Practica 6. Factory y Observer

Jorge Andrés Hernández Rentería

2025-09-29

“ “ ” PATRÓN FACTORY - Fábrica de Notificaciones Propósito: Centralizar la creación de objetos. En lugar de crear objetos directamente, usamos una fábrica que decide qué tipo de objeto crear. “ “ ”

```python
from abc import ABC, abstractmethod

# Paso 1: Crear una interfaz común para todos los productos
class Notificacion(ABC):
    @abstractmethod
    def enviar(self, mensaje: str):
        pass

# Paso 2: Crear implementaciones concretas
class EmailNotificacion(Notificacion):
    def enviar(self, mensaje: str):
        print(f" Enviando EMAIL: {mensaje}")

class SMSNotificacion(Notificacion):
    def enviar(self, mensaje: str):
        print(f" Enviando SMS: {mensaje}")

class PushNotificacion(Notificacion):
    def enviar(self, mensaje: str):
        print(f" Enviando PUSH: {mensaje}")

# Paso 3: Crear la FÁBRICA
class FabricaNotificaciones:
    """
    La fábrica es como un menú de restaurante:
    - Tú pides "email", "sms" o "push"
    - La fábrica te da el objeto correcto
```

```
    - No necesitas saber cómo se crea cada uno
    """

    def crear_notificacion(self, tipo: str) -> Notificacion:
        if tipo == "email":
            return EmailNotificacion()
        elif tipo == "sms":
            return SMSNotificacion()
        elif tipo == "push":
            return PushNotificacion()
        else:
            raise ValueError("Tipo de notificación no válido")

# Paso 4: Usar la fábrica
def demo_factory():
    print("  DEMO PATRÓN FACTORY")
    print("=" * 40)

    fabrica = FabricaNotificaciones()

    # Pedimos notificaciones a la fábrica
    email = fabrica.crear_notificacion("email")
    sms = fabrica.crear_notificacion("sms")
    push = fabrica.crear_notificacion("push")

    # Usamos las notificaciones sin saber su tipo concreto
    email.enviar("¡Bienvenido al banco!")
    sms.enviar("Código de verificación: 123456")
    push.enviar("Tienes una transferencia pendiente")

"""
  VENTAJAS DEL FACTORY:
• Oculta la complejidad de creación
• Fácil agregar nuevos tipos (WhatsApp, Telegram)
• Código más limpio y mantenible
"""

demo_factory()

"""
PATRÓN OBSERVER - Sistema de Alertas Bancarias
Propósito: Notificar automáticamente a múltiples objetos cuando algo cambia.
```

```
Como una lista de suscriptores que reciben notificaciones.
"""

from abc import ABC, abstractmethod
from typing import List

# Paso 1: Definir el Observador (quien recibe notificaciones)
class Observador(ABC):
    @abstractmethod
    def actualizar(self, mensaje: str):
        pass

# Paso 2: Implementar observadores concretos
class Cliente(Observador):
    def __init__(self, nombre: str):
        self.nombre = nombre

    def actualizar(self, mensaje: str):
        print(f"  {self.nombre} recibió: {mensaje}")

class DepartamentoFraude(Observador):
    def actualizar(self, mensaje: str):
        if "sospechosa" in mensaje.lower():
            print(f"  DEPARTAMENTO FRAUDE - ALERTA: {mensaje}")

class Auditoria(Observador):
    def actualizar(self, mensaje: str):
        print(f"  AUDITORÍA registró: {mensaje}")

# Paso 3: Crear el Subject (quien notifica)
class CuentaBancaria:
    """
    El Subject es como un YouTuber:
    - Tiene suscriptores (observadores)
    - Cuando publica nuevo contenido (cambia estado), notifica a todos
    """

    def __init__(self, numero: str):
        self.numero = numero
        self.saldo = 0
        self.observadores: List[Observador] = []  # Lista de suscriptores
```

```
    def agregar_observador(self, observador: Observador):
        """Agregar un nuevo suscriptor"""
        self.observadores.append(observador)

    def eliminar_observador(self, observador: Observador):
        """Eliminar un suscriptor"""
        self.observadores.remove(observador)

    def notificar_observadores(self, mensaje: str):
        """Notificar a TODOS los suscriptores"""
        for observador in self.observadores:
            observador.actualizar(mensaje)

    def depositar(self, monto: float):
        """Cuando depositamos, notificamos a todos"""
        self.saldo += monto
        mensaje = f"Depósito de ${monto} en cuenta {self.numero}. Saldo: ${self.saldo}"
        self.notificar_observadores(mensaje)

    def retirar(self, monto: float):
        """Cuando retiramos, notificamos a todos"""
        if monto > self.saldo:
            mensaje = f"INTENTO DE RETIRO SOSPECHOSO de ${monto} en cuenta {self.numero}"
        else:
            self.saldo -= monto
            mensaje = f"Retiro de ${monto} en cuenta {self.numero}. Saldo: ${self.saldo}"

        self.notificar_observadores(mensaje)

# Paso 4: Usar el patrón Observer
def demo_observer():
    print("\n DEMO PATRÓN OBSERVER")
    print("=" * 40)

    # Crear cuenta bancaria
    cuenta = CuentaBancaria("123-456")

    # Crear observadores (suscriptores)
    cliente = Cliente("Juan Pérez")
    fraude = DepartamentoFraude()
    auditoria = Auditoria()
```

```
    # Suscribir observadores a la cuenta
    cuenta.agregar_observador(cliente)
    cuenta.agregar_observador(fraude)
    cuenta.agregar_observador(auditoria)

    print(" Realizando transacciones...")

    # Las transacciones notificarán automáticamente a todos
    cuenta.depositar(1000)
    cuenta.retirar(200)
    cuenta.retirar(5000)  # Transacción sospechosa

"""
 VENTAJAS DEL OBSERVER:
• Desacopla el objeto que cambia de los que reaccionan
• Fácil agregar nuevos observadores
• Notificaciones automáticas en tiempo real
"""

demo_observer()
```

```
 DEMO PATRÓN FACTORY
========================================
 Enviando EMAIL: ¡Bienvenido al banco!
 Enviando SMS: Código de verificación: 123456
 Enviando PUSH: Tienes una transferencia pendiente

 DEMO PATRÓN OBSERVER
========================================
 Realizando transacciones...
 Juan Pérez recibió: Depósito de $1000 en cuenta 123-456. Saldo: $1000
 AUDITORÍA registró: Depósito de $1000 en cuenta 123-456. Saldo: $1000
 Juan Pérez recibió: Retiro de $200 en cuenta 123-456. Saldo: $800
 AUDITORÍA registró: Retiro de $200 en cuenta 123-456. Saldo: $800
 Juan Pérez recibió: INTENTO DE RETIRO SOSPECHOSO de $5000 en cuenta 123-456
 AUDITORÍA registró: INTENTO DE RETIRO SOSPECHOSO de $5000 en cuenta 123-456
```

#EXPLICACIONES “ “ ” *Patrón FACTORY: Problema: Cuando crear objetos es complejo o puede cambiar

Solución: Una fábrica centralizada que crea objetos

Analogía: Menú de restaurante - pides “pizza” y te la traen sin saber cocinarla

Ventajas: Código más limpio, fácil mantenimiento, oculta complejidad

*Patrón OBSERVER: Problema: Cuando un cambio debe notificar a muchos objetos

Solución: Sistema de suscripción donde objetos se registran para recibir notificaciones

Analogía: Suscripción a YouTube - cuando suben video, todos los suscriptores son notificados

Ventajas: Desacoplamiento, notificaciones automáticas, fácil escalar

¿Cuándo usar cada uno? -FACTORY: Cuando la creación de objetos es compleja o puede variar

-OBSERVER: Cuando un cambio de estado debe notificar a múltiples componentes “ “ ”