ARQUITECTURA DE SOFTWARE

EVALUACIÓN 2

EXPOSICIÓN – 10 PUNTOS

EQUIPOS: 2 PERSONAS

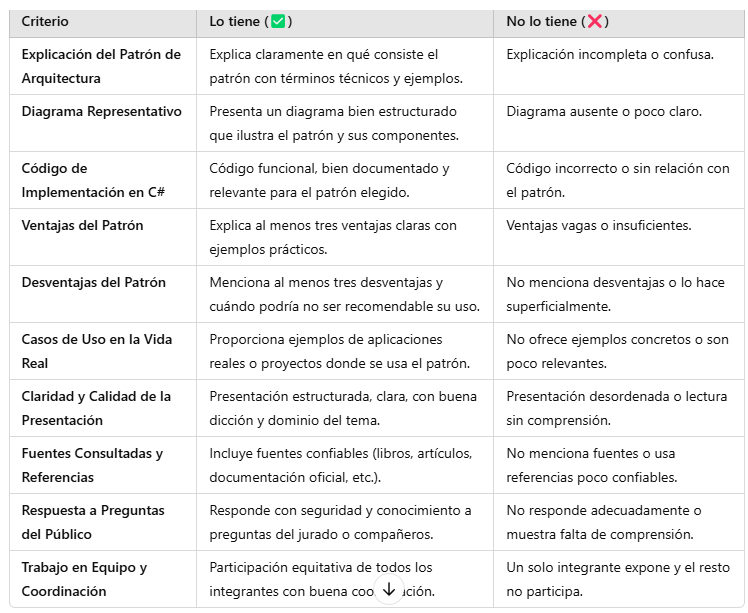
TEMA: PATRONES ARQUITECTÓNICOS DE COMPORTAMIENTO

CONTENIDO DE LA EXPOSICIÓN:

* Explicar de qué se trata
* Diagrama básico del Patrón y Diagrama del ejemplo
* Código implementando el patrón en C#
* En qué situaciones del día a día es más utilizado
* Los pro y los contra
* Fuentes consultadas.

Se califica por Forms, 50% los estudiantes, 50% el profesor.

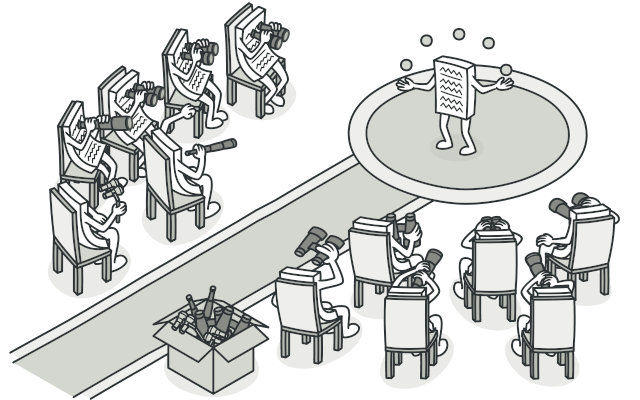
Rúbrica:



# Observer

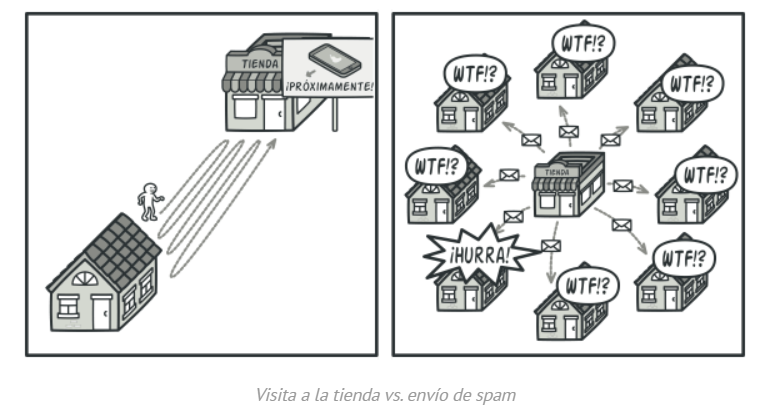
**También llamado:** Observador, Publicación-Suscripción, Modelo-patrón, Event-Subscriber, Listener

Observer es un patrón de diseño de comportamiento que te permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.



Imagina que tienes dos tipos de objetos: un objeto Cliente y un objeto Tienda. El cliente está muy interesado en una marca particular de producto (digamos, un nuevo modelo de iPhone) que estará disponible en la tienda muy pronto.

El cliente puede visitar la tienda cada día para comprobar la disponibilidad del producto. Pero, mientras el producto está en camino, la mayoría de estos viajes serán en vano.



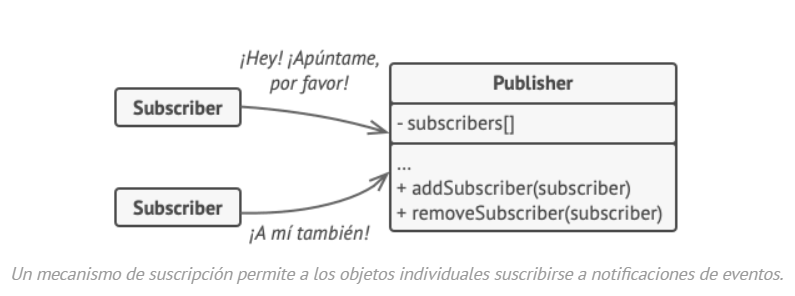
Por otro lado, la tienda podría enviar cientos de correos (lo cual se podría considerar spam) a todos los clientes cada vez que hay un nuevo producto disponible. Esto ahorraría a los clientes los interminables viajes a la tienda, pero, al mismo tiempo, molestaría a otros clientes que no están interesados en los nuevos productos.

Parece que nos encontramos ante un conflicto. O el cliente pierde tiempo comprobando la disponibilidad del producto, o bien la tienda desperdicia recursos notificando a los clientes equivocados.

## Solución

El objeto que tiene un estado interesante suele denominarse *sujeto*, pero, como también va a notificar a otros objetos los cambios en su estado, le llamaremos *notificador* (en ocasiones también llamado *publicador*). El resto de los objetos que quieren conocer los cambios en el estado del notificador, se denominan *suscriptores*.

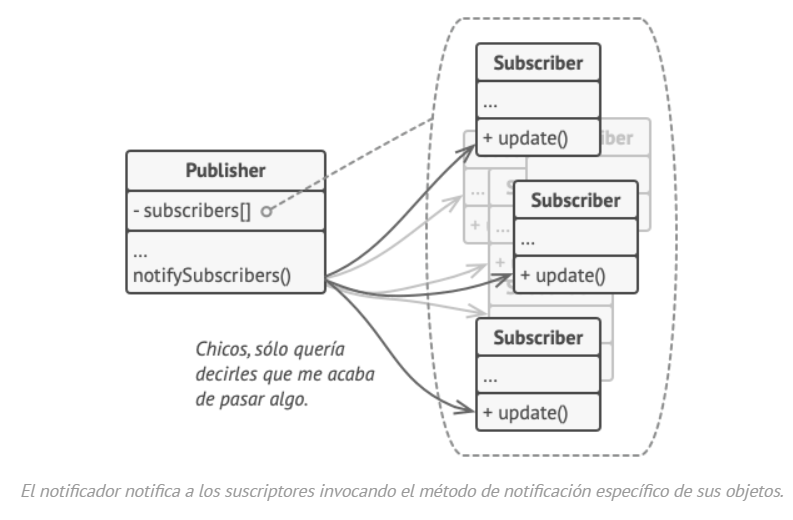
El patrón Observer sugiere que añadas un mecanismo de suscripción a la clase notificadora para que los objetos individuales puedan suscribirse o cancelar su suscripción a un flujo de eventos que proviene de esa notificadora. ¡No temas! No es tan complicado como parece. En realidad, este mecanismo consiste en: 1) un campo matriz para almacenar una lista de referencias a objetos suscriptores y 2) varios métodos públicos que permiten añadir suscriptores y eliminarlos de esa lista.



Ahora, cuando le sucede un evento importante al notificador, recorre sus suscriptores y llama al método de notificación específico de sus objetos.

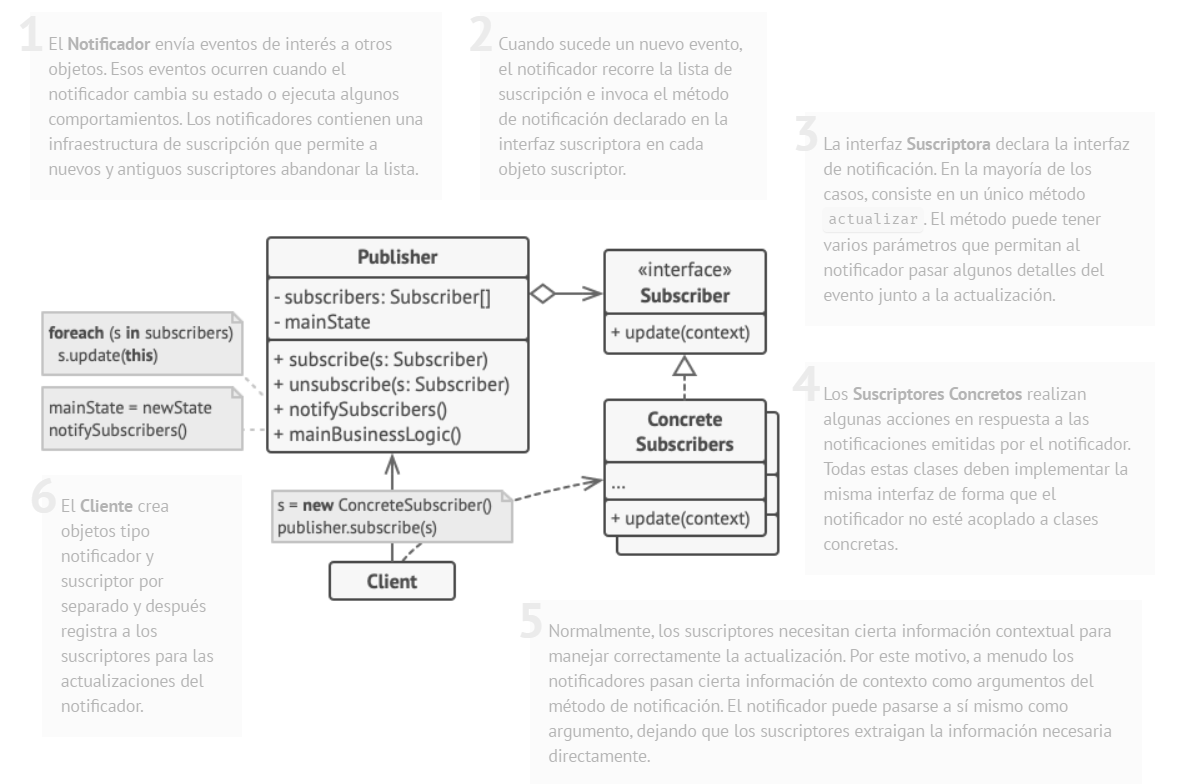
Las aplicaciones reales pueden tener decenas de clases suscriptoras diferentes interesadas en seguir los eventos de la misma clase notificadora. No querrás acoplar la notificadora a todas esas clases. Además, puede que no conozcas algunas de ellas de antemano si se supone que otras personas pueden utilizar tu clase notificadora.

Por eso es fundamental que todos los suscriptores implementen la misma interfaz y que el notificador únicamente se comunique con ellos a través de esa interfaz. Esta interfaz debe declarar el método de notificación junto con un grupo de parámetros que el notificador puede utilizar para pasar cierta información contextual con la notificación.



Si tu aplicación tiene varios tipos diferentes de notificadores y quieres hacer a tus suscriptores compatibles con todos ellos, puedes ir más allá y hacer que todos los notificadores sigan la misma interfaz. Esta interfaz sólo tendrá que describir algunos métodos de suscripción. La interfaz permitirá a los suscriptores observar los estados de los notificadores sin acoplarse a sus clases concretas.

## Estructura

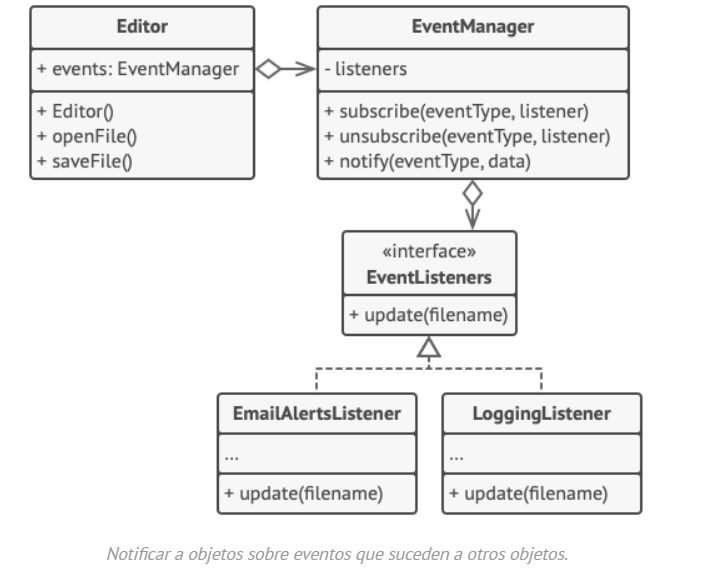


Código Implementación

La lista de suscriptores se compila dinámicamente: los objetos pueden empezar o parar de escuchar notificaciones durante el tiempo de ejecución, dependiendo del comportamiento que desees para tu aplicación.

En esta implementación, la clase editora no mantiene la lista de suscripción por sí misma. Delega este trabajo al objeto ayudante especial dedicado justo a eso. Puedes actualizar ese objeto para que sirva como despachador centralizado de eventos, dejando que cualquier objeto actúe como notificador.

Añadir nuevos suscriptores al programa no requiere cambios en clases notificadoras existentes, siempre y cuando trabajen con todos los suscriptores a través de la misma interfaz.



## Analogía en el mundo real

Si te suscribes a un periódico o una revista, ya no necesitarás ir a la tienda a comprobar si el siguiente número está disponible. En lugar de eso, el notificador envía nuevos números directamente a tu buzón justo después de la publicación, o incluso antes.

El notificador mantiene una lista de suscriptores y sabe qué revistas les interesan. Los suscriptores pueden abandonar la lista en cualquier momento si quieren que el notificador deje de enviarles nuevos números.

