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.

```
// La interfaz fábrica abstracta declara un grupo de métodos que
// devuelven distintos productos abstractos. Estos productos se
// denominan familia y están relacionados por un tema o concepto
// de alto nivel. Normalmente, los productos de una familia
// pueden colaborar entre sí. Una familia de productos puede
// tener muchas variantes, pero los productos de una variante
// son incompatibles con los productos de otra.
interface GUIFactory is
  method createButton():Button
```

```
    method createCheckbox():Checkbox


// 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)
```

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.