**Universidad de las Fuerzas Armadas ESPE**

**Departamento:** Ciencias de la Computación

**Carrera:** Ingeniería de Software

**Taller académico Nª: 3**

**1. Información General**

* **Asignatura:** Análisis y Diseño de Software
* **Apellidos y nombres de los estudiantes:**
  + Chavez Oscullo Klever Enrique
  + Guacan Rivera Alexander David
  + Trejo Duque Alex Fernando
* **NRC:** 23305
* **Fecha de realización: 11/06/2025**

**2. Objetivo del Taller y Desarrollo**

**Objetivo del Taller:**  
Investigar y analizar el patrón de diseño *Facade* a través de diversas fuentes bibliográficas y prácticas, con el propósito de comprender su definición, funcionamiento y utilidad en la industria del software, así como demostrar su aplicación mediante un ejemplo implementado en un entorno de programación orientado a objetos.

**Desarrollo:**  
El presente taller se centra en la investigación del patrón de diseño Facade, un enfoque estructural que, según Freeman y Robson (2004), proporciona una interfaz unificada para simplificar el uso de subsistemas complejos. Este patrón, ampliamente reconocido en la ingeniería de software, permite reducir el acoplamiento entre el cliente y los componentes internos, facilitando así la mantenibilidad y escalabilidad de los sistemas. Por ello, el análisis de su definición, aplicaciones y ejemplo práctico resulta esencial para comprender su relevancia.

**Definición del patrón de diseño Facade**

El patrón Facade, según Freeman y Robson (2004), consiste en una clase que ofrece una interfaz de alto nivel para un conjunto de interfaces dentro de un subsistema complejo. Su principal función es ocultar la complejidad interna, proporcionando al cliente un punto de acceso simplificado sin eliminar la posibilidad de interactuar directamente con los componentes subyacentes si es necesario. Este enfoque promueve la claridad y la facilidad de uso en el diseño de software.

**Aplicación en la industria**

**Desarrollo de aplicaciones empresariales:**

**Contexto:** En sistemas donde múltiples módulos se tienen que comunicar como contabilidad, inventarios, recursos humanos tienen interfaces complejas, una Fachada puede unificar las operaciones comunes.

**Ejemplo:** Una Fachada para un sistema de gestión de pedidos podría ofrecer métodos como crearPedido(), cancelarPedido() o consultarEstado(), ocultando la interacción con subsistemas como inventario, facturación y logística.

**Beneficio:** Los desarrolladores de la interfaz de usuario o aplicaciones externas como apps móviles pueden interactuar con el sistema sin conocer los detalles de cada módulo.

**Desarrollo de APIs y microservicios:**

**Contexto:** En arquitecturas de microservicios, donde múltiples servicios tienen que ejecutarse para llegar al objetivo como autenticación, pagos, notificaciones trabajan juntos, una Fachada puede actuar como una API Gateway.

**Ejemplo:** En una plataforma de comercio electrónico, una Fachada podría ofrecer un método procesarCompra(), que coordina la autenticación del usuario, la verificación de inventario, el procesamiento del pago y el envío de notificaciones, todo en una sola llamada.

**Beneficio:** Simplifica la integración para clientes externos y reduce la cantidad de llamadas a servicios individuales.

**Industria de videojuegos:**

**Contexto:** Los motores de videojuegos como Unity o Unreal Engine suelen usar Fachadas para simplificar el acceso a sistemas complejos como renderizado, física o gestión de audio.

**Ejemplo:** Una Fachada en un motor gráfico podría proporcionar un método dibujarEscena(), que coordina el renderizado de modelos 3D, la aplicación de texturas y la gestión de luces, sin que el desarrollador del juego necesite interactuar con cada subsistema.

**Beneficio:** Reduce la curva de aprendizaje y acelera el desarrollo de juegos.

**Aplicaciones financieras:**

**Contexto:** En sistemas bancarios, donde se integran módulos de transferencias, validación de cuentas y registro de transacciones, una Fachada simplifica operaciones complejas.

**Ejemplo:** Un método transferirDinero() en una Fachada podría encargarse de validar la cuenta de origen, verificar fondos, realizar la transferencia y registrar la transacción en un solo paso.

**Beneficio:** Mejora la seguridad y reduce errores al centralizar la lógica.

Ventajas del patrón Fachada en la industria

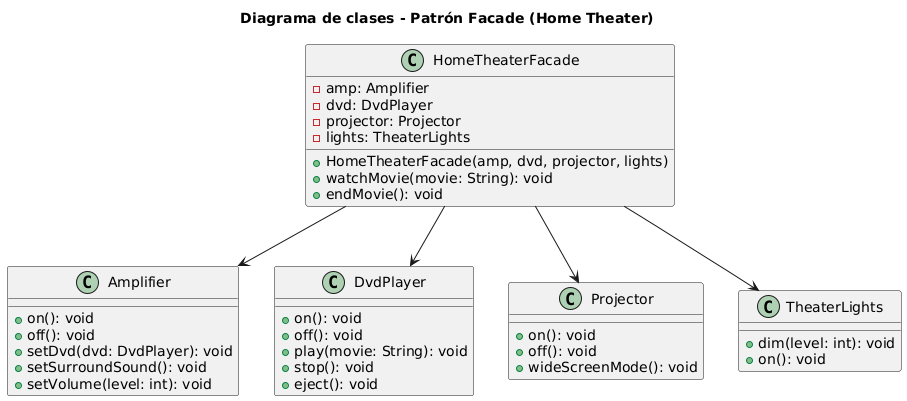
Reducción de acoplamiento: Los clientes solo interactúan con la Fachada, no con los subsistemas, lo que facilita cambios internos sin afectar a los usuarios.

**Simplificación:** Reduce la complejidad para los desarrolladores, especialmente en sistemas grandes.

**Mantenimiento:** Centraliza la lógica de coordinación, haciendo que el código sea más fácil de mantener.

Escalabilidad: Facilita la integración de nuevos subsistemas sin alterar la interfaz pública.

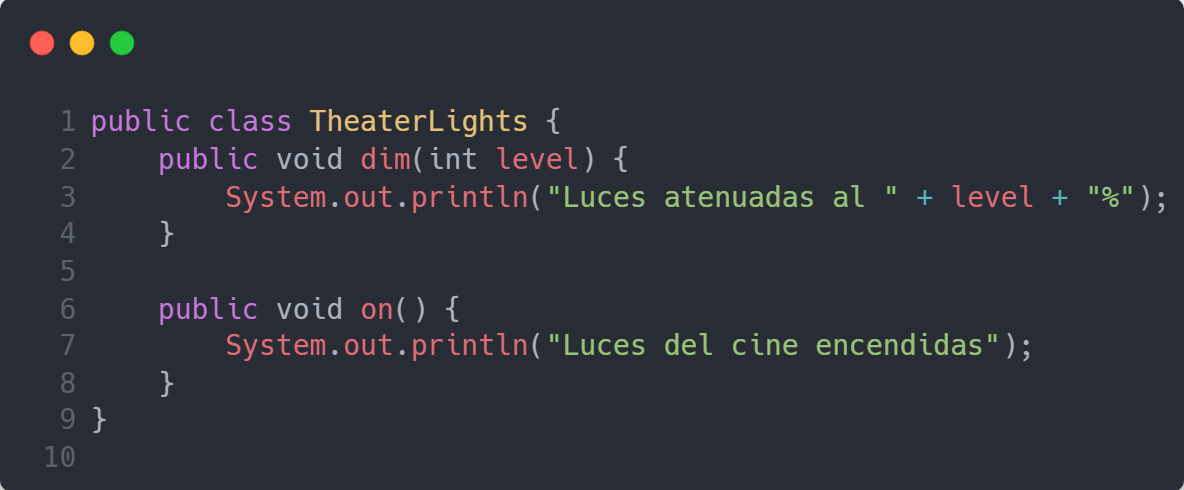
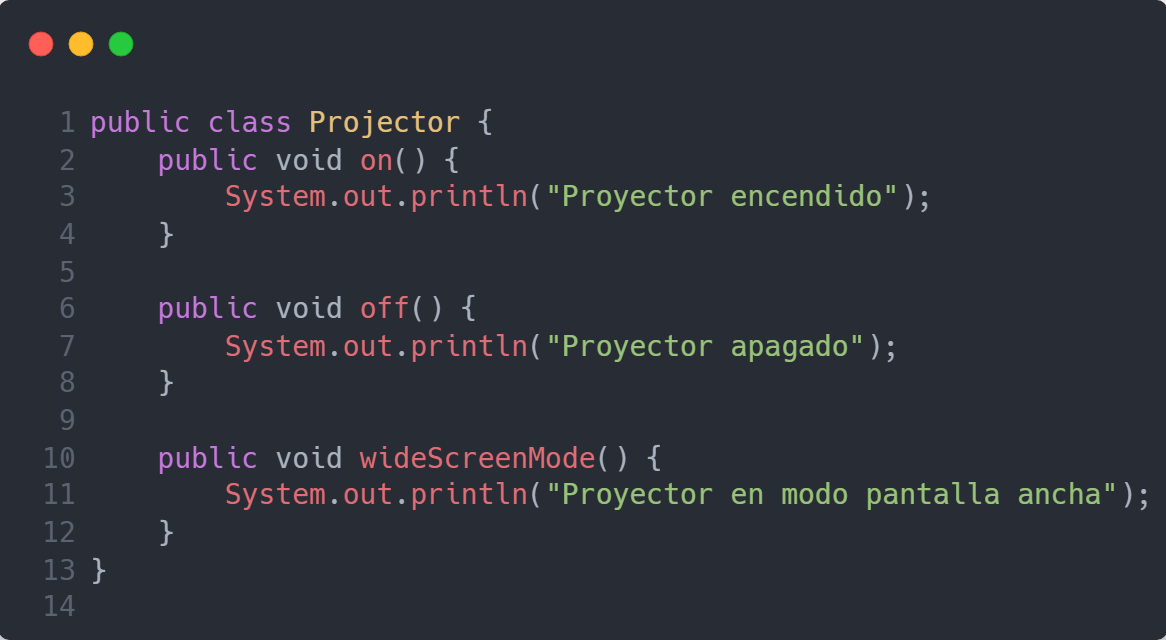
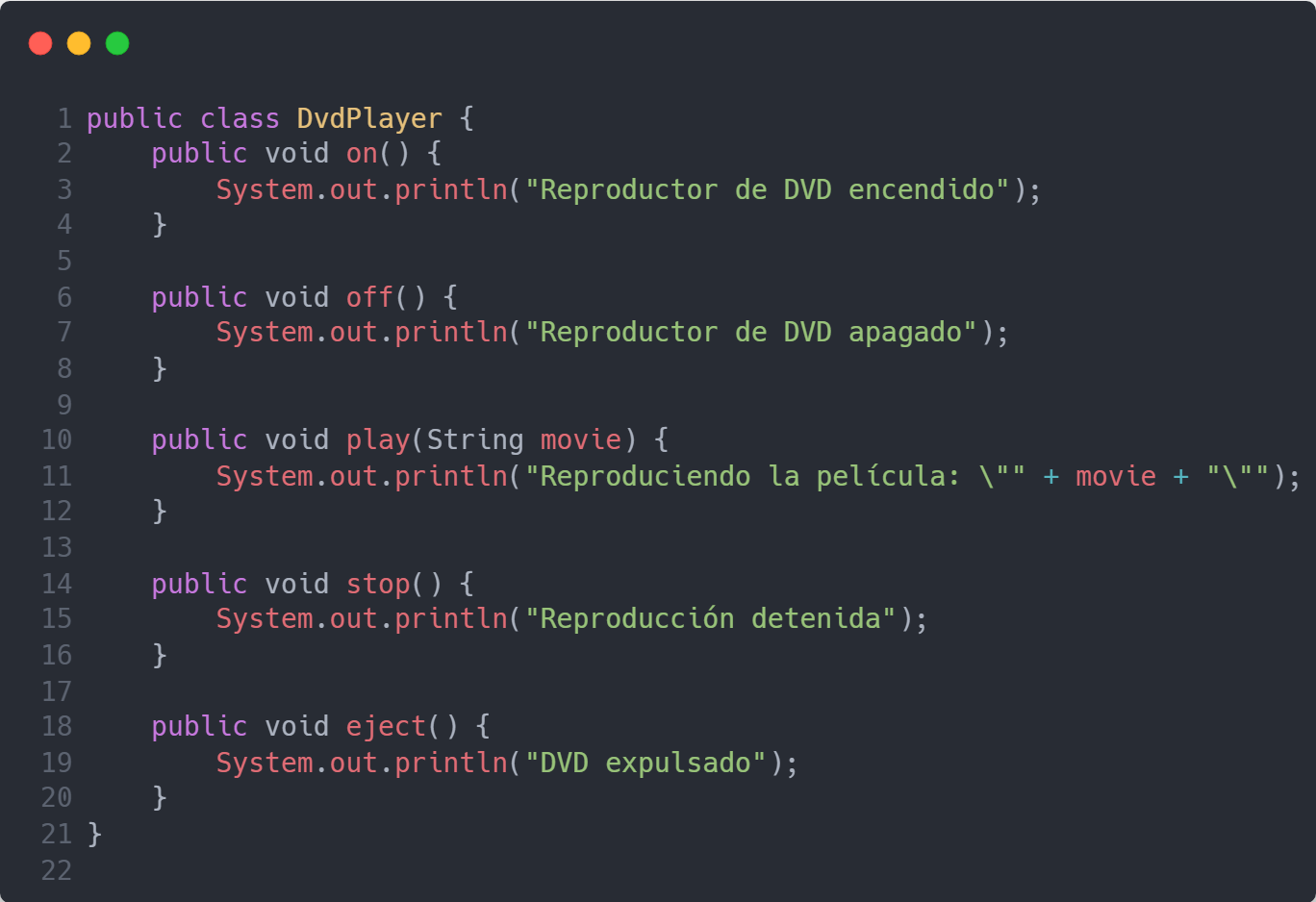
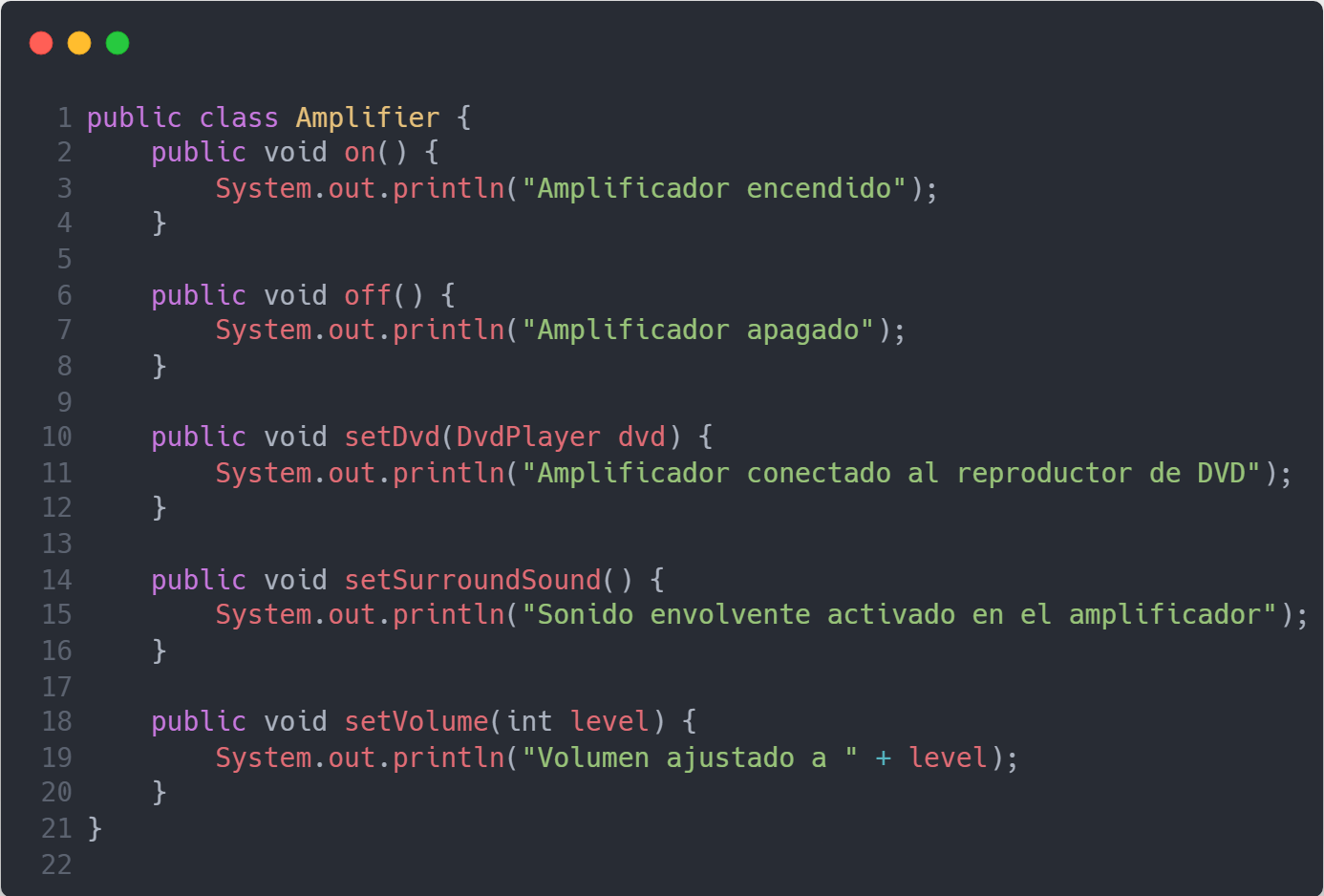
**Ejemplo con código fuente IDE OO**

Este ejemplo implementa el patrón de diseño Facade para simplificar la interacción con un sistema de entretenimiento en casa compuesto por múltiples componentes, como el amplificador, el reproductor de DVD, el proyector y las luces del teatro.

Componentes del sistema (subsistemas)

Cada componente del sistema tiene su propia lógica y métodos, por ejemplo:

* Amplifier: controla el sonido.
* DvdPlayer: maneja la reproducción de la película.
* Projector: se encarga de la imagen.
* TheaterLights: controla la iluminación del ambiente.



Estos objetos tienen métodos individuales como *on(), off(), play(), dim()*, etc., que normalmente deberían llamarse por separado para iniciar la experiencia de ver una película.

Rol de la fachada (HomeTheaterFacade)

La clase HomeTheaterFacade actúa como una interfaz de alto nivel que coordina todos los subsistemas. Ofrece métodos simples como:

* watchMovie(String movie): Enciende los dispositivos, atenúa las luces, configura el proyector y empieza la película.
* endMovie(): Apaga todo y detiene la reproducción.

De esta forma, el cliente no necesita interactuar con cada componente de forma individual ni preocuparse por el orden de las acciones.

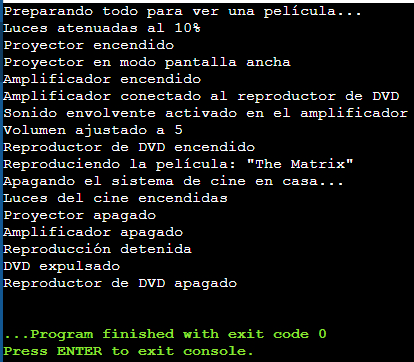


Cliente (HomeTheaterTestDrive)

La clase HomeTheaterTestDrive representa el cliente que usa el sistema. Este simplemente crea una instancia de HomeTheaterFacade y llama a watchMovie("Matrix") y luego a endMovie().



Resultado



Beneficio clave

Este diseño demuestra cómo el patrón Facade:

* Oculta la complejidad interna del sistema.
* Reduce el acoplamiento entre el cliente y los subsistemas.
* Simplifica el uso del sistema proporcionando una interfaz clara y directa.

**3. Referencias (Norma APA 7.0)**

* Freeman, E., & Robson, E. (2004). Head First Design Patterns. O'Reilly Media.