Abstract Factory

Programación II

Leonardo Camacho

Annet Jiménez Montero/ C04051

Abstract Factory

**Abstract Factory** es un patrón de diseño creacional que resuelve el problema de crear familias enteras de productos sin especificar sus clases concretas.

El patrón Abstract Factory define una interfaz para crear todos los productos, pero deja la propia creación de productos para las clases de fábrica concretas.

// Las fábricas concretas producen una familia de productos que

// pertenecen a una única variante. La fábrica garantiza que los

// productos resultantes sean compatibles. Las firmas de los

// métodos de las fábricas concretas devuelven un producto

// abstracto mientras que dentro del método se instancia un

// producto concreto.

**class** **WinFactory** **implements** GUIFactory **is**

**method** createButton():Button **is**

**return** **new** WinButton()

**method** createCheckbox():Checkbox **is**

**return** **new** WinCheckbox()

// Cada fábrica concreta tiene una variante de producto

// correspondiente.

**class** **MacFactory** **implements** GUIFactory **is**

**method** createButton():Button **is**

**return** **new** MacButton()

**method** createCheckbox():Checkbox **is**

**return** **new** MacCheckbox()

// Cada producto individual de una familia de productos debe

// tener una interfaz base. Todas las variantes del producto

// deben implementar esta interfaz.

**interface** **Button** **is**

**method** paint()

// Los productos concretos son creados por las fábricas

// concretas correspondientes.

**class** **WinButton** **implements** Button **is**

**method** paint() **is**

// Representa un botón en estilo Windows.

**class** **MacButton** **implements** Button **is**

**method** paint() **is**

// Representa un botón en estilo macOS.

// Aquí está la interfaz base de otro producto. Todos los

// productos pueden interactuar entre sí, pero sólo entre

// productos de la misma variante concreta es posible una

// interacción adecuada.

**interface** **Checkbox** **is**

**method** paint()

**class** **WinCheckbox** **implements** Checkbox **is**

**method** paint() **is**

// Representa una casilla en estilo Windows.

**class** **MacCheckbox** **implements** Checkbox **is**

**method** paint() **is**

// Representa una casilla en estilo macOS.

// El código cliente funciona con fábricas y productos

// únicamente a través de tipos abstractos: GUIFactory, Button y

// Checkbox. Esto te permite pasar cualquier subclase fábrica o

// producto al código cliente sin descomponerlo.

**class** **Application** **is**

**private** **field** factory: GUIFactory

**private** **field** button: Button

**constructor** Application(factory: GUIFactory) **is**

**this**.factory = factory

**method** createUI() **is**

**this**.button = factory.createButton()

**method** paint() **is**

button.paint()

// La aplicación elige el tipo de fábrica dependiendo de la

// configuración actual o de los ajustes del entorno y la crea

// durante el tiempo de ejecución (normalmente en la etapa de

// inicialización).

**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)

Cómo implementarlo

1. Mapea una matriz de distintos tipos de productos frente a variantes de dichos productos.
2. Declara interfaces abstractas de producto para todos los tipos de productos. Después haz que todas las clases concretas de productos implementen esas interfaces.
3. Declara la interfaz de la fábrica abstracta con un grupo de métodos de creación para todos los productos abstractos.
4. Implementa un grupo de clases concretas de fábrica, una por cada variante de producto.
5. Crea un código de inicialización de la fábrica en algún punto de la aplicación. Deberá instanciar una de las clases concretas de la fábrica, dependiendo de la configuración de la aplicación o del entorno actual. Pasa este objeto de fábrica a todas las clases que construyen productos.
6. Explora el código y encuentra todas las llamadas directas a constructores de producto. Sustitúyelas por llamadas al método de creación adecuado dentro del objeto de fábrica.

Pros y contras

* Puedes tener la certeza de que los productos que obtienes de una fábrica son compatibles entre sí.
* Evitas un acoplamiento fuerte entre productos concretos y el código cliente.
* *Principio de responsabilidad única*. Puedes mover el código de creación de productos a un solo lugar, haciendo que el código sea más fácil de mantener.
* *Principio de abierto/cerrado*. Puedes introducir nuevas variantes de productos sin descomponer el código cliente existente.
* (X) Puede ser que el código se complique más de lo que debería, ya que se introducen muchas nuevas interfaces y clases junto al patrón.