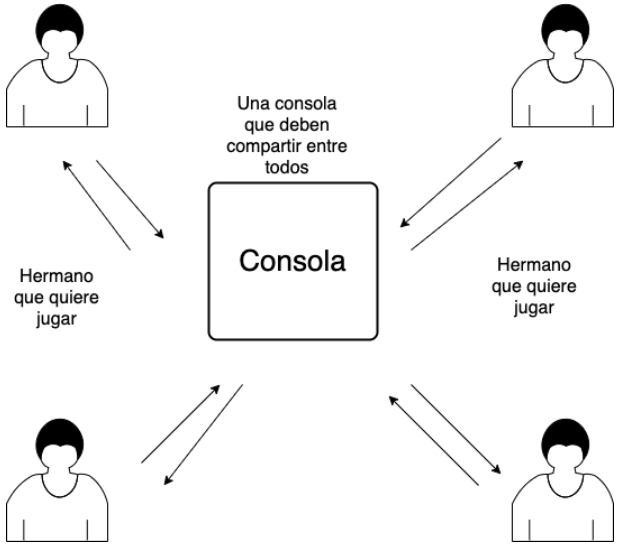
Patrones

Crysthel Aparicio 31511318

Singleton

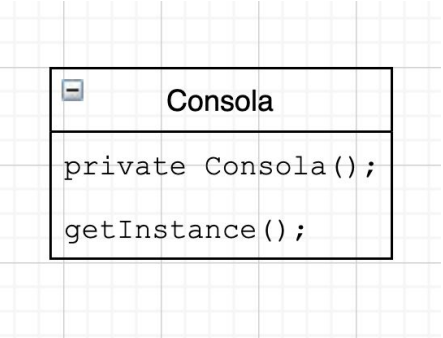
Desde una perspectiva de la vida real, vemos esto todo el tiempo. Siempre que solo haya una instancia de un producto (por ejemplo, una aspiradora) y se compartiera con varias personas (por ejemplo, compañeros de habitación).



Mi ejemplo es un hogar con 4 hermanos donde tienen la mala suerte de **compartir la misma consola**. Esto tiene sentido, ya que es algo que viví, y muchos han vivido.

Sin el patrón singleton, simplemente tengo un sistema con dos tipos de objetos: la consola y el objeto hermano.
El objeto de hermano contiene el juego que queremos jugar. Para jugar, le pedimos a un hermano que juegue.

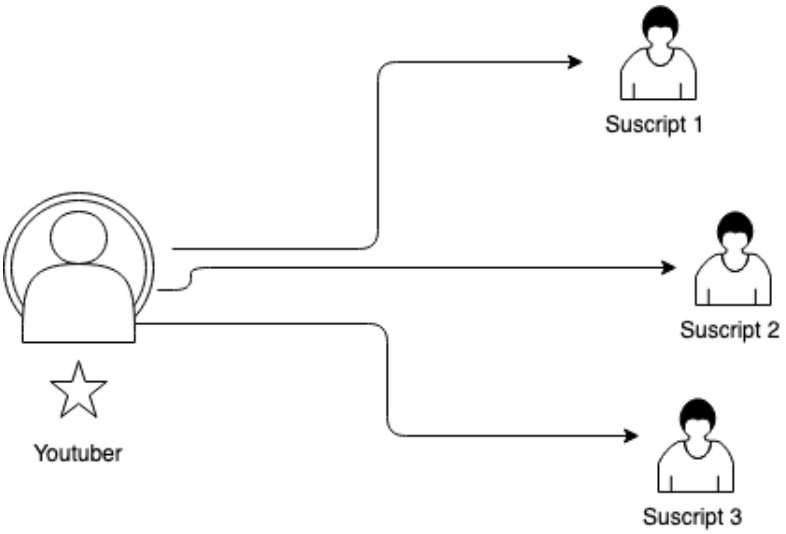
Sin embargo, para que el objeto hermano juegue, necesita una instancia de una consola. No proporcionaré a cada hermano su propia instancia de la consola, ya que eso no simulará el escenario real.
En su lugar, proporcionamos la consola como argumento para cuando a un hermano le toque jugar pues que juegue.



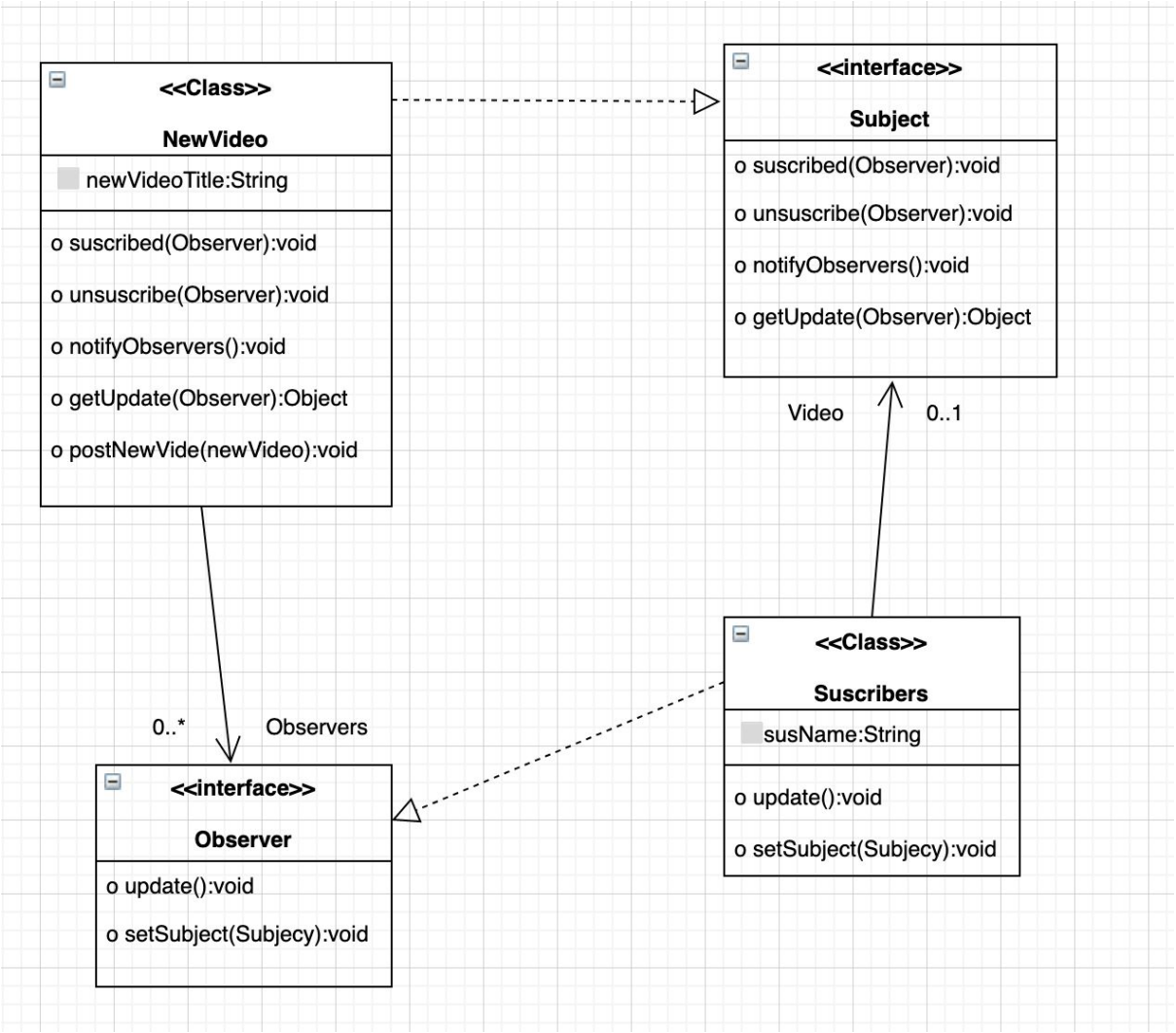
Observer

En este patrón, hay muchos observadores (objetos) que observan un sujeto (objeto). Los observadores están básicamente interesados y quieren que se les notifique cuando hay un cambio realizado dentro del tema. Por lo tanto, se registran con el tema.

Mi ejemplo un youtuber que tiene muchos suscriptores. Cada uno de estos subs quiere obtener las últimas actualizaciones de su youtuber favorito. Por lo tanto, él / ella puede suscribirse a el canal mientras su interés persista. Cuando pierde interés, simplemente deja de seguir a ese canal. Aquí los suscriptores son observadores y el youtuber es un tema.

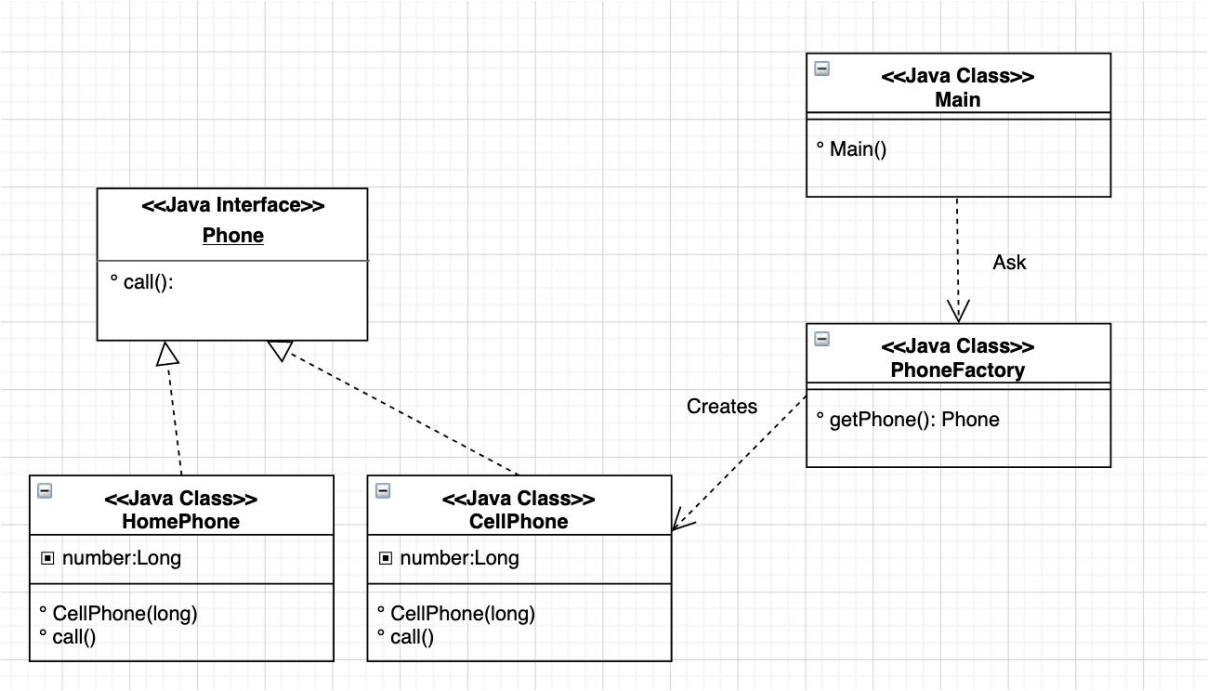


Donde el Youtuber es el Subject y el Suscriptor es el Observer.



Factory

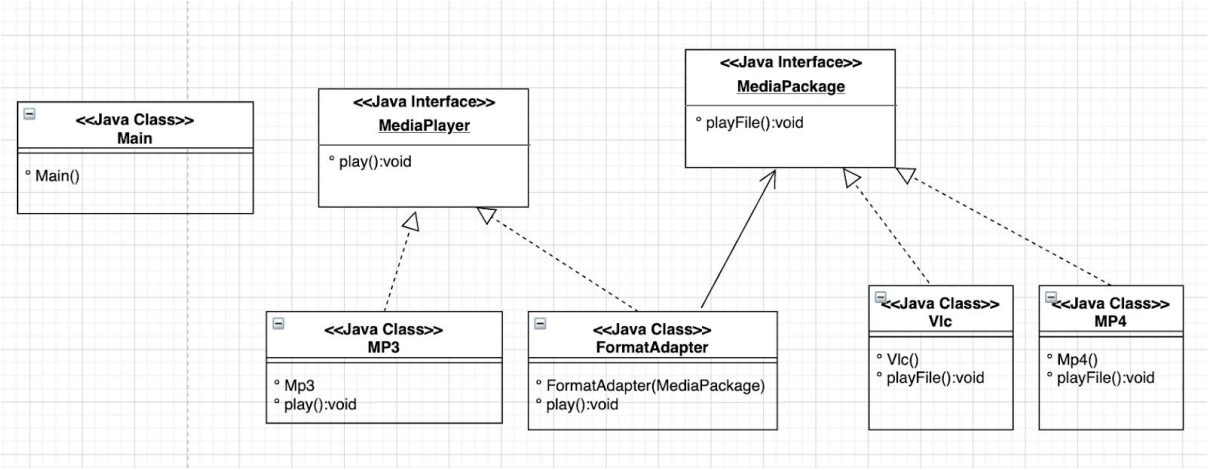
El método Factory es un patrón de creación que ayuda a abstraer el proceso de creación de objetos. Lo que esto significa es que la clase que usa el objeto no es necesariamente la clase que instancia el objeto. De hecho, ¡la clase que usa el objeto no tiene absolutamente ninguna idea sobre el tipo exacto del objeto creado! Simplemente tiene una referencia a la interfaz. Usaré como ejemplo un caso en donde al mudarse a un nuevo país y todavía no tener un teléfono, pero debes llamar a tu madre y decir que has llegado con seguridad (suponiendo que no le guste tener télefono por su edad avanzada). Entonces decide comprar un teléfono celular y le pide a la compañía telefónica que le envíe uno. Después de recibirlo, se da cuenta de que es muy costoso para usted, por lo que decide obtener un teléfono residencial y finalmente hace la llamada.



Adapter

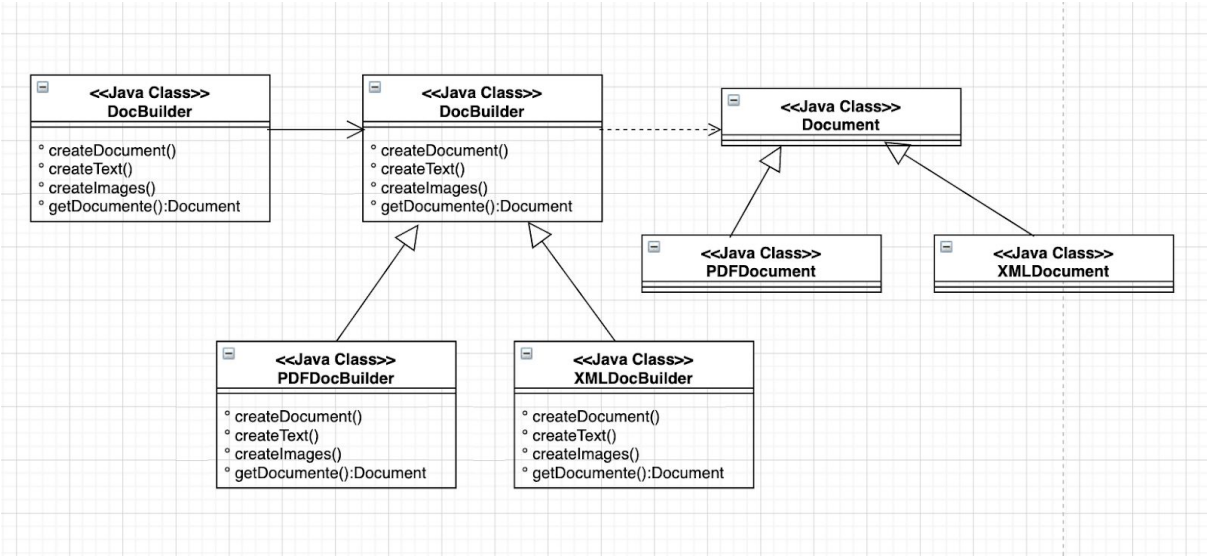
El patrón usa una interfaz y su implementación para crear una capa alrededor de otro objeto, que luego permite que el **comportamiento del objeto se convierta en la forma deseada**. Aunque gestiona el comportamiento de otra clase, no lo cambia de ninguna manera. Por lo tanto, no debería sorprender que el patrón del adaptador se clasifique como un **patrón de diseño estructural**.

Como ejemplo, de algo que me pasaba antes de spotify..., tenemos dos interfaces incompatibles: MediaPlayer y MediaPackage. La clase MP3 es una implementación de la interfaz MediaPlayer y tenemos VLC y MP4 como implementaciones de la interfaz MediaPackage. Queremos usar implementaciones de MediaPackage como instancias de MediaPlayer. Entonces, necesitamos crear un adaptador para ayudar a trabajar con dos clases incompatibles.
El adaptador se llamará FormatAdapter y debe implementar la interfaz MediaPlayer. Además, la clase FormatAdapter debe tener una referencia a MediaPackage, la interfaz incompatible.



Builder

La característica clave del patrón Builder es que implica un proceso paso a paso para construir algo, es decir, cada producto seguirá el mismo proceso aunque cada paso sea diferente. El ejemplo más práctico es un Builder de Documentos.



State

El patrón de estado no especifica dónde se definirán las transiciones de estado. Las opciones son dos: el objeto "contexto", o cada clase individual derivada del Estado. La ventaja de esta última opción es la facilidad de agregar nuevas clases derivadas de estado. La desventaja es que cada clase derivada del estado tiene conocimiento de acoplamiento a sus hermanos, lo que introduce dependencias entre subclases.

Usaré un ejemplo práctico como un ATM con tarjetas de Crédito ó Débito.