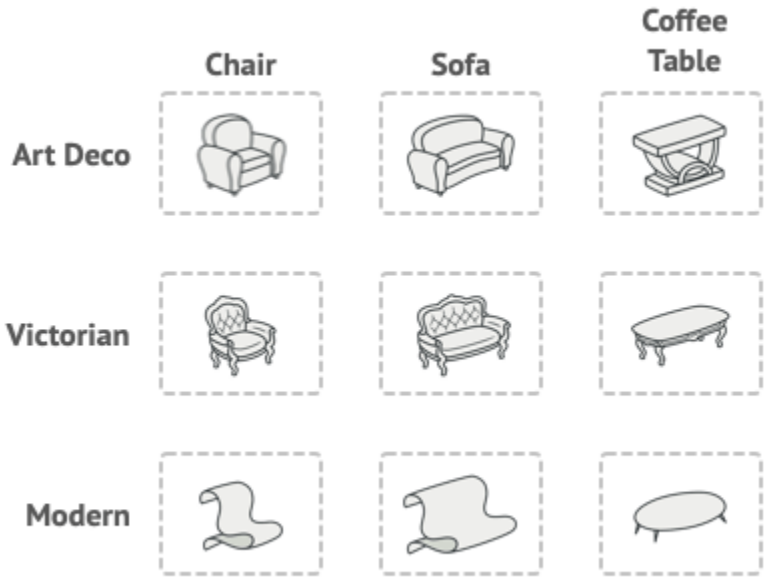


Problema

Imaginase que está creando un simulador de tienda de muebles. Su código consta de clases que representan:

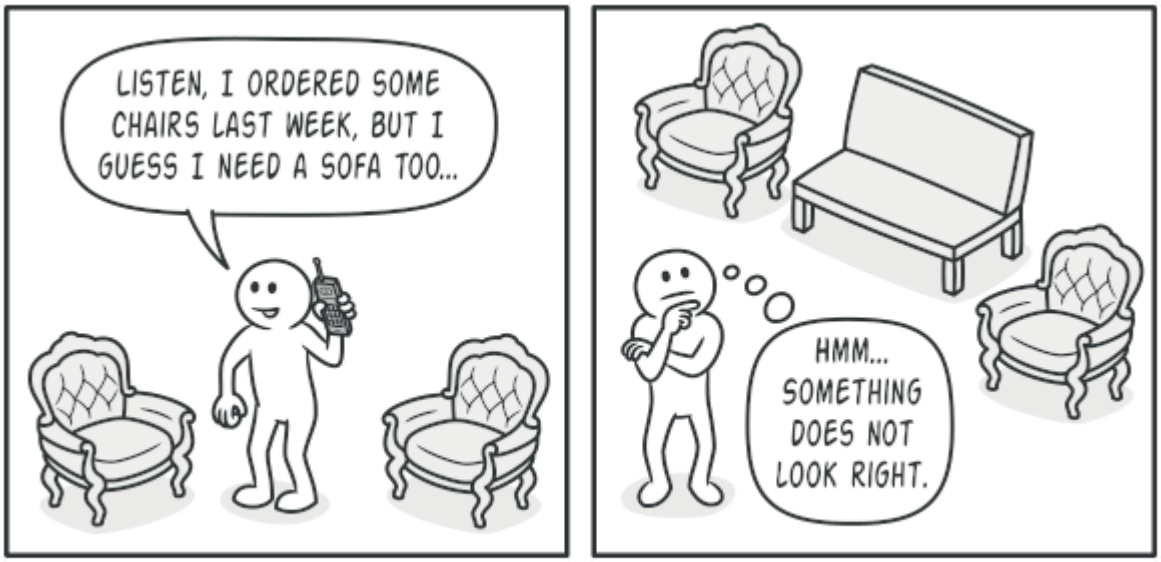
1. Una familia de productos relacionados , digamos: `Chair`, `Sofa`, `CoffeeTable`

2. Surgen variantes de esta familia. Por ejemplo, los productos están disponibles en estas variantes: `Modern, Victorian y ArtDeco`



Familias de productos y sus variantes.

Necesita una forma de crear muebles individuales para que coincidan con otros objetos de la misma familia. Los clientes se enojan bastante cuando reciben muebles que no combinan.

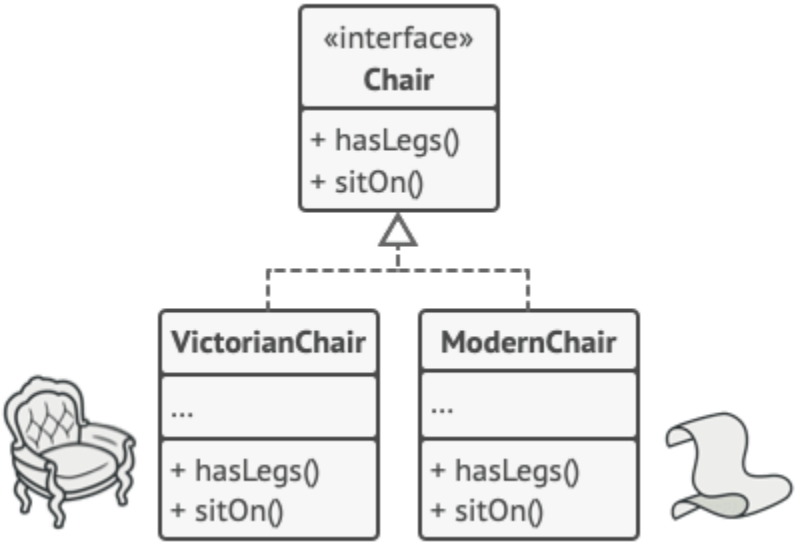


Un sofá de estilo moderno no combina con sillas de estilo victoriano.

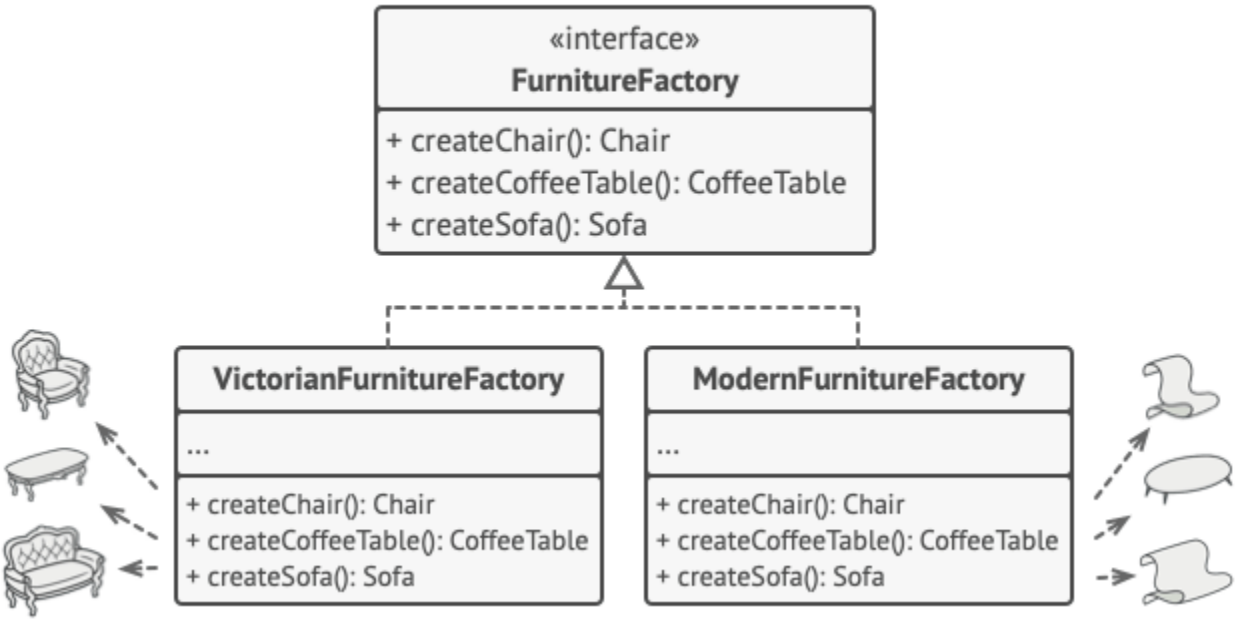
Hagas lo que hagas, no desea cambiar el código existente al agregar nuevos productos o familias de productos al programa. Los vendedores de muebles actualizan sus catálogos con mucha frecuencia y no querrá cambiar el código central cada vez que sucede.

Solución

Lo primero que sugiere el patrón **Abstract Factory** es declarar explícitamente las interfaces para cada producto distinto de la familia de productos (por ejemplo, una `Chair`, un `sofá` o `CoffeeTable`). Luego puede hacer que todas las variantes de productos sigan esas interfaces. Por ejemplo, todas las variantes de sillas pueden implementar la `Chair` interfaz; todas las variantes de mesas de café pueden implementar la `CoffeeTable` interfaz.



El siguiente paso es declarar una interfaz con los métodos de creación para todos los productos que forman parte de la familia de productos (por ejemplo, createChair, createSofa y createCoffeeTable. Estos métodos deben devolver tipos de productos **abstractos** representados por las interfaces que extrajimos anteriormente: `Chair`, `Sofa`, `CoffeeTable` etc.



Ahora, ¿qué hay de las variantes del producto? Para cada variante de una familia de productos, creamos una clase de fábrica separada basada en la `AbstractFactory` interfaz. Una fábrica es una clase que devuelve productos de un tipo particular. Por ejemplo, `ModernFurnitureFactory`solo pueden crear objetos `ModernChair`y .`ModernSofaModernCoffeeTable`.

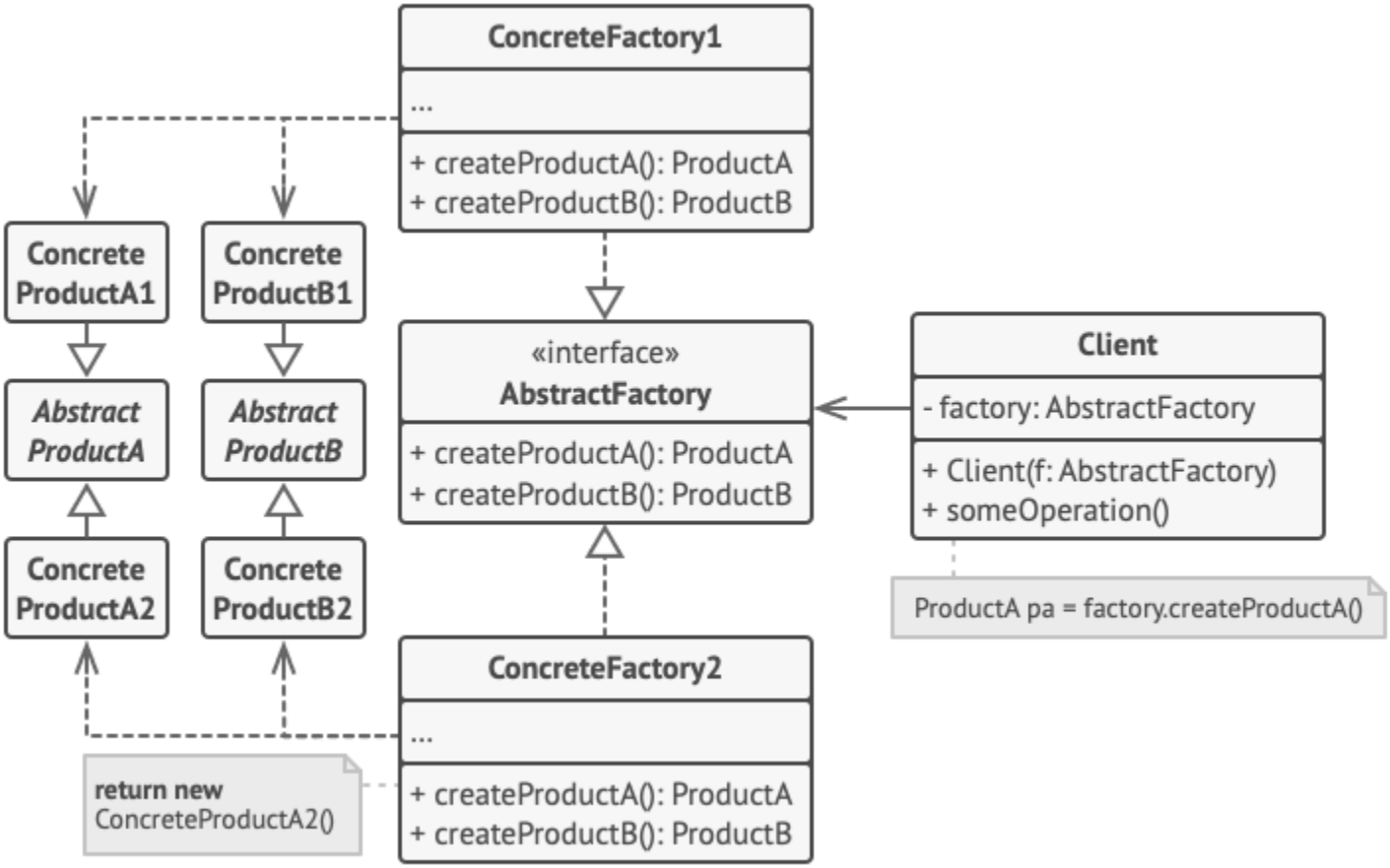
El código del cliente tiene que funcionar tanto con las fábricas como con los productos a través de sus respectivas interfaces abstractas. Esto le permite cambiar el tipo de fábrica que pasa al código de cliente, así como la variante de producto que recibe el código de cliente, sin romper el código de cliente real.

Digamos que el cliente quiere una fábrica para producir una silla. El cliente no tiene por qué ser consciente de la clase de la fábrica, ni le importa el tipo de silla que reciba. Ya sea un modelo moderno o una silla de estilo victoriano, el cliente debe tratar todas las sillas de la misma manera, utilizando la Chair interfaz abstracta.

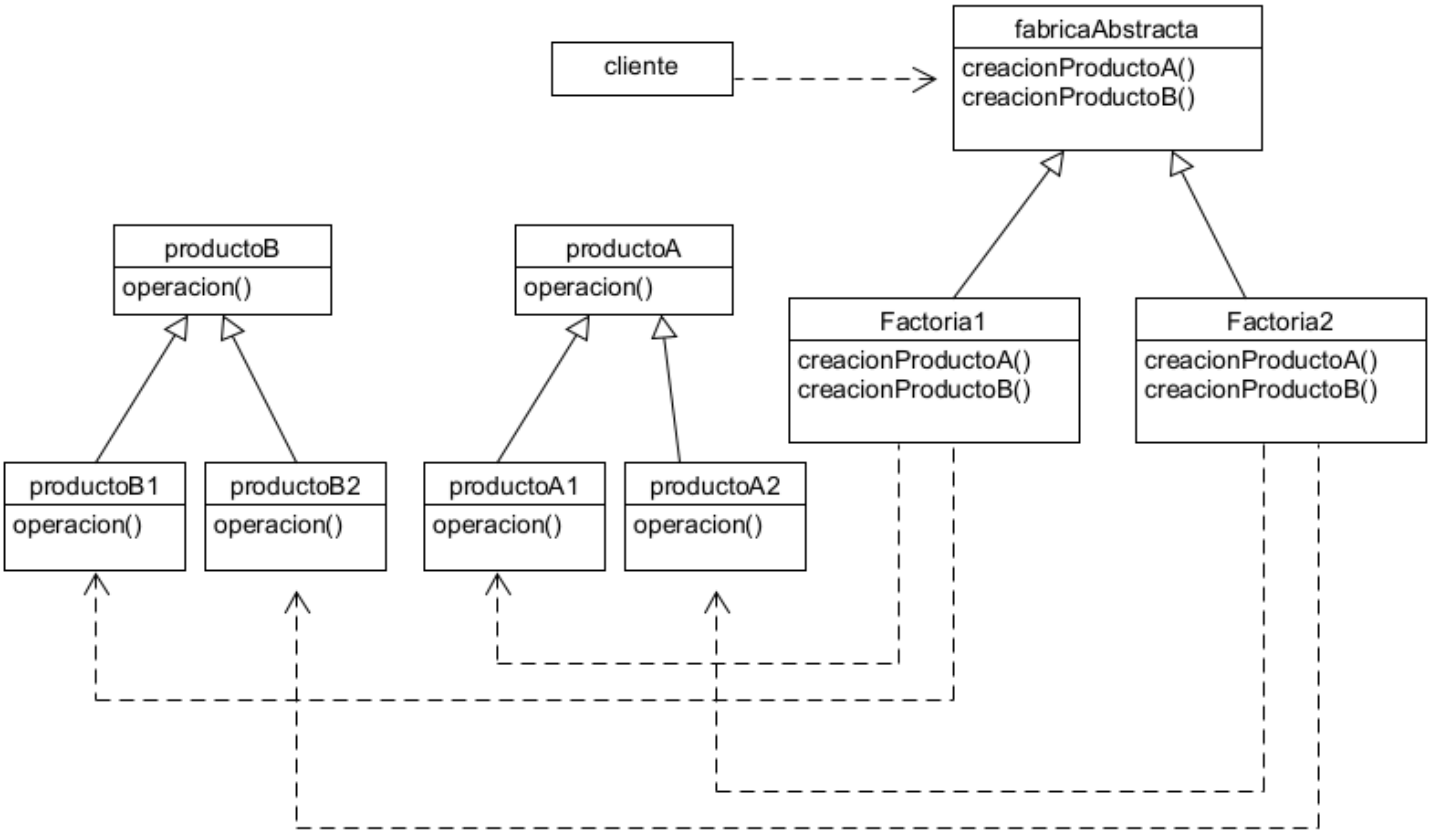
Con este enfoque, lo único que el cliente sabe sobre la silla es que implementa el método de alguna manera. Además, sea cual sea la variante de la silla que se devuelva, siempre coincidirá con el tipo de sofá o mesa de centro producido por el mismo objeto de fábrica.

Queda una cosa más por aclarar: si el cliente solo está expuesto a las interfaces abstractas, ¿qué crea los objetos de fábrica reales? Por lo general, la aplicación crea un objeto de fábrica concreto en la etapa de inicialización. Justo antes de eso, la aplicación debe seleccionar el tipo de fábrica según la configuración o la configuración del entorno.

Estructura



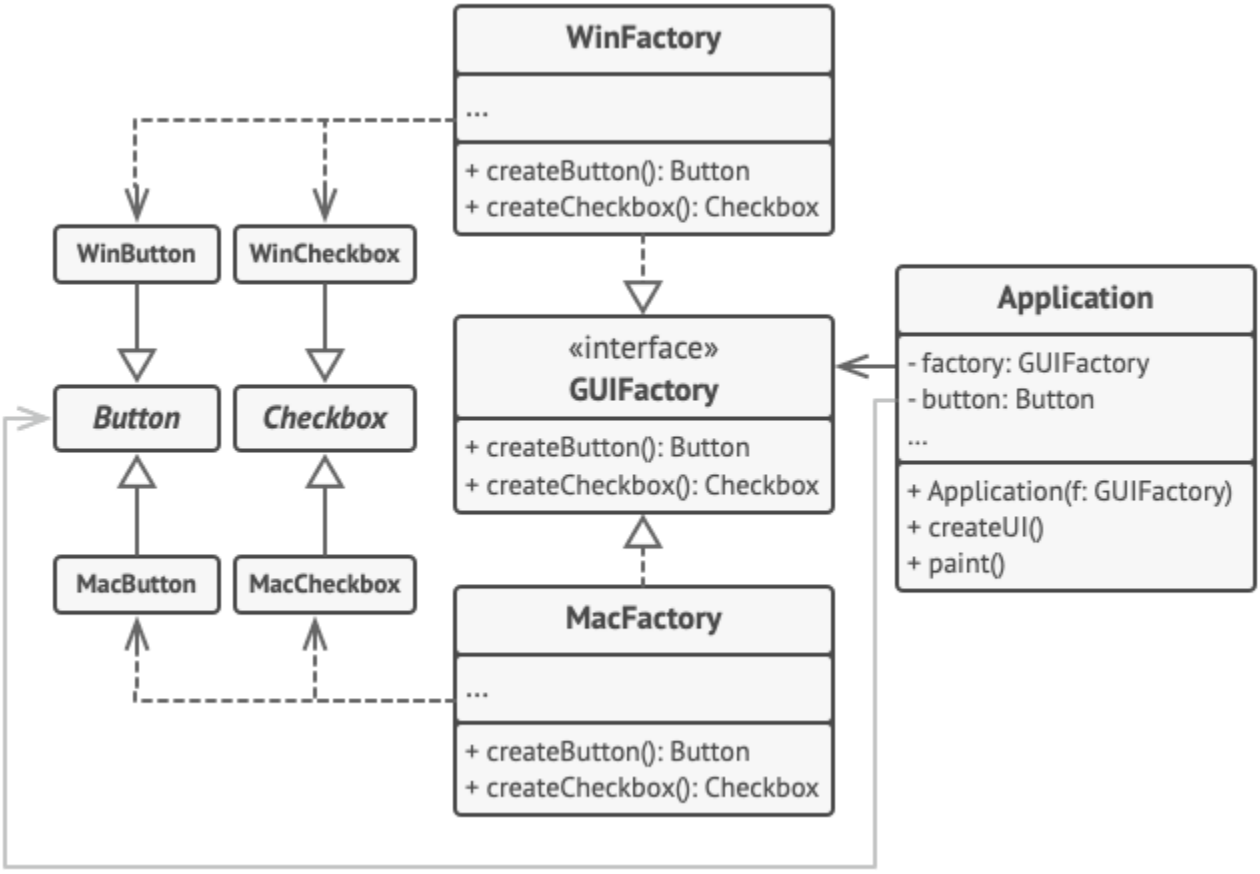
A continuación se presenta el mismo diagrama anterior, solo que acomodado en forma diferente:



1. Los **Productos Abstractos** declaran interfaces para un grupo de productos diferentes pero relacionados que forman una familia de productos.
2. La interfaz **Fábrica Abstracta** declara un grupo de métodos para crear cada uno de los productos abstractos.
3. Los **Productos Concretos** son implementaciones distintas de productos abstractos agrupados por variantes. Cada producto abstracto (silla/sofá) debe implementarse en todas las variantes dadas (victoriano/moderno).
4. Los **Productos Abstractos** declaran interfaces para un grupo de productos diferentes pero relacionados que forman una familia de productos.
5. Aunque las fábricas concretas instancian productos concretos, las firmas de sus métodos de creación deben devolver los productos abstractos correspondientes. De este modo, el código cliente que utiliza una fábrica no se acopla a la variante específica del producto que obtiene de una fábrica. El Cliente puede funcionar con cualquier variante fábrica/producto concreta, siempre y cuando se comunique con sus objetos a través de interfaces abstractas.

Ejemplo Interfaz de Usuario

Este ejemplo ilustra cómo puede utilizarse el patrón **Abstract Factory** para crear elementos de interfaz de usuario (UI) multiplataforma sin acoplar el código cliente a clases UI concretas, mientras se mantiene la consistencia de todos los elementos creados respecto al sistema operativo seleccionado.



Es de esperar que los mismos elementos UI de una aplicación multiplataforma se comporten de forma parecida, aunque tengan un aspecto un poco diferente en distintos sistemas operativos. Además, es nuestro trabajo que los elementos UI coincidan con el estilo del sistema operativo en cuestión. No queremos que nuestro programa represente controles de macOS al ejecutarse en Windows.

La interfaz fábrica abstracta declara un grupo de métodos de creación que el código cliente puede utilizar para producir distintos tipos de elementos UI. Las fábricas concretas coinciden con sistemas operativos específicos y crean los elementos UI correspondientes.

Funciona así: cuando se lanza, la aplicación comprueba el tipo de sistema operativo actual. La aplicación utiliza esta información para crear un objeto de fábrica a partir de una clase que coincida con el sistema operativo. El resto del código utiliza esta fábrica para crear elementos UI. Esto evita que se creen elementos equivocados.

Con este sistema, el código cliente no depende de clases concretas de fábricas y elementos UI, siempre y cuando trabaje con estos objetos a través de sus interfaces abstractas. Esto también permite que el código cliente soporte otras fábricas o elementos UI que pudiéramos añadir más adelante.

Como consecuencia, no necesitas modificar el código cliente cada vez que añades una nueva variedad de elementos UI a tu aplicación. Tan solo debes crear una nueva clase de fábrica que produzca estos elementos y modifique ligeramente el código de inicialización de la aplicación, de modo que seleccione esa clase cuando resulte apropiado.

Aplicabilidad

Utiliza el patrón Abstract Factory cuando tu código deba funcionar con varias familias de productos relacionados, pero no desees que dependa de las clases concretas de esos productos, ya que puede ser que no los conozcas de antemano o sencillamente quieras permitir una futura extensibilidad.

El patrón Abstract Factory nos ofrece una interfaz para crear objetos a partir de cada clase de la familia de productos. Mientras tu código cree objetos a través de esta interfaz, no tendrás que preocuparte por crear la variante errónea de un producto que no combine con los productos que ya ha creado tu aplicación.

En un programa bien diseñado *cada clase es responsable tan solo de una cosa*. Cuando una clase lidia con varios tipos de productos, puede ser que valga la pena extraer sus métodos de fábrica para ponerlos en una clase única de fábrica o una implementación completa del patrón Abstract Factory

Relaciones con otros patrones

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).

Recursos y fuentes:

- https://www.youtube.com/watch?v=CVlpjFJN17U

- https://refactoring.guru/es/design-patterns/abstract-factory