Estudio del patrón de diseño Facade para un proyecto de GitHub

Nicolás Lara Gómez - 202012455

Mayo 2023

1. Patrón Facade

El patrón a analizar en este documento es el patrón Facade (patrón estructural). Este consiste en proveer una interfaz unificada a un set de interfaces en un subsistema. Facade define una interfaz de alto nivel que hace que los subsistemas sean más sencillos de emplear. Este patrón se usa principalmente para reducir la complejidad mediante la implementación de una única interfaz sencilla (esta es la que se denomina cono facade) que le permita al usuario acceder a las subclases de manera única sin tener que conocer al completo las funcionalidades de bajo nivel de estas. La idea primordial de este patrón es introducir un facade para desacoplar el subsistema de los clientes y otros subsistemas, promoviendo así la independencia del subsistema y portabilidad.

2. Proyecto escogido

El trabajo que fue escogido para estudiar el patrón de diseño Facode fue el proyecto denominado "design-pattern-facade"subido a un repositorio GitHub por el usuario BrijeshSaxena el 13 de octubre de 2020. Se puede acceder a este mediante el siguiente enlace: `https://github.com/BrijeshSaxena/design-pattern-facade`.

2.1. Retos de diseño que enfrenta

Entre los retos a los cuales se enfrenta se puede mencionar lo siguiente:

- Bajar el acoplamiento mediante la implementación de varias subclases que están conectadas a una única clase que interactúa con el main del proyecto
- Aumentar la cohesión del código. Al estar más subdividido todo, es más fácil de entender y de acceder a la información.
- Proporcionar una interfaz simplificada para un subsistema complejo (de varios elementos en este caso).

2.2. Uso

En este proyecto, se ha planteado el objetivo de crear una simulación sencilla que refleje los pasos y actividades que podrían llevarse a cabo durante un plan familiar para ver películas y disfrutar de una comida. La idea es proporcionar una representación virtual de un posible fin de semana en familia, con todas las recomendaciones y sugerencias necesarias para ver la pelicula que se desee y consumir lo que sea de su agrado.

Para lograr esto, se ha desarrollado un código con una estructura modular y flexible. A través de la clase principal que contiene el método main, se permite al usuario ingresar una serie de parámetros en formato de cadena de texto. Estos parámetros son procesados y, como resultado, se genera un protocolo detallado que se imprime en la consola.

El programa tiene en cuenta una amplia gama de aspectos relacionados con la planificación de un fin de semana familiar. Esto incluye la selección de películas adecuadas para todos los miembros de la familia, la

preparación de una variedad de alimentos y la creación de un ambiente acogedor y relajado para disfrutar del tiempo juntos.

En la figura 1, se puede apreciar un ejemplo de lo que el programa retorna. Esta representación visual proporciona una visión general del protocolo propuesto, mostrando los pasos sugeridos y las recomendaciones específicas para cada actividad. Es importante tener en cuenta que los detalles y las opciones específicas pueden variar según las preferencias y circunstancias de cada familia, pero la simulación busca brindar una guía general que se puede adaptar y personalizar según las necesidades individuales.

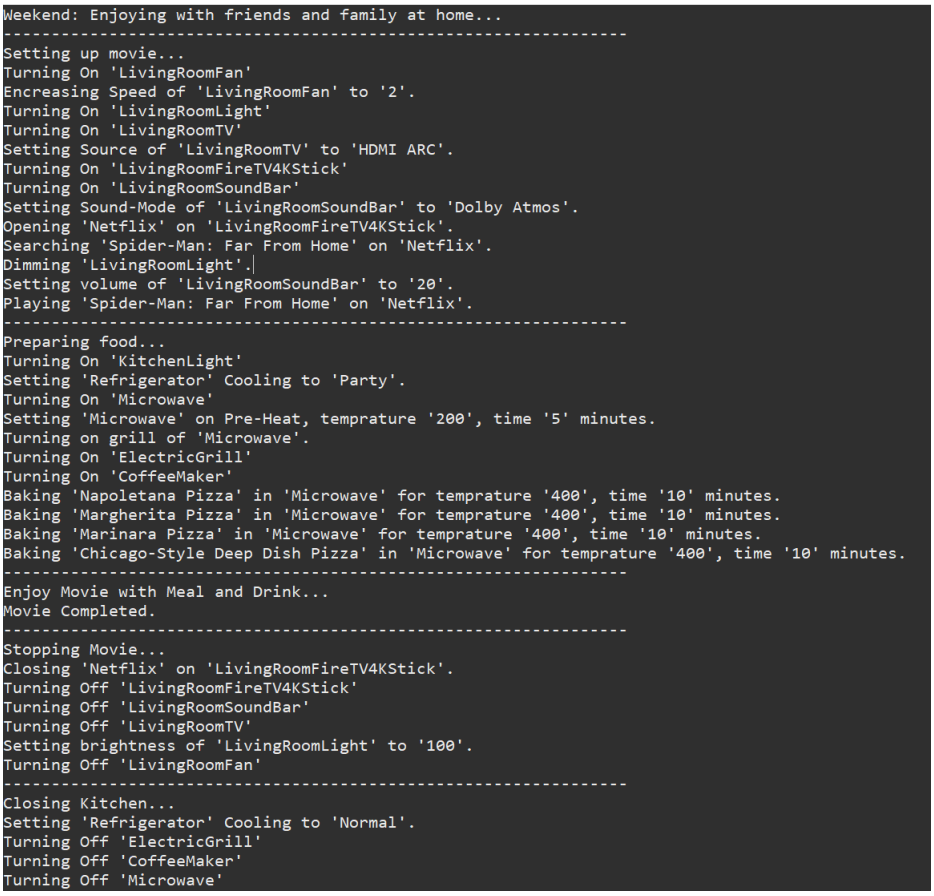

Figura 1: Ejemplo de la respuesta que puede dar el proyecto escogido para unos parámetros en concreto brindados.

2.3. Estructura general del diseño

El programa consiste en en varias clases cuyo nombres y funcionalidades van de la mano con los elementos que pueden estar presentes al momento de determinar el protocolo para compartir un fin de semana en familia. En estos se pueden apreciar los de la cocina (CoffeeMaker, ElectricGrill, KitchenLight, Microwave, Refrigerator) y los de la sala de estar (Appliance, Fan, Light, SoundBar, TV), siendo estas últimas superclases que extienden a unas clases más concretas (LivingRoomFan, LivingRoomTV, etc.) empleadas al momento de imprimir la respuesta en consola. Todas estas clases mencionadas están conectadas a una clase unificada (nuestra facade en este caso) llamada "HomeFacade"que hace las funcionalidades de bajar el acoplamiento y facilitar el uso de los subsistemas ya mencionados. Esta facade se conecta a la clase Main que es la encargada de ejecutar el sistema e imprimir la información esperada. Esto se puede apreciar mucho más fácilmente en el diagrama los siguiente diagramas de alto y bajo nivel (que pueden ser apreciados en las figuras 2 y 3)

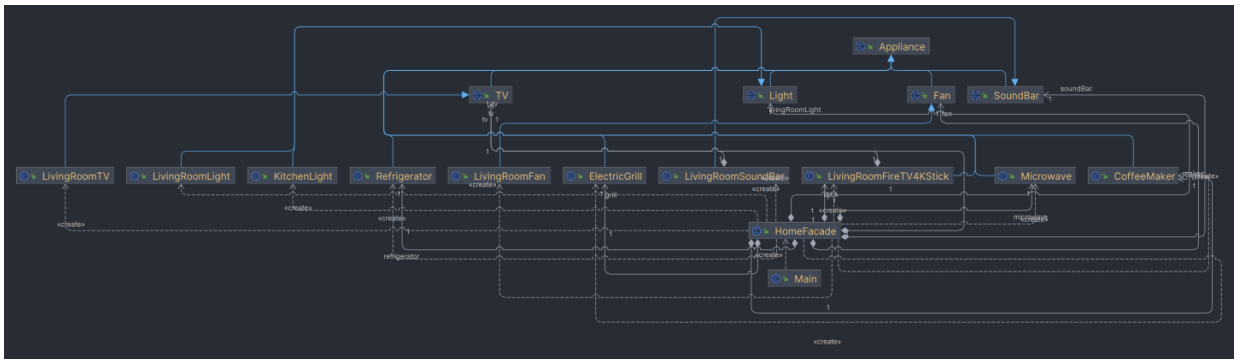

Figura 2: Diagrama UML de alto nivel que da muestra de la estructura general del diseño del proyecto escogido

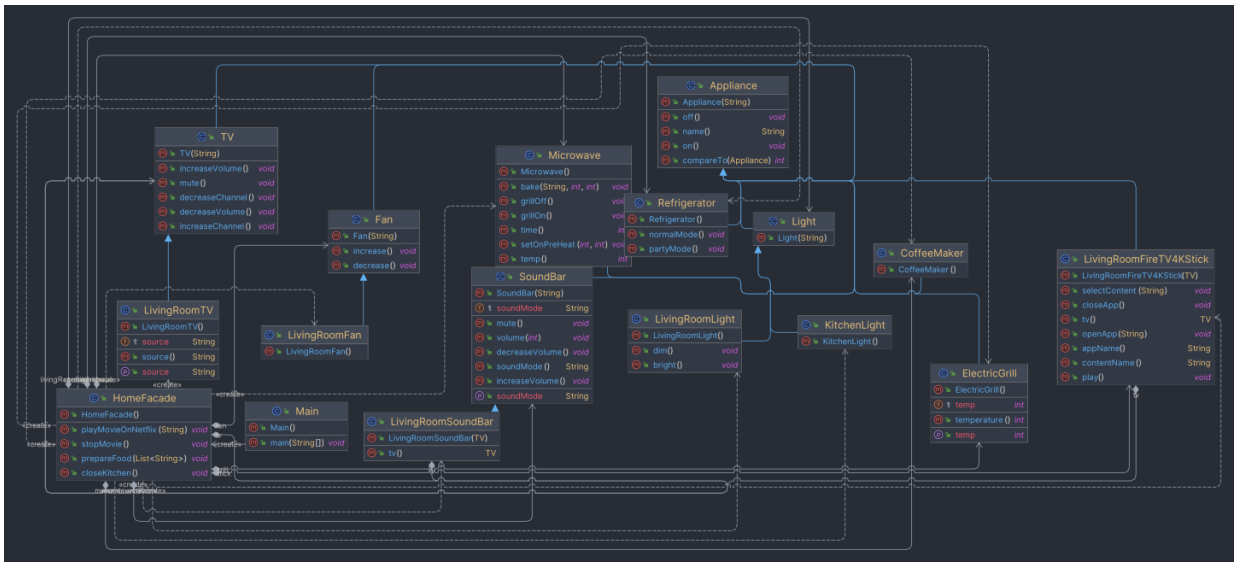

Figura 3: Diagrama UML de bajo nivel que da muestra de la estructura general del diseño del proyecto escogido. Además, muestra los atributos y los métodos para cada una de las clases empleadas

3. Patrón Facade en el proyecto

3.1. Información y estructura

Debido a la sencillez del proyecto y, teniendo en cuenta que este fue diseñado precisamente para mostrar el comportamiendo de este patrón, todo el trabajo apreciado en la figura 2 corresponde con el patrón facade. Como ya se menconó antes, en este se pueden apreciar los subsistemas (subclases) que están conectados al facade (clase HomeFacade en este caso) que se encarga de unificar el código y bajar la complejidad y el acoplamiento.

3.2. Motivación/ventajas

El sentido de haber utilizado este patrón en el proyecto es mostrar como, por medio de una única clase facade (HomeFacade) es posible acceder a todos los subsistemas que componen al programa. El usuario no tiene que preocuparse por el análisis o la generación del código. No es necesario que conozca las (poderosas) interfaces de bajo nivel ya que esto solo complicaría su tarea. Para proporcionar una interfaz de nivel superior

que pueda proteger a los clientes de estas clases, el subsistema del compilador incluye la clase HomeFacade que se encarga que unificar las subclases sin ocultarlas por completo. La facade del compilador hace la vida más fácil para la mayoría de los programadores sin ocultar la funcionalidad de nivel inferior de los pocos que la necesitan.

3.3. Desventajas

Entre algunas de las desventajas que hay al utilizar el patrón facade en este proyecto, destacan las siguientes:

- Aumento de la complejidad: Si bien la facade (clase HomeFacade) proporciona una interfaz simplificada para un subsistema complejo, la implementación interna del subsistema sigue siendo compleja. Esto puede llevar a un aumento en la complejidad general del sistema, ya que la fachada no elimina la complejidad, sino que la oculta.
- Exceso de dependencia de la facade: En este caso, los clientes se vuelven demasiado dependientes de la facade y se usa como un punto central de acceso a todas las funcionalidades del subsistema.

3.4. Otras posibles soluciones

Algunas de las posibles soluciones son las siguientes:

- Inyección de dependencias: En lugar de utilizar una facade para ocultar la complejidad de un subsistema, se podría utilizar la inyección de dependencias para proporcionar a los clientes solo los componentes específicos que necesitan. Esto permitiría una mayor flexibilidad y modularidad, ya que cada cliente puede acceder directamente a los componentes necesarios sin depender de una facade generalizada.
- Interfaz específica por cliente: En lugar de proporcionar una única facade que abarque todas las funcionalidades del subsistema, se podría considerar definir interfaces específicas para cada cliente o grupo de clientes. Cada interfaz se adaptaría a las necesidades particulares del cliente y proporcionaría solo las funcionalidades necesarias. Esto permitiría una mayor granularidad y personalización, evitando la dependencia de una única facade general.
- Composición de objetos: En lugar de utilizar la facade como un punto centralizado de acceso al subsistema, se podría utilizar la composición de objetos para construir y configurar los componentes del subsistema directamente en los clientes. Esto permitiría una mayor flexibilidad en la configuración y personalización de los componentes, ya que los clientes podrían elegir y combinar los objetos necesarios de acuerdo con sus requerimientos específicos.