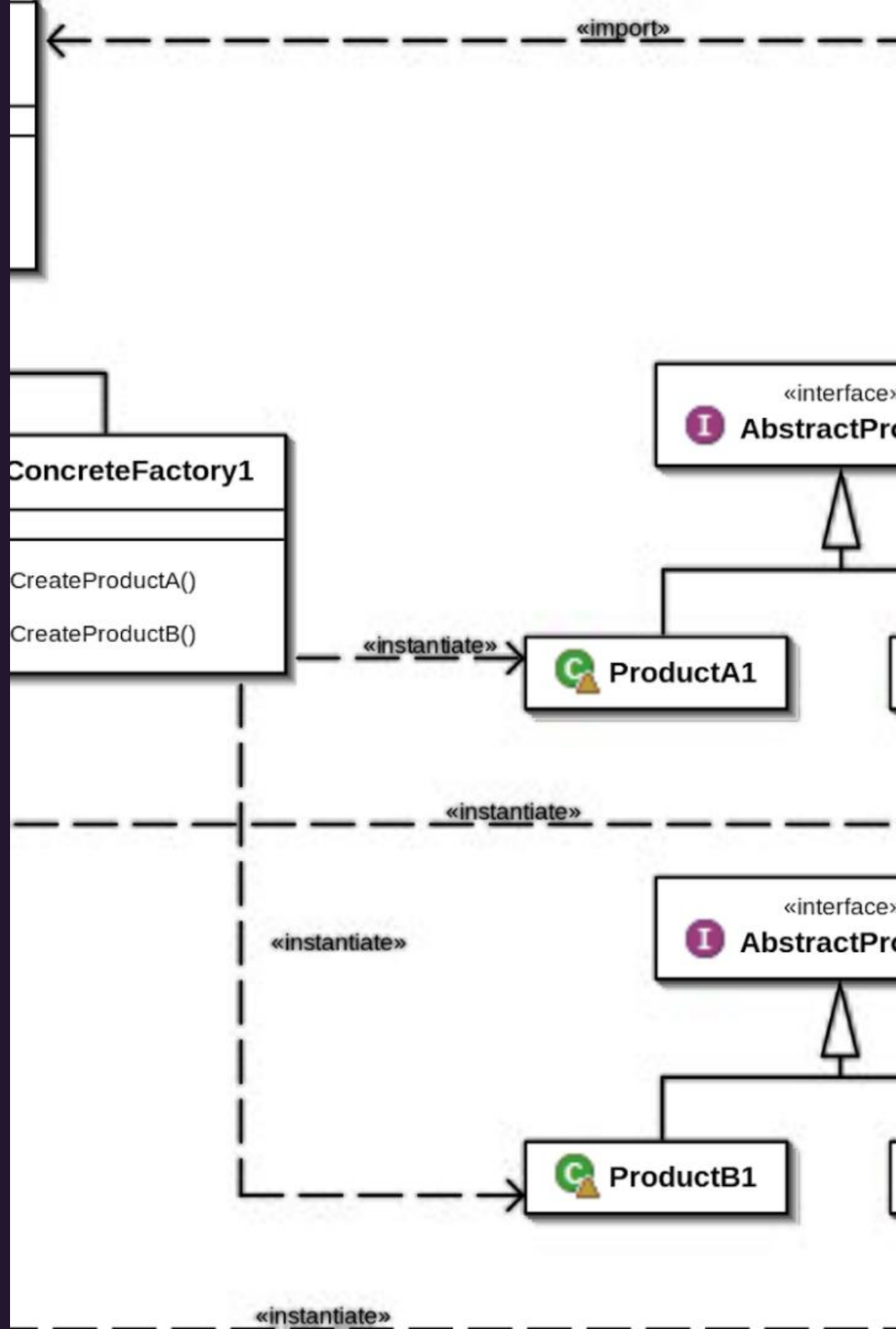
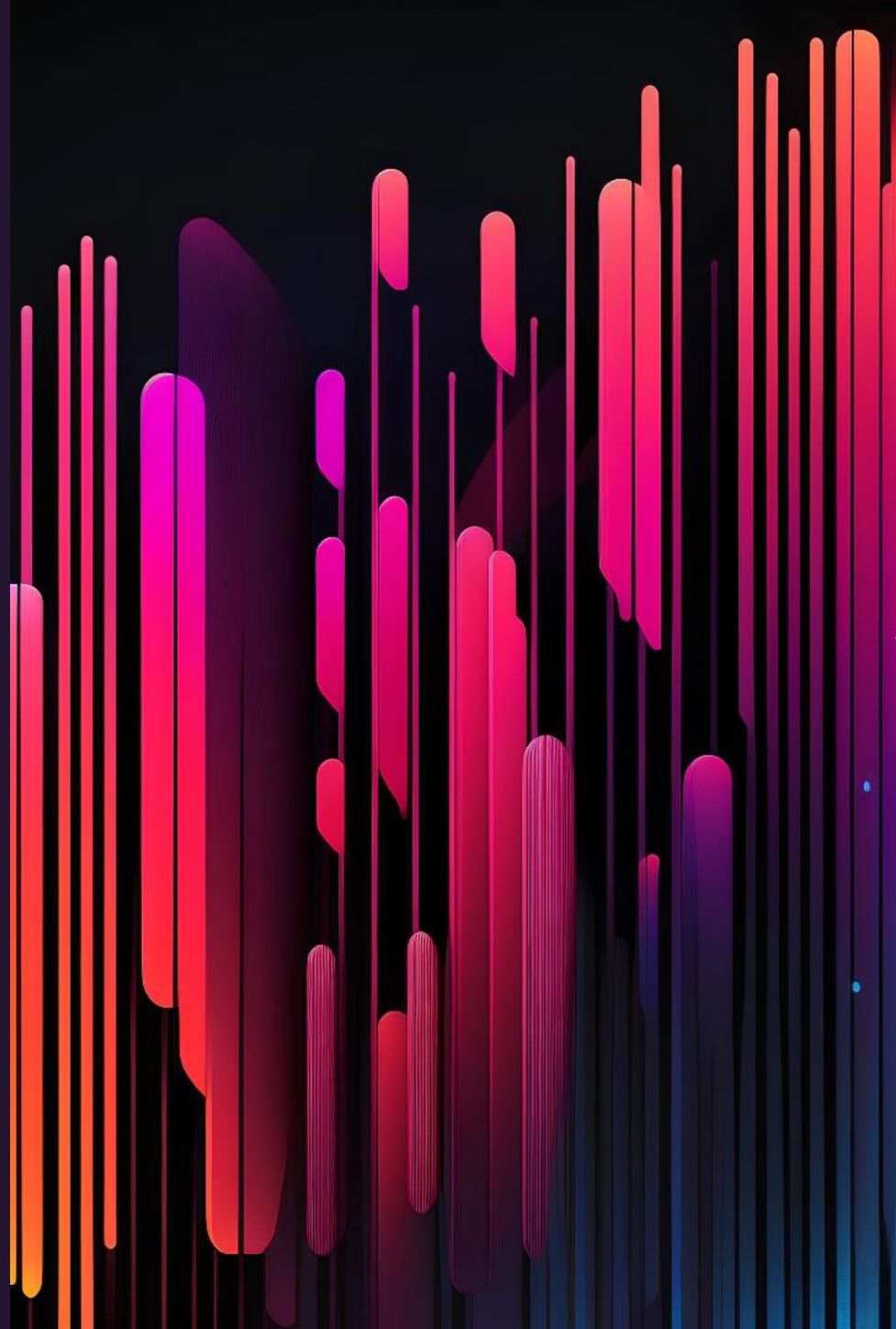
Patrón de Diseño: Abstract Factory



Definición

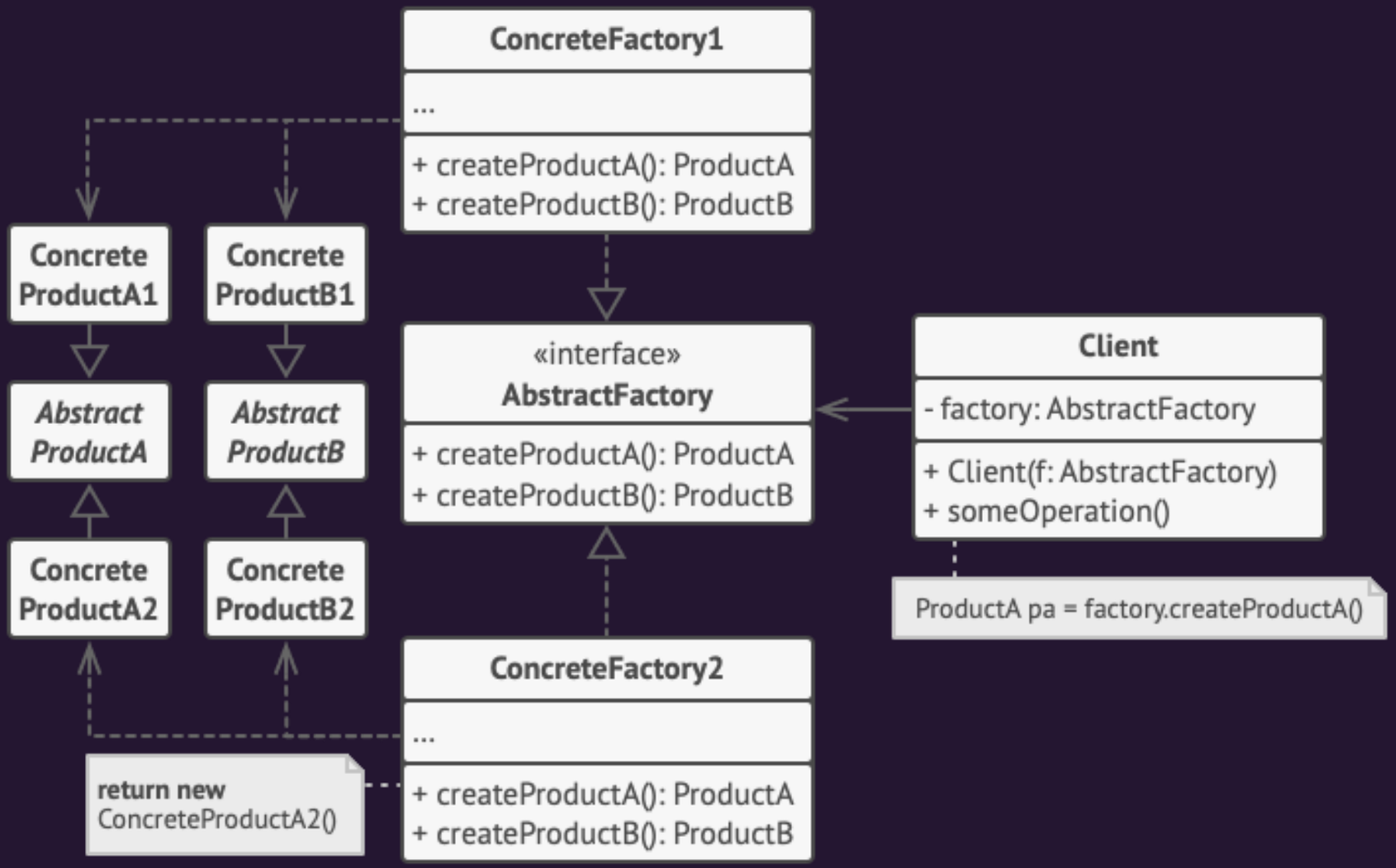
Abstract Factory es un patrón de diseño creacional que nos permite producir familias de objetos relacionados sin especificar sus clases concretas.



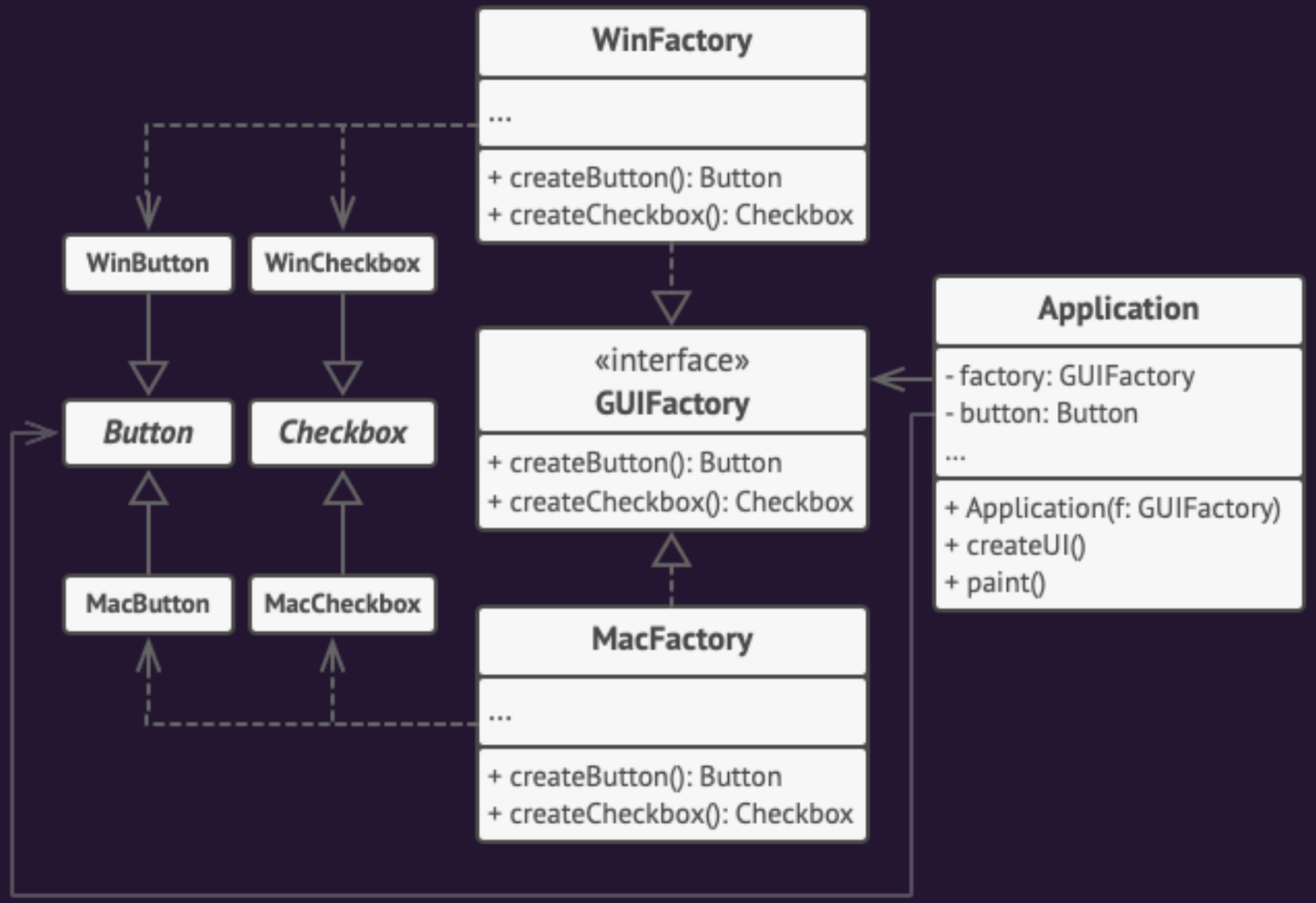
Ejemplo de uso en el mundo real

Imaginemos que estamos diseñando un software para una empresa de automóviles que produce autos de lujo y autos deportivos. Cada tipo tiene características específicas, como motores, interiores y diseños exteriores distintos.

Estructura


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
Abstract ProductA
Abstract ProductB
ProductA pa = factory.createProductA()
Concrete ProductA2
Concrete ProductB2
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA2()

Estructura


WinFactory
...
+ createButton(): Button
+ createCheckbox(): Checkbox
WinButton
WinCheckbox
Application
- factory: GUIFactory
- button: Button
...
+ Application(f: GUIFactory)
+ createUI()
+ paint()
«interface»
GUIFactory
+ createButton(): Button
+ createCheckbox(): Checkbox
Button
Checkbox
MacButton
MacCheckbox
MacFactory
...
+ createButton(): Button
+ createCheckbox(): Checkbox

Código

```python
# Define la interfaz fábrica abstracta
class GUIFactory:
    def create_button(self):
        pass

    def create_checkbox(self):
        pass

# Implementa la fábrica concreta para Windows
class WinFactory(GUIFactory):
    def create_button(self):
        return WinButton()

    def create_checkbox(self):
        return WinCheckbox()

# Implementa la fábrica concreta para macOS
class MacFactory(GUIFactory):
    def create_button(self):
        return MacButton()

    def create_checkbox(self):
        return MacCheckbox()

# Define la interfaz base para el producto Button
class Button:
    def paint(self):
        pass
```

```python
# Implementa productos concretos para Button
class WinButton(Button):
    def paint(self):
        print("Dibujando un botón en estilo Windows")

class MacButton(Button):
    def paint(self):
        print("Dibujando un botón en estilo macOS")

# Define la interfaz base para el producto Checkbox
class Checkbox:
    def paint(self):
        pass

# Implementa productos concretos para Checkbox
class WinCheckbox(Checkbox):
    def paint(self):
        print("Dibujando una casilla en estilo Windows"

class MacCheckbox(Checkbox):
    def paint(self):
        print("Dibujando una casilla en estilo macOS")
```

```python
# La aplicación cliente
class Application:
    def __init__(self, factory):
        self.factory = factory
        self.button = None

    def create_ui(self):
        self.button = self.factory.create_button()

    def paint(self):
        self.button.paint()

# Uso de las clases sin la configuración del sistema operativ
factory = WinFactory()  # O puedes usar MacFactory()
app = Application(factory)
app.create_ui()
app.paint()
```

Ventajas del Uso del Patrón

1 **Separación de Responsabilidades**

Promueve la separación de responsabilidades al separar la creación de objetos de su uso.

2 **Coherencia en la Creación**

Puedes tener la certeza de que los productos que obtienes de una fábrica son compatibles entre sí.

3 **Flexibilidad y Extensibilidad**

Facilita la introducción de nuevas familias de productos o variantes sin modificar el código existente.

Desventajas del Uso del Patrón

1 **Mayor Complejidad**

Puede ser que el código se complique más de lo que debería.

2 **Dificultad en la reutilización**

Puede ser complicado reutilizar las fábricas en diferentes contextos si no se diseñan cuidadosamente para ser genéricas.

Gracias!