UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ciencias de la Ingeniería y Tecnología

Unidad Valle de las Palmas

Patrones de Software

Meta 3.2 Analizar la aplicación de cada patrón de diseño en casos prácticos.

José Humberto Moreno Mejía

SEPTIEMBRE 2024

1. Patrón Observador (Observer Pattern)

¿Dónde se utiliza?

Se utiliza en sistemas donde múltiples objetos deben estar sincronizados o notificados de cambios en otro objeto sin necesidad de acoplarse directamente, como en interfaces de usuario (UI) o sistemas de eventos.

¿Cómo implementarlo?

Se define un sujeto (subject) que mantiene una lista de observadores. Cada vez que el sujeto cambia su estado, notifica a todos los observadores mediante un método como notify().

Ventajas

Desacopla los objetos observadores del sujeto.
Facilita agregar nuevos observadores sin modificar el sujeto.

Desventajas

Puede generar problemas de rendimiento con muchos observadores.
No garantiza el orden de notificación.

2. Patrón Decorador (Decorator Pattern)

¿Dónde se utiliza?

Se utiliza para añadir funcionalidades adicionales a objetos de manera dinámica, como en componentes gráficos donde se agregan comportamientos sin modificar el objeto original.

¿Cómo implementarlo?

Se crea una clase decoradora que implementa la misma interfaz que el objeto decorado y añade el comportamiento adicional sobre este.

Ventajas

Añade responsabilidades de forma flexible sin modificar el código existente.
Respeta el principio de abierto/cerrado.

Desventajas

Puede llevar a un aumento de complejidad si hay muchos decoradores.

3. Patrón Fábrica (Factory Patterns)

¿Dónde se utiliza?

Se usa cuando la creación de objetos es compleja o varía según el contexto, como en sistemas donde se requieren diferentes tipos de objetos según parámetros.

¿Cómo implementarlo?

Se crea una clase que se encarga de la creación de instancias de varias clases derivadas de una misma jerarquía.

Ventajas

Centraliza la creación de objetos y facilita cambios.
Aumenta la flexibilidad y desacopla la creación del uso.

Desventajas

Añade complejidad si se abusa de su uso.

4. Patrón Singleton (Singleton Pattern)

¿Dónde se utiliza?

Se usa cuando solo debe existir una única instancia de una clase en todo el sistema, como en la gestión de recursos compartidos (conexiones a bases de datos, configuraciones globales).

¿Cómo implementarlo?

Se crea una clase con un constructor privado y un método estático que retorna la misma instancia siempre.

Ventajas

Controla el acceso a una única instancia.

Ahorra recursos al evitar la creación de múltiples instancias.

Desventajas

Dificulta las pruebas unitarias.

Puede ser problemático en sistemas multithread si no se implementa correctamente.

5. Patrón Comando (Command Pattern)

¿Dónde se utiliza?

Se utiliza cuando se quiere encapsular solicitudes o acciones como objetos, facilitando la parametrización de operaciones y la implementación de deshacer y rehacer, común en interfaces gráficas.

¿Cómo implementarlo?

Se crea una clase Command que encapsula una acción a ser ejecutada, con un método execute() que realiza la acción.

Ventajas

Permite implementar operaciones deshacer/rehacer.

Desacopla la lógica de la solicitud de la ejecución.

Desventajas

Puede llevar a una gran cantidad de clases comando si hay muchas acciones.

6. Patrón Adaptador (Adapter Pattern) y Fachada (Facade Pattern)

¿Dónde se utiliza?

Adaptador: Se utiliza cuando se necesita que dos interfaces incompatibles trabajen juntas, como al integrar sistemas o bibliotecas de terceros.

Fachada: Se utiliza para simplificar el acceso a un conjunto complejo de interfaces, como en sistemas grandes o frameworks.

¿Cómo implementarlo?

Adaptador: Se crea una clase que implementa la interfaz esperada por el cliente y traduce las llamadas a la clase que no tiene una interfaz compatible.

Fachada: Se implementa una clase con métodos simplificados que gestionan llamadas a subsistemas más complejos.

Ventajas

Adaptador: Permite integrar sistemas sin modificar el código original.

Fachada: Simplifica la interacción con subsistemas complejos.

Desventajas

Adaptador: Puede añadir sobrecarga y complejidad.

Fachada: Puede ocultar funcionalidades críticas si no se usa con cuidado.

7. Patrón Método Plantilla (Template Method Pattern)

¿Dónde se utiliza?

Se usa cuando es necesario definir la estructura general de un algoritmo, pero con algunos pasos específicos que se delegan a subclases. Es común en frameworks.

¿Cómo implementarlo?

Se define un método en una clase base que contiene la estructura del algoritmo, mientras que algunas partes se implementan en las subclases.

Ventajas

Facilita la reutilización de código común y personaliza comportamientos específicos.

Desventajas

Puede acoplar subclases a la estructura del algoritmo.

8. Patrón Iterador (Iterator Pattern) y Composición (Composite Pattern)

¿Dónde se utiliza?

Iterador: Se utiliza cuando se quiere recorrer una colección de elementos sin exponer su representación interna, como listas o arreglos.

Composición: Se usa para tratar de manera uniforme tanto objetos individuales como agregados de objetos, como estructuras de árbol.

¿Cómo implementarlo?

Iterador: Se crea una interfaz con métodos como next() y hasNext() para recorrer los elementos.

Composición: Se definen objetos compuestos que contienen otros objetos y permiten tratarlos de forma uniforme.

Ventajas

Iterador: Desacopla la lógica de recorrido de la estructura de datos.

Composición: Facilita la gestión de estructuras jerárquicas.

Desventajas

Iterador: Puede ser ineficiente con estructuras grandes.

Composición: Añade complejidad a la implementación de operaciones recursivas.

9. Patrón Estado (State Pattern)

¿Dónde se utiliza?

Se utiliza cuando un objeto cambia su comportamiento basado en su estado interno, como en máquinas de estado finito (FSM).

¿Cómo implementarlo?

Se encapsulan los comportamientos correspondientes a cada estado en clases separadas, y el objeto principal delega su comportamiento a la clase que representa el estado actual.

Ventajas

Facilita la adición de nuevos estados sin modificar el código existente.
Mejora la claridad cuando un objeto tiene múltiples comportamientos.

Desventajas

Puede aumentar el número de clases en el sistema.

10. Patrón Proxy (Proxy Pattern)

¿Dónde se utiliza?

Se utiliza cuando se desea controlar el acceso a un objeto, como en el caso de objetos remotos o cargados perezosamente.

¿Cómo implementarlo?

Se crea una clase proxy que implementa la misma interfaz que el objeto original y controla el acceso a él.

Ventajas

Controla el acceso a recursos o datos sensibles.
Permite optimizar el rendimiento al retrasar la carga de objetos hasta que sean necesarios.

Desventajas

Añade complejidad al sistema.
Puede degradar el rendimiento si no se implementa correctamente.