Pattern Abuser

Tomás Felipe Melli

June 24, 2025

Índice

1 Pat's Confession

Pat cuenta que era un programador común que trabajaba mucho para que su código funcionara, sin técnicas de diseño formales, solo modificando hasta que el código servía.

Todo cambió cuando llegó Jerry (a quien llaman JERR), un diseñador experto que les presentó el libro Gang of Four con patrones de diseño. JERR les mostró cómo aplicar patrones para mejorar el código: hacerlo más entendible, modificable, testeable y mantenible.

Pat se volvió fanático de los patrones: memorizó todos, reescribió todo su código usando patrones, y empezó a exigir a los demás que hicieran lo mismo. Organizó revisiones de código que generaban miedo, porque criticaba duramente la ausencia de patrones.

Gracias a esto, convenció a los gerentes de que más patrones significaban mejor reutilización y calidad, lo que hizo que su proyecto fuera reconocido como el mejor de la empresa. Pat pasó de programador a diseñador estrella y consultor famoso, invitado a conferencias y contratado por muchas empresas.

Sin embargo, la obsesión de Pat creció al punto que quería aplicar la mayor cantidad posible de patrones a una sola clase, sin importar las consecuencias al código. Desarrolló su propia "técnica" para aplicar muchos patrones a un mismo componente, y planea mostrar un ejemplo con clases de cuentas bancarias.

Aplicación de Patrones de Diseño a una Jerarquía de Clases (Ejemplo con Cuentas Bancarias)

- Strategy Pattern para Métodos
  - Aplicar el Strategy Pattern a métodos como calcInterest.
  - Crear una clase estrategia InterestCalculator con métodos para calcular interés.
  - Definir subclases para distintos tipos de interés (simple, compuesto, fijo, variable).
  - La clase Account (o su estado) contiene una referencia a la estrategia específica para calcular el interés.
- Chain of Responsibility para Agrupaciones
  - Usar Chain of Responsibility para decidir dinámicamente qué instancia maneja una solicitud.
  - Por ejemplo, para una AccountPortfolio que agrupa cuentas, determinar desde qué cuenta retirar dinero cuando hay sobregiro, buscando en la cadena la cuenta con saldo suficiente.
- Composite, Iterator y Visitor para Agrupaciones
  - Composite organiza cuentas en una estructura árbol (nodos grupo y hojas).
  - Iterator permite iterar sobre las cuentas en grupos.
  - Visitor separa operaciones (como imprimir estados, calcular saldos) del objeto cuenta.
  - Agrupación posible: por tipo de cuenta, tipo de interés, tipo de cliente, etc.
- Observer para Clientes Dependientes
  - Clientes se registran para observar cambios en cuentas o portafolios.
  - Cuando una cuenta cambia (ej. saldo), notifica a los observadores, quienes actualizan sus vistas o datos.
- Proxy para Acceso Controlado y Distribuido
  - Crear proxies para Account y AccountPortfolio que representen servicios (e.g., cajero automático).
  - Proxies aceptan mensajes, invocan métodos reales y devuelven resultados, facilitando control o acceso remoto.
- Facade para Simplificar Interfaces
  - Definir una fachada (Account Facade) que agrupe la complejidad de múltiples clases y patrones aplicados, exponiendo una interfaz unificada (retirar, depositar, transferir, consultar saldos).
- Bridge para Métodos Complejos
  - Separar la interfaz del cálculo de interés de su implementación, especialmente si hay varias formas de calcular el mismo tipo (e.g., diferentes formas de interés compuesto).
- Adapter para Objetos Legados
  - Adaptar objetos antiguos para que cumplan con la nueva interfaz y puedan integrarse al sistema actual sin problemas.

- Abstract Factory, Builder y Factory Method para Creación de Objetos Familiares
  - Crear familias de objetos relacionados (Customer, AccountPortfolio, Account) usando estos patrones para controlar la creación y ensamblado.
  - El cliente controla el ensamblaje (Abstract Factory) o el patrón devuelve un producto final (Builder).
- Prototype para Crear Objetos Similares
  - Clonar prototipos para crear nuevas cuentas en una cartera, facilitando la creación rápida basada en un objeto modelo.

2 The Dark

Pat confiesa que, aunque al principio disfrutó del prestigio, fama y dinero que le trajo aplicar patrones en sus proyectos, empezó a cuestionar si realmente mejoraba los sistemas que construía.

En el caso del banco, aunque quedaron satisfechos que Pat aplicara muchos patrones, pronto los desarrolladores tuvieron dificultades para hacer cambios sencillos en el sistema. Pat justificó esto diciendo que los desarrolladores debían "madurar" en el uso de patrones y que la solución era agregar aún más patrones, lo que le valió ser contratado de nuevo.

Sin embargo, Pat se siente cada vez más frustrado y deprimido porque se da cuenta de que abusar de los patrones genera problemas: no sabe cuándo un patrón realmente aporta valor ni cómo medir si el sistema es más reutilizable o "mejor" tras su aplicación. Se cuestiona cómo definir y determinar si una aplicación ha mejorado.

Pat pide ayuda para resolver estas dudas.