APRENDE A IDENTIFICAR QUE PATRON DE DISEÑO APLICAR

Entender cuál patrón de diseño aplicar en un escenario específico requiere analizar cuidadosamente la situación y buscar pistas clave relacionadas con los problemas que los patrones resuelven. Aquí tienes una guía paso a paso para ayudarte a identificar cuál de los seis patrones (Factory Method, Facade, Proxy, State, Observer, Singleton) deberías aplicar:

1. Identifica el problema principal

- Pregúntate: ¿Cuál es el desafío principal que se describe en el escenario?
  - ¿Se trata de **crear objetos**? (Factory Method).
  - ¿Se requiere **gestionar el acceso a recursos compartidos** o garantizar una única instancia? (Singleton).
  - ¿Hay **muchas dependencias entre subsistemas** y necesitas simplificarlas? (Facade).
  - ¿Se requiere cargar algo **de forma diferida o bajo demanda**? (Proxy).
  - ¿El sistema tiene **estados cambiantes** que afectan su comportamiento? (State).
  - ¿Se necesita notificar a **múltiples objetos** sobre cambios en el sistema? (Observer).

2. Relaciona los patrones con sus características principales

Patrón	Cuándo Aplicar
Factory Method	Cuando no sabes qué clase exacta necesitas instanciar.
Facade	Cuando hay subsistemas complejos y necesitas simplificar el acceso a ellos.
Proxy	Cuando necesitas controlar el acceso a un recurso costoso o remoto.
State	Cuando el comportamiento del objeto cambia según su estado interno.
Observer	Cuando necesitas notificar automáticamente a varias entidades sobre cambios.
Singleton	Cuando necesitas garantizar que solo haya una instancia de una clase.

3. Analiza las pistas del escenario

Aquí hay algunos indicadores comunes que puedes encontrar en los escenarios y el patrón correspondiente:

Indicador en el escenario	Patrón Sugerido
"No sabemos de antemano qué tipo de objeto debemos crear."	Factory Method
"Queremos ocultar la complejidad de interactuar con múltiples subsistemas."	Facade
"Queremos evitar múltiples instancias de una clase."	Singleton
"El recurso tarda mucho en cargarse o es costoso, queremos cargarlo a demanda."	Proxy
"El sistema tiene diferentes comportamientos según su estado actual."	State
"Queremos que varios objetos sean notificados automáticamente sobre cambios."	Observer

4. Aplica la lógica a ejemplos concretos

Ejemplo 1: Winforms (Calculadora con múltiples ventanas abiertas)

- **Problema**: Múltiples instancias abiertas, pero queremos reutilizar una sola ventana.
- **Indicador clave**: Necesitamos una única instancia.
- **Patrón**: **Singleton**.

Ejemplo 2: Feeds (Contenido en múltiples formatos)

- **Problema**: Necesitamos manejar diferentes formatos de contenido, y pueden agregarse nuevos.
- **Indicador clave**: Necesitamos una solución extensible para crear objetos sin acoplarlos al código principal.
- **Patrón**: **Factory Method**.

Ejemplo 3: Biblioteca privada (Acceso a varios subsistemas)

- **Problema**: Hay varios subsistemas con los que necesitamos interactuar.
- **Indicador clave**: Queremos ocultar la complejidad de estos subsistemas.
- **Patrón**: **Facade**.

Ejemplo 4: Facebook (Cargar imágenes de amigos remotamente)

- **Problema**: Queremos evitar cargar imágenes innecesarias inicialmente.
- **Indicador clave**: Queremos control de acceso y carga diferida.
- **Patrón**: **Proxy**.

Ejemplo 5: Juego (Máquina de estados del personaje)

- **Problema**: El comportamiento del personaje cambia según su estado.
- **Indicador clave**: Cambios de comportamiento basados en estados internos.
- **Patrón**: **State**.

Ejemplo 6: Notificaciones de Torneo

- **Problema**: Los jugadores necesitan ser notificados automáticamente sobre eventos.
- **Indicador clave**: Notificaciones automáticas a múltiples entidades.
- **Patrón**: **Observer**.

5. Practica con más escenarios

Un buen ejercicio es escribir escenarios simples y aplicar esta metodología. Por ejemplo:

1. **¿Qué necesitas?**
    - Crear objetos → **Factory Method**.
    - Única instancia → **Singleton**.
    - Simplificar accesos → **Facade**.
    - Controlar el acceso o cargar algo bajo demanda → **Proxy**.
    - Cambiar comportamiento dinámicamente → **State**.
    - Notificar múltiples objetos → **Observer**.
2. **Por qué lo necesitas?** Esto te ayudará a justificar la elección del patrón en un examen.

Si tienes más ejemplos o escenarios, ¡puedo ayudarte a practicarlos! 😊

EXPLICADO DE OTRA FORMA

Para identificar qué patrón de diseño aplicar en un escenario dado, es importante comprender el propósito de cada patrón y reconocer las señales o "pistas" que nos proporciona la descripción del problema. A continuación, te detallo cada patrón y cómo identificarlo en diferentes situaciones:

1. Singleton

Propósito: Garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.

Indicadores clave:

- Necesidad de una única instancia.
- Control centralizado de recursos.
- Evitar la creación de múltiples objetos que representan lo mismo.
- Frases como "única conexión", "evitar múltiples ventanas", "solo debe existir uno".

Ejemplo aplicado:

- **Winforms (Ejemplo 1):** Se abren múltiples ventanas de la calculadora. Para evitar esto, se necesita una única instancia de la calculadora. **Patrón:** Singleton.

2. Observer (Observador)

Propósito: Definir una dependencia uno a muchos entre objetos, de manera que cuando uno cambie su estado, todos sus dependientes sean notificados y actualizados automáticamente.

Indicadores clave:

- Necesidad de notificar a múltiples objetos sobre cambios.
- Actualizaciones automáticas.
- Conceptos de "suscripción", "notificación", "eventos".
- Frases como "se les notifique cuando...", "recibir novedades", "avisos a los clientes".

Ejemplo aplicado:

- **Torneo de jugadores (Ejemplo de examen 1):** Los jugadores deben ser notificados sobre inscripciones, fixtures, cambios y resultados. **Patrón:** Observer.
- **Nuevos videos cargados (Ejemplo de examen 2):** Notificar a los clientes sobre nuevos videos de su preferencia. **Patrón:** Observer.

3. State (Estado)

Propósito: Permitir que un objeto altere su comportamiento cuando su estado interno cambia. El objeto parecerá cambiar su clase.

Indicadores clave:

- Comportamiento que cambia en tiempo de ejecución basado en el estado.
- Máquina de estados.
- Transiciones entre diferentes estados.
- Frases como "puede moverse, atacar y detenerse", "diferentes estados".

Ejemplo aplicado:

- **Juego (Ejemplo 5):** El personaje cambia de comportamiento según su estado (moverse, atacar, detenerse). **Patrón:** State.

4. Facade (Fachada)

Propósito: Proporcionar una interfaz unificada y simplificada a un conjunto de interfaces en un subsistema, haciendo que el subsistema sea más fácil de usar.

Indicadores clave:

- Simplificar interacciones complejas.
- Ocultar la complejidad de subsistemas.
- Proporcionar una interfaz más simple al cliente.
- Frases como "facilitar el acceso", "simplificar", "unificar interfaces", "múltiples subsistemas".

Ejemplo aplicado:

- **Biblioteca privada (Ejemplo 3):** Acceso simplificado a los subsistemas de alumnado y administrativo para verificar condiciones. **Patrón:** Facade.

5. Proxy (Representante/Sustituto)

Propósito: Proporcionar un sustituto o representante de otro objeto para controlar el acceso a este.

Indicadores clave:

- Control de acceso a objetos.
- Carga diferida o "perezosa" de recursos pesados.
- Actuar como intermediario.
- Frases como "evitar la carga inicial", "cargar imágenes de forma remota", "existen muchos objetos a cargar".

Ejemplo aplicado:

- **Facebook (Ejemplo 4):** Evitar la carga inicial de imágenes de 5000 amigos. Las imágenes se cargan bajo demanda. **Patrón:** Proxy.

6. Factory Method (Método de Fábrica)

Propósito: Definir una interfaz para crear un objeto, pero dejar que las subclases decidan qué clase instanciar. Permite una clase delegar la instanciación a subclases.

Indicadores clave:

- Necesidad de crear objetos sin especificar la clase exacta.
- Delegación de la creación de objetos.
- Facilitar la extensión para soportar nuevas clases.
- Frases como "agregar nuevos tipos fácilmente", "crear objetos dinámicamente".

Ejemplo aplicado:

- **Feeds (Ejemplo 2):** Necesidad de soportar múltiples formatos de feeds (XML, JSON, HTML, etc.) y facilitar la adición de nuevos formatos. **Patrón:** Factory Method.

Consejos para Identificar el Patrón Correcto:

1. **Analiza los Requerimientos Clave:** Identifica las necesidades principales del escenario. ¿Se requiere notificación, control de instancias, cambio de comportamientos?
2. **Busca Palabras Clave en la Descripción:** Las frases y términos utilizados suelen dar pistas sobre el patrón adecuado.
3. **Considera el Propósito de Cada Patrón:** Relaciona las necesidades del escenario con el propósito del patrón.
4. **Piensa en las Consecuencias:** ¿Qué problemas se resolverán aplicando cierto patrón? ¿Evita la duplicación de instancias? ¿Simplifica el acceso a subsistemas?
5. **Practica con Ejemplos:** Repasa escenarios y asocia patrones, como en los ejemplos proporcionados.

Aplicación de los Patrones en los Ejemplos:

- **Ejemplo de examen 3 (Conexión única a la BD):**
  - **a.** Se requiere una única conexión a la base de datos. **Patrón:** Singleton.
  - **b.** Código que implementa el patrón Singleton para gestionar la conexión.

Al familiarizarte con el propósito y las señales de cada patrón, podrás identificar más fácilmente cuál aplicar en cada situación. Es útil practicar con distintos escenarios y, si es posible, implementar pequeños ejemplos de cada patrón para entender su funcionamiento en la práctica.