**Propósito**

Facade es un patrón de diseño estructural que proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.

**Problema**

Imagina que debes lograr que tu código trabaje con un amplio grupo de objetos que pertenecen a una sofisticada biblioteca o framework. Normalmente, debes inicializar todos esos objetos, llevar un registro de las dependencias, ejecutar los métodos en el orden correcto y así sucesivamente.

Como resultado, la lógica de negocio de tus clases se vería estrechamente acoplada a los detalles de implementación de las clases de terceros, haciéndola difícil de comprender y mantener.

**Solución**

Una fachada es una clase que proporciona una interfaz simple a un subsistema complejo que contiene muchas partes móviles. Una fachada puede proporcionar una funcionalidad limitada en comparación con trabajar directamente con el subsistema. Sin embargo, tan solo incluye las funciones realmente importantes para los clientes.

Tener una fachada resulta útil cuando tienes que integrar tu aplicación con una biblioteca sofisticada con decenas de funciones, de la cual sólo necesitas una pequeña parte.

Por ejemplo, una aplicación que sube breves vídeos divertidos de gatos a las redes sociales, podría potencialmente utilizar una biblioteca de conversión de vídeo profesional. Sin embargo, lo único que necesita en realidad es una clase con el método simple codificar(nombreDelArchivo, formato). Una vez que crees dicha clase y la conectes con la biblioteca de conversión de vídeo, tendrás tu primera fachada.

**Aplicabilidad**

  Utiliza el patrón Facade cuando necesites una interfaz limitada pero directa a un subsistema complejo.

 A menudo los subsistemas se vuelven más complejos con el tiempo. Incluso la aplicación de patrones de diseño suele conducir a la creación de un mayor número de clases. Un subsistema puede hacerse más flexible y más fácil de reutilizar en varios contextos, pero la cantidad de código de configuración que exige de un cliente, crece aún más. El patrón Facade intenta solucionar este problema proporcionando un atajo a las funciones más utilizadas del subsistema que mejor encajan con los requisitos del cliente.

Utiliza el patrón Facade cuando quieras estructurar un subsistema en capas.

 Crea fachadas para definir puntos de entrada a cada nivel de un subsistema. Puedes reducir el acoplamiento entre varios subsistemas exigiéndoles que se comuniquen únicamente mediante fachadas.

Por ejemplo, regresemos a nuestro framework de conversión de vídeo. Puede dividirse en dos capas: la relacionada con el vídeo y la relacionada con el audio. Puedes crear una fachada para cada capa y hacer que las clases de cada una de ellas se comuniquen entre sí a través de esas fachadas. Este procedimiento es bastante similar al patrón [Mediator](https://refactoring.guru/es/design-patterns/mediator).

**Pros**

* Puedes aislar tu código de la complejidad de un subsistema.

## Contras

* Una fachada puede convertirse en [un objeto todopoderoso](https://refactoring.guru/es/antipatterns/god-object) acoplado a todas las clases de una aplicación.