

PATRONES DE DISEÑO

Adrián A. Alarcón O.

PATRONES DE DISEÑO

Es una solución de diseño general, reutilizable y aplicable a problemas comunes de software.

PATRONES DE DISEÑO

Desarrolladores se han enfrentado a problemas a lo largo del tiempo y a través de prueba y error han encontrado soluciones.

PATRONES DE DISEÑO

En 1994, Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, a los que llamaron Gang of Four (GoF), publicaron un libro titulado Design Patterns, elementos de software orientado a objetos reutilizables.

Contiene 23 patrones de diseño comunes.

TIPOS DE PATRONES



ABSTRACT FACTORY

ABSTRACT FACTORY










En este patrón, una interfaz crea conjuntos o familias de objetos relacionados sin especificar el nombre de la clase.

EL PROBLEMA

Imagina que estás creando un simulador de tienda de muebles. Su código consta de clases que representan:

- Familia de productos: **Chair, Sofa, CoffeeTable**
- Variantes: **Modern, Victorian, ArtDeco**

EL PROBLEMA

	Chair	Sofa	Coffee Table
Art Deco			
Victorian			
Modern			

EL PROBLEMA

Necesita una forma de crear objetos de mobiliario individuales para que coincidan con otros objetos de la misma familia.

Los clientes se enojan mucho cuando reciben muebles que no combinan.

EL PROBLEMA

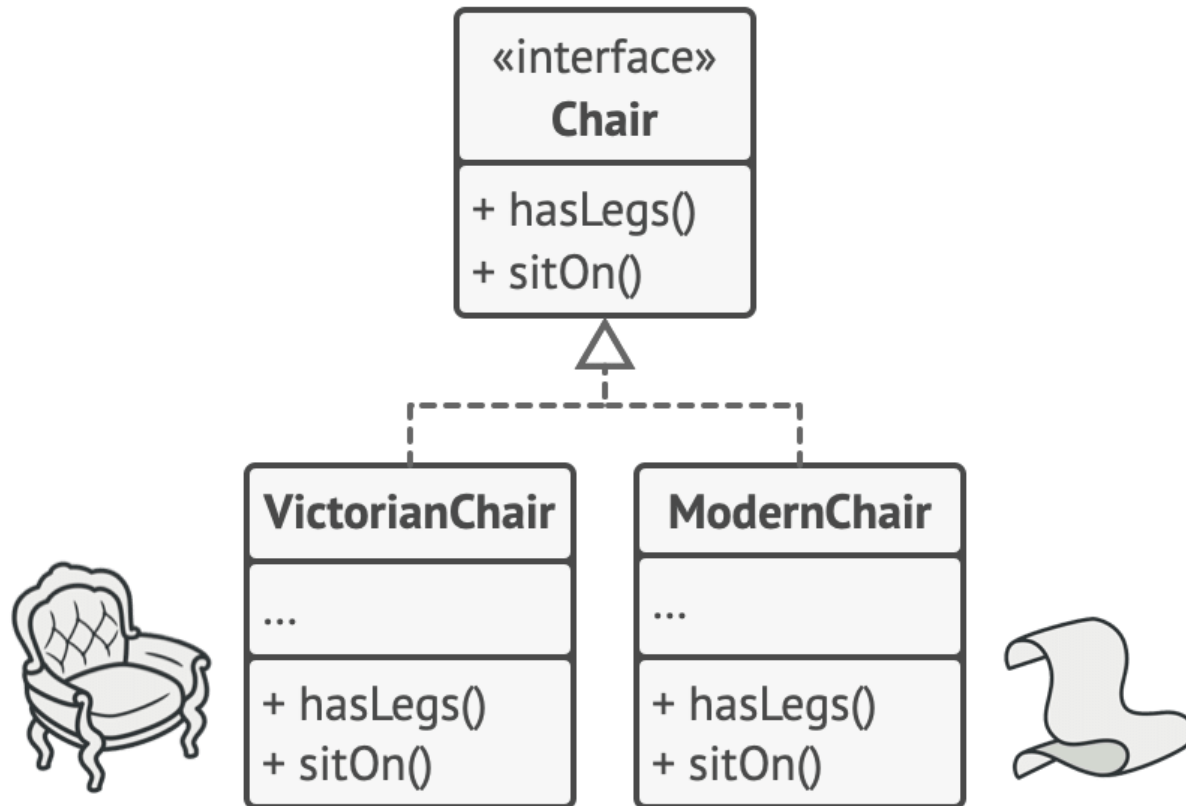
Además, no desea cambiar el código existente al agregar nuevos productos o familias de productos al programa. Los proveedores de muebles actualizan sus catálogos con mucha frecuencia y usted no querrá cambiar el código principal cada vez que sucede.

LA SOLUCIÓN

Primero declarar explícitamente interfaces para cada producto distinto de la familia de productos.

Luego, puede hacer que todas las variantes de productos sigan esas interfaces.

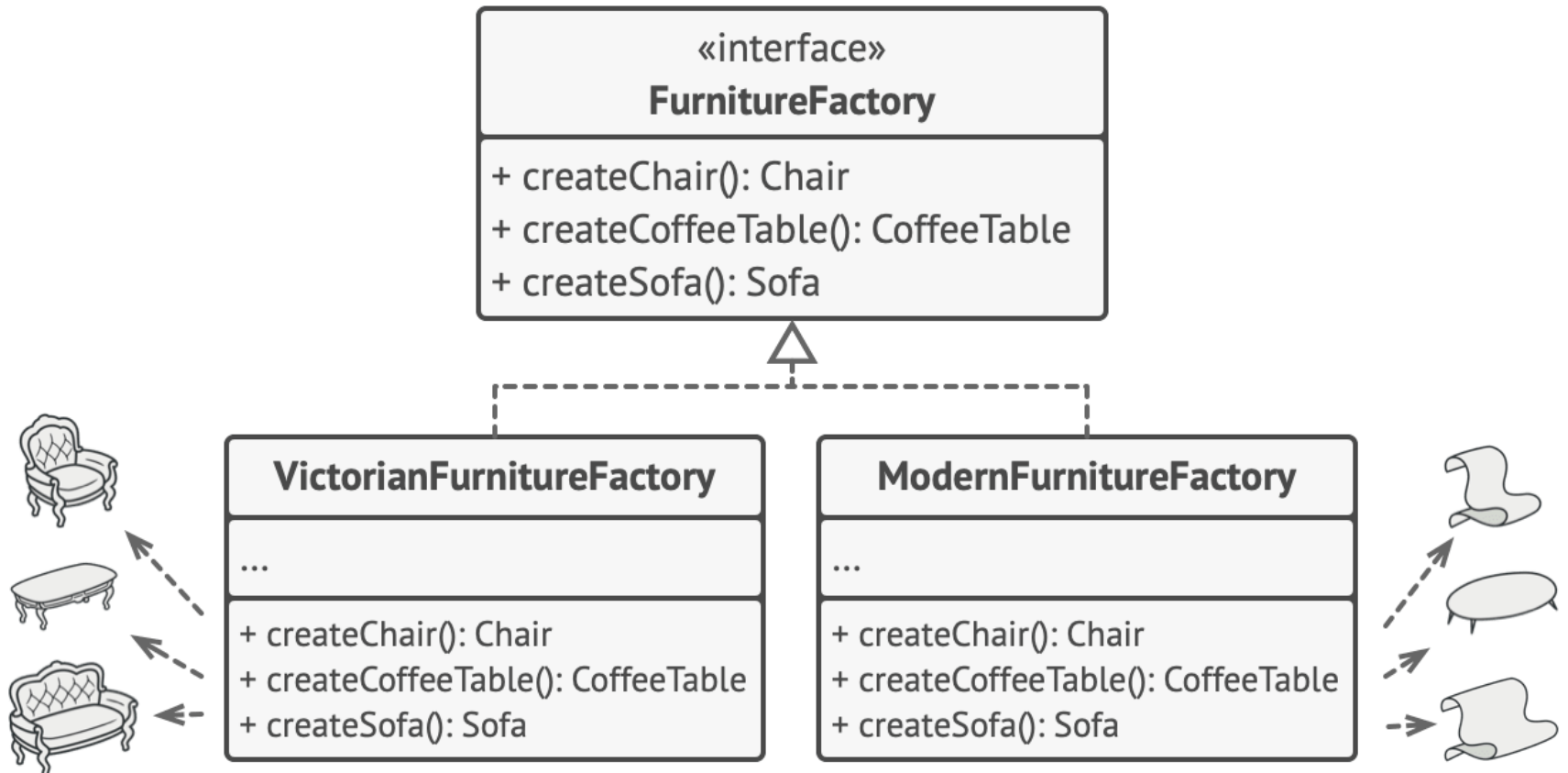
LA SOLUCIÓN



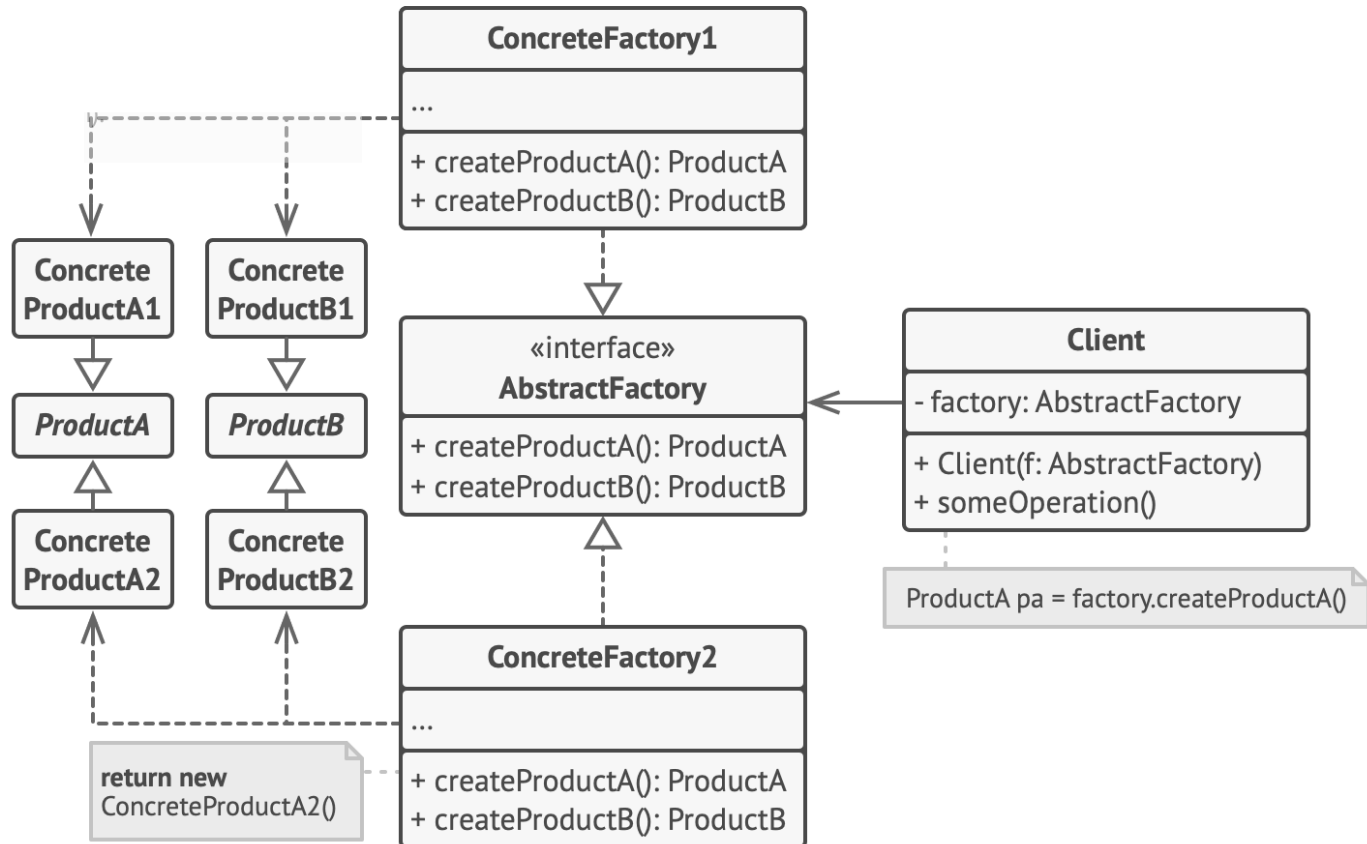
LA SOLUCIÓN

El siguiente paso es declarar una interfaz con una lista de métodos de creación para todos los productos que forman parte de la familia de productos

LA SOLUCIÓN



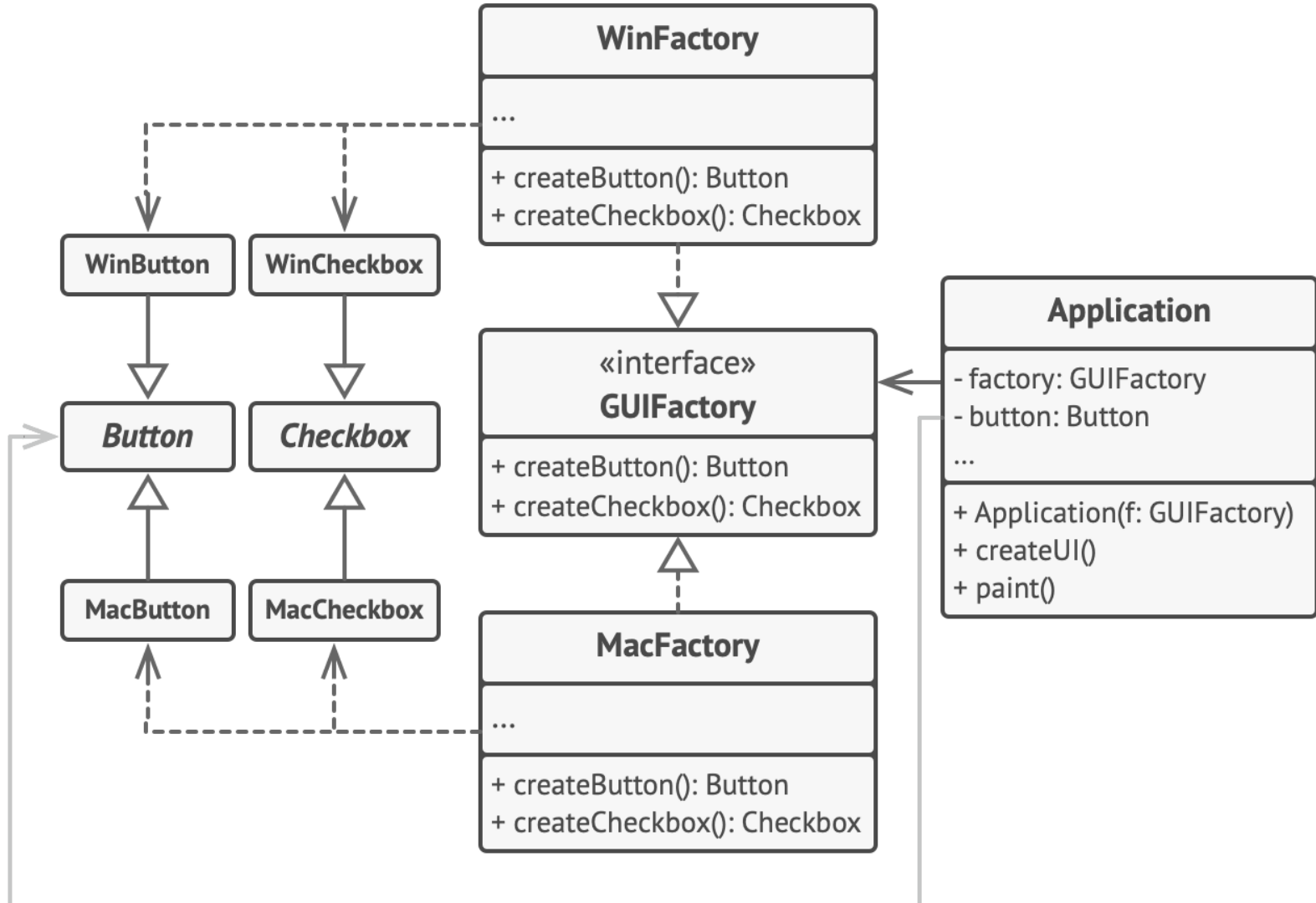
ESTRUCTURA



EJEMPLO

Tenemos una aplicación que se va a encargar de crear botones para Windows o para Mac

LA SOLUCIÓN



LA SOLUCIÓN

```
interface GUIFactory is
    method createButton():Button
    method createCheckbox():Checkbox

class WinFactory implements GUIFactory is
    method createButton():Button is
        return new WinButton()
    method createCheckbox():Checkbox is
        return new WinCheckbox()

class MacFactory implements GUIFactory is
    method createButton():Button is
        return new MacButton()
    method createCheckbox():Checkbox is
        return new MacCheckbox()
```

LA SOLUCIÓN

```
interface Button  
    method paint();
```

```
interface Checkbox  
    method paint();
```

LA SOLUCIÓN

```
class WinButton implements Button  
    method paint();
```

```
class MacButton implements Button is  
    method paint();
```

LA SOLUCIÓN

```
class WinCheckbox implements Checkbox  
    method paint();
```

```
class MacCheckbox implements Checkbox is  
    method paint();
```

LA SOLUCIÓN

```
class Application is
    private field factory: GUIFactory
    private field button: Button

    constructor Application(factory: GUIFactory)
        this.factory = factory;

    method createUI();
        this.button = factory.createButton()

    method paint();
        button.paint();
```

LA SOLUCIÓN

```
class ApplicationConfigurator is
  method main() is
    config = readApplicationConfigFile();

    if (config.OS == "Windows") then
      factory = new WinFactory();
    else if (config.OS == "Mac") then
      factory = new MacFactory();
    else
      throw new Exception("Error! Unknown operating system.");

    Application app = new Application(factory)
```