Patrones estructurales

Facade

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"
[Christopher Alexander].

Por: Brayan Estiven Aguirre Aristizábal – 20231020156
Santiago Zamudio Diaz – 20231020128
Daniel Josué Rozo Rodríguez – 20222020006

1. Introducción

En el diseño de software, los patrones estructurales son fundamentales para organizar objetos y clases en estructuras cohesivas y complejas. Estos patrones aseguran que los cambios en los requisitos de la aplicación no afecten las relaciones entre objetos existentes, centrándose en las interacciones y dependencias durante la ejecución del programa.
Uno de estos patrones es el Facade, que simplifica y unifica interfaces complejas en un sistema. Proporciona una interfaz única y fácil de usar que oculta la complejidad interna de múltiples subsistemas. Al encapsular estos subsistemas detrás de una fachada común, el patrón Facade mejora la claridad y la mantenibilidad del código, facilitando la integración de componentes y reduciendo la dependencia de detalles internos. En esta presentación, exploraremos cómo el patrón Facade optimiza la arquitectura de software para aplicaciones robustas y escalables.

2. Facade

Nombre del patrón:

Facade

Clasificación del patrón:

Estructural

Intención:

Proporcionar una interfaz unificada y simplificada a un conjunto de interfaces más complejas en un sistema.

También conocido por:

Wrapper, Fachada

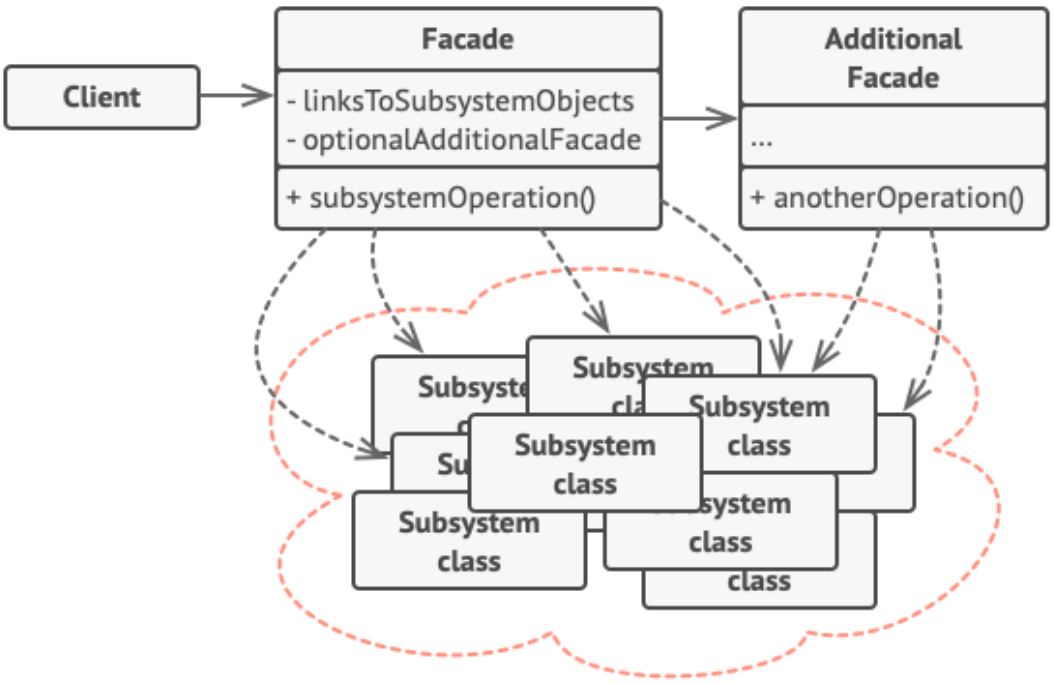
Motivación:

El patrón Facade permite encapsular la complejidad de un sistema detrás de una interfaz unificada y simplificada. Su objetivo principal es proporcionar a los clientes una manera fácil y coherente de interactuar con un conjunto de subsistemas complejos. En lugar de exponer las interfaces individuales y complicadas de cada subsistema, el Facade ofrece una única interfaz que oculta los detalles internos y facilita el uso del sistema en su conjunto. Esto mejora la claridad, la mantenibilidad y la facilidad de uso del sistema, haciendo que sea más fácil integrar y mantener componentes complejos dentro de una aplicación.

Aplicabilidad:

- Simplificar interfaces complejas de múltiples subsistemas.
- Encapsular la complejidad de subsistemas existentes detrás de una interfaz unificada.
- Reducir la dependencia de los clientes respecto a los detalles internos.
- Mejorar la modularidad al proporcionar puntos de entrada cohesivos.
- Facilitar la integración y migración de sistemas existentes con diferentes interfaces.

Estructura:



Participantes:

Facade: Proporciona un acceso práctico y unificado a una parte específica del Subsistema Complejo. Sabe cómo dirigir las solicitudes del cliente y manejar todos los detalles de implementación, como inicializar objetos en el orden correcto y suministrar datos en el formato adecuado. Esto simplifica el acceso a la funcionalidad del sistema, ocultando su complejidad interna y mejorando la modularidad del código.

Facade Adicional: Para mantener la fachada principal simple, se pueden crear fachadas adicionales. Estas fachadas adicionales actúan como capas separadas que pueden ser usadas por clientes u otras fachadas, mejorando la organización del código y facilitando la reutilización sin complicar la fachada principal.

Subsistema Complejo: Consiste en decenas de objetos diversos que trabajan entre sí dentro del sistema. Las clases del subsistema operan sin conocimiento de la fachada, realizando tareas complejas como inicializar objetos y procesar datos sin la necesidad de interacción directa con la interfaz de la fachada.

Cliente: Utiliza la fachada como punto de entrada para interactuar con el Subsistema Complejo en lugar de invocar directamente los objetos individuales del subsistema. Esto promueve una arquitectura más limpia y desacoplada, donde el cliente se beneficia de una interfaz simplificada proporcionada por la fachada.

Colaboraciones:

- El cliente trabaja con el sistema a través de la interfaz unificada proporcionada por la fachada. No necesita conocer los detalles internos del subsistema ni cómo están implementadas sus partes.
- Facade actúa como un punto de entrada único y simplificado para el cliente, coordinando las solicitudes y delegándolas a los objetos y clases adecuadas dentro del sistema.

Consecuencias:

- Proporciona una interfaz unificada y simplificada para interactuar con un sistema complejo. Esto oculta la complejidad interna del sistema y facilita su uso para el cliente.
- Reduce la dependencia del cliente respecto a las clases individuales del subsistema. El cliente no necesita conocer ni manejar directamente los detalles de implementación del subsistema.
- Promueve la modularidad al definir una capa de abstracción que separa el cliente del subsistema complejo. Esto facilita la mejora y mantenimiento independiente de cada parte del sistema.

- Centraliza el acceso a las funcionalidades del subsistema, lo que permite aplicar políticas de control de acceso, seguridad u optimización de manera más efectiva y cohesiva.
- Una fachada puede ser un objeto muy poderoso conectado a todas las clases de una aplicación.

Implementación:

- Explora la posibilidad de simplificar la interfaz proporcionada por un subsistema existente. Un buen enfoque es crear una interfaz que permita al código cliente operar de manera independiente de las múltiples clases del subsistema.
- Define e implementa esta interfaz en una nueva clase de fachada. La fachada deberá manejar las llamadas del código cliente hacia los componentes adecuados del subsistema. Además, la fachada puede encargarse de inicializar el subsistema y gestionar su ciclo de vida, a menos que esto ya esté manejado por el código cliente.
- Para optimizar el uso del patrón, establece que todo el código cliente interactúe exclusivamente a través de la fachada. Esto protege al cliente de cualquier cambio interno en el subsistema. Por ejemplo, al actualizar el subsistema a una nueva versión, solo será necesario ajustar el código de la fachada.
- Si la fachada crece demasiado, considera la posibilidad de dividir su funcionalidad y trasladar partes a una nueva clase de fachada refinada.

Código ejemplo:

Ejemplo de Facade en Java

```java
// Sistema de Reservas
public class SistemaReservas {
    public void reservarAsiento(String pelicula, int asiento) {
        System.out.println("Asiento " + asiento + " reservado para la película " + pelicula);
    }
}

// Sistema de Pagos
public class SistemaPagos {
    public void procesarPago(String tarjeta, double monto) {
        System.out.println("Pago de $" + monto + " procesado para la tarjeta " + tarjeta);
    }
}

// Sistema de Notificaciones
public class SistemaNotificaciones {
    public void enviarNotificacion(String mensaje) {
        System.out.println("Notificación enviada: " + mensaje);
    }
}

//Facade

public class CineFacade {
    private SistemaReservas reservas;
    private SistemaPagos pagos;
    private SistemaNotificaciones notificaciones;

    public CineFacade() {
        this.reservas = new SistemaReservas();
        this.pagos = new SistemaPagos();
        this.notificaciones = new SistemaNotificaciones();
    }

    public void comprarEntrada(String pelicula, int asiento, String tarjeta, double monto) {
        reservas.reservarAsiento(pelicula, asiento);
        pagos.procesarPago(tarjeta, monto);
        notificaciones.enviarNotificacion("Entrada comprada para " + pelicula + ", asiento " + asiento);
    }
}

//Cliente

public class Cliente {
    public static void main(String[] args) {
        CineFacade cine = new CineFacade();
        cine.comprarEntrada("Avengers", 42, "1234-5678-9101-1121", 10.5);
    }
}
```

Usos Conocidos:

- El patrón Facade es comúnmente utilizado en aplicaciones y proyectos que involucran sistemas complejos o heredados, integración de múltiples sistemas, APIs externas, bibliotecas o frameworks extensos, y en general, en situaciones donde se necesita simplificar el acceso y la interacción con funcionalidades complejas para mejorar la mantenibilidad y la claridad del código.

Relaciones con otros patrones:

- Facade redefine la interfaz de objetos existentes, mientras que Adapter adapta una interfaz existente para hacerla utilizable. Adapter típicamente envuelve un solo objeto, mientras que Facade maneja todo un subsistema de objetos.
- Abstract Factory puede ser una alternativa a Facade cuando se desea ocultar la creación de objetos del subsistema desde el código cliente.
- Flyweight demuestra cómo crear muchos objetos pequeños, mientras que Facade muestra cómo encapsular un objeto único que representa un subsistema completo.
- Facade y Mediator comparten objetivos similares: ambos organizan la colaboración entre varias clases altamente acopladas.
- Facade proporciona una interfaz simplificada a un subsistema de objetos sin introducir nueva funcionalidad. El subsistema no está consciente de la fachada y sus objetos pueden comunicarse directamente entre sí. Mediator centraliza la comunicación entre componentes del sistema. Los componentes solo conocen al mediador y no se comunican directamente.
- Frecuentemente, una fachada puede evolucionar hacia un Singleton, ya que un único objeto fachada suele ser suficiente.
- Facade y Proxy comparten similitudes en el sentido de que ambos pueden almacenar temporalmente una entidad compleja e inicializarla según sea necesario. Sin embargo, a diferencia de Facade, Proxy tiene la misma interfaz que su objeto de servicio, lo que permite que sean intercambiables.

Referencias Bibliográficas

[1] *Facade*. (2014). Refactoring.guru. https://refactoring.guru/es/design-patterns/facade

[2] de. (2021, February 22). *Facade pattern: interfaz unificada para proyectos de software*. IONOS Digital Guide; IONOS. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/facade-pattern/