**Taller 5**

**Integrantes:**

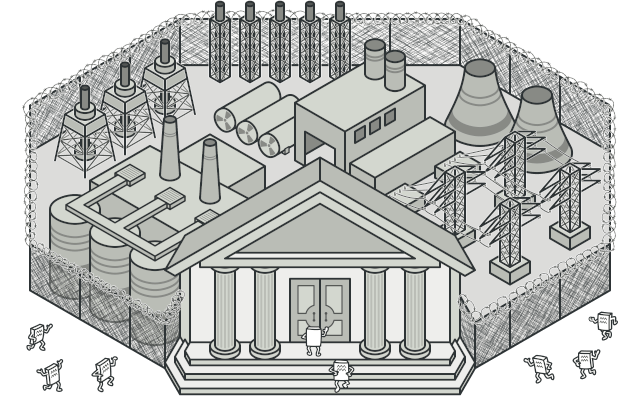
Nicolás Díaz Montaña - 202021006

Carlos Medina - 202112046

Juan Diego Cruz - 202015076

Eliana Palacio – 202112428

**PATTERN FACADE**



**Información General del repositorio:**

URL: <https://github.com/piyush6348/Design-Patterns.git>

El repositorio creado por el usuario piyush6348, tiene el objetivo de enseñar diferentes patrones de diseño utilizando JAVA. Para esto se tienen varias carpetas, que están divididas en tres tipos de diseño, donde se explica los patrones de un manera sencilla y, hasta cierto punto, coloquial de como implementarlos. Esta explicación viene acompañada de un código ejemplo, el cual el usuario puede editar para empezar a familiarizarse con el patrón.

**Información y estructura del fragmento del proyecto donde aparece el patrón:**

Es un simple y sencillo código de prueba que toma de base el ejemplo de utilizar el teléfono como el mediador de todos los componentes de un teatro en casa. Estos componentes/subsistemas son el amplificador, reproductor de discos, reproductor de DVD, máquina de palomitas, proyector, pantalla, luces del teatro y un tuner. Como es un código de prueba, lo único que hace es imprimir el estado de cada componente durante el proceso, de comienzo a fin, en una sesión de películas.

**Información general sobre el patrón**

La intención del patrón facade es proveer una interfaz unificada a partir de un conjunto de interfaces de un subsistema, facade define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar, estructurar un sistema en subsistemas ayuda a reducir la complejidad, es un diseño común cuando el objetivo es minimizar la comunicación y dependencia entre subsistemas, una manera de conseguir este objetivo es introducir un objeto facade que provee una interfaz simplificada a las funciones generales del subsistema.

¿Cuándo usar el patrón facade?, cuando usted quiere proveer una interfaz simple a un subsistema complejo, los subsistemas usualmente se hacen más complejos con el tiempo, la mayoría de los patrones cuando son aplicados resultan en más y pequeñas clases, esto hace que el subsistema sea más reusable y más fácil de personalizar, pero se hace más difícil para clientes que no necesitan personalizarlo, por esto facade puede proveer una vista por defecto del subsistema suficientemente buena para la mayoría de los clientes de modo que solo los clientes que necesitan mayor personalización necesitarán mirar facade en profundidad.

Existen muchas dependencia entre los clientes y la implementación de las clases de una abstracción, introducir facade ayuda a desacoplar el subsistema de clientes y de otros subsistemas promoviendo la independencia y la portabilidad, si usted construir capas de su subsistema use facade para definir cada nivel del subsistema, si los subsistemas son dependientes se puede simplificar la independencia entre ellos haciendo que se comunique solamente a través de sus interfaces facade.

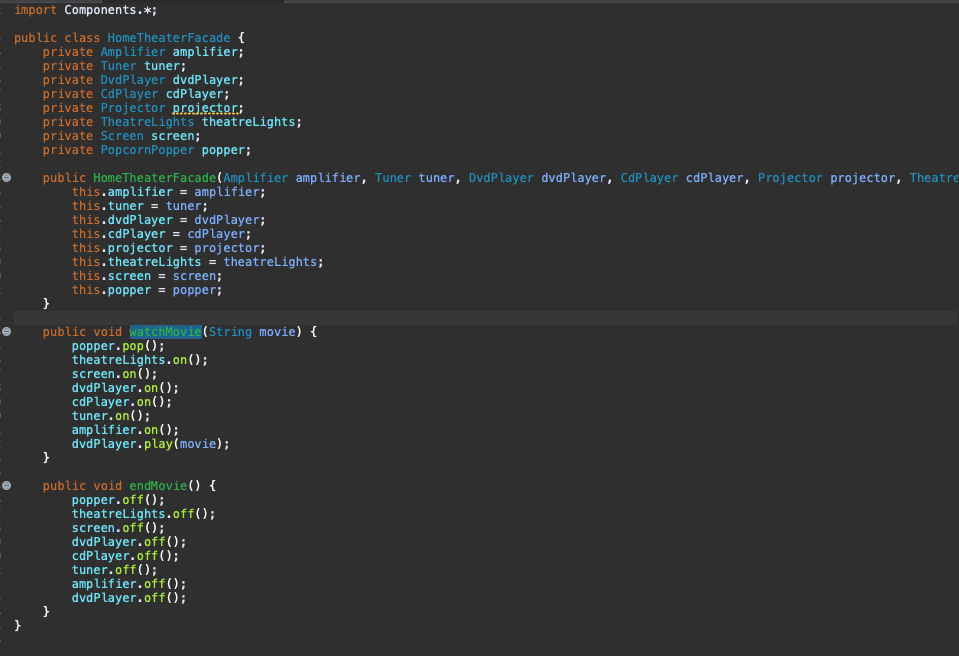
Beneficios, protege a los clientes de los componentes del subsistema reduciendo el número de objetos que el cliente debe manejar haciendo el uso del subsistema más fácil, promueve el acoplamiento débil entre el subsistema y los clientes, a menudo los componentes de un sistema están fuertemente acoplados, un acoplamiento débil permite modificar los componentes de un subsistema sin afectar sus clientes, facade ayuda las capas del sistema y la dependencia entre objetos eliminando dependencia complejas o circulares, esto no previene el uso de clases de subsistemas si es necesario, entonces usted debe elegir entre simplicidad de uso y generalidad.

Implementación, reducción del acoplamiento cliente subsistema, el acoplamiento entre cliente y subsistema se puede reducir incluso más profundamente haciendo de facade una clase abstracta con subclases concretas para diferentes implementaciones del subsistema, entonces los clientes pueden comunicarse con el subsistema a través de la interfaz de la clase abstracta facade, una alternativa a realizar subclases es configurar un objeto facade con diferentes objetos del subsistema, para personalizar facade simplemente reemplaza uno o más objetos del subsistema. Subsistema de clase publico vs privado, un subsistema es análogo a una clase ya que los dos tienen interfaces y los dos encapsulan algo, una clase encapsula estados y operaciones mientras que un subsistema encapsula clases y así como es útil pensar en interfaces públicas y privadas de una clase podemos pensar en interfaces públicas y privadas de un subsistema.

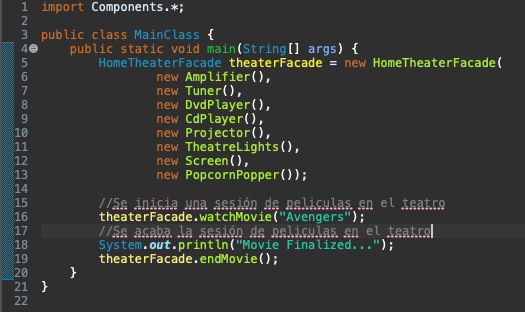
La interfaz pública de un subsistema consiste en las clases que todos los clientes pueden acceder, la interfaz privada solo es usada por el subsistema, la clase facade es parte de una interfaz privada por supuesto, pero no es la única.

**Información del patrón aplicado al proyecto:**

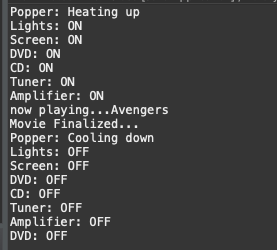
En el proyecto se utiliza una clase llamada HomeTheatherFacade, que es donde se implementa el patrón. En esta clase se hace llamado a los componentes para que realicen cierta funcionalidad, que este caso solo son prints, dependiendo de si el usuario quiere empezar o terminar una película.



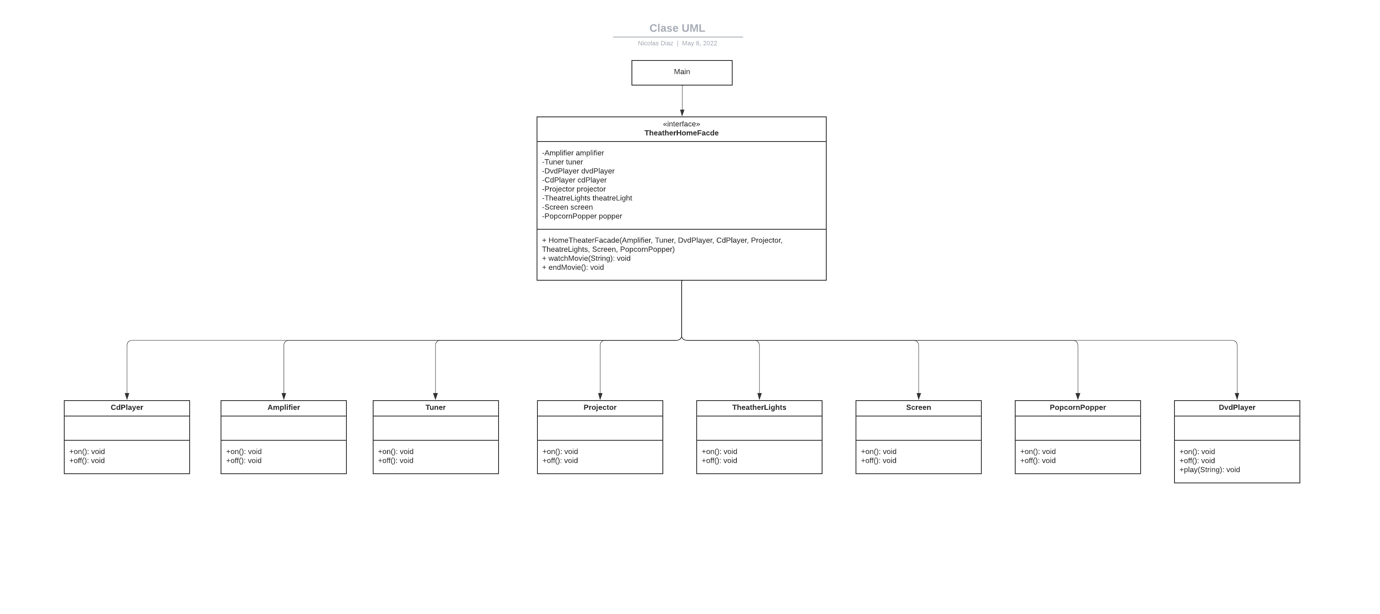
Esta clase se crea y se llama directamente desde el main. Este objeto responde al comando del usuario y se comunica con los demás componentes utilizando una línea de código, sin la necesidad de llamar a los objetos uno por uno.



El resultado del programa es el siguiente:



UML del código:



**¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto?**

Como bien se mencionó antes, el patrón Facade se caracteriza por ocultar la complejidad de un conjunto de subsistemas y dejar una interfaz de alto nivel, lo cual facilita la interpretación e interacción de la información con el usuario. En este proyecto, la utilización del patrón Facade es acertada ya que va acorde con el objetivo del sistema, el cual consiste en mostrar de forma simplificada el estado de todos los componentes de un teatro en casa al iniciar y al terminar de ver una película.

**¿Qué ventajas tiene?**

* Proporciona una interface de alto nivel que oculta la complejidad de interactuar con varios sistemas para realizar una operación.
* Debido al bajo grado de interdependencia de los componentes individuales, los cambios (modificaciones, mantenimiento) son convenientes y posibles en cualquier momento.
* Gracias al acoplamiento suelto, un subsistema es más fácil de expandir.

**¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?**

Generalmente, debido a su papel central, la **implementación** de una fachada es una **tarea tediosa y complicada**, especialmente si tiene que ser insertada en un código existente. En este caso en particular, la implementación de una interfaz de la fachada requirió una **etapa adicional indirecta**, que a su vez aumentó el tiempo de computación para las llamadas de método y función, el acceso a la memoria, etc.

**¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**

Mientras que Facade define una interfaz simplificada a un subsistema de objetos, pero no introduce ninguna nueva funcionalidad. Por su parte, Mediator centraliza la comunicación entre componentes del sistema, los componentes conocen únicamente el objeto mediador y no se comunican directamente. Para este caso en particular la implementación del patrón Mediator resolvería el objetivo del sistema, el cual consiste en mostrar de forma simplificada el estado de todos los componentes de un teatro en casa al iniciar y al terminar de ver una película.