Patrón
Observer

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

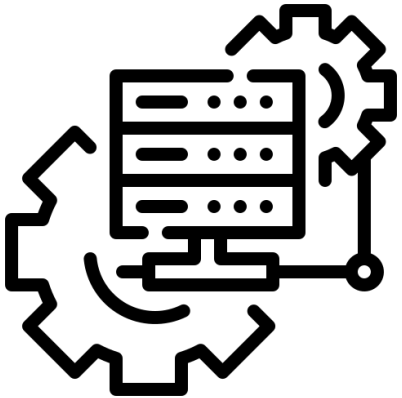
1 | Motivación

Vas a aprender una manera en que funcionan las notificaciones en relación de uno a muchos.

Propósito

Un determinado objeto puede tener otros dependientes de este. Estos otros objetos tal vez necesiten actualizarse según cambio de estado en el objeto del que dependen. Al intentar implementar esta lógica surgen muchas dificultades.

El patrón ***Observer*** propone una solución creando una interfaz que, cuando un objeto cambie de estado, se notifique y se actualicen automáticamente todos los objetos que dependen de él.



Solución

Se dispone de una interfaz para el **SujetoObservable** (Observable) y una para los **Observadores** (Observador). La clase concreta que será observada implementa la interface *Observable* y los observadores concretos implementan a *Observador*. Estas dos interfaces tienen un método que deben declarar las clases concretas, y por medio de estos métodos es que se actualizarán los **observadores** con cada cambio de estado en el **sujetoObservable**.

Así, siempre se obtendrán las actualizaciones de estado independientemente del tipo de objeto que sean tanto los observadores como el sujeto observable.

Ventajas y desventajas

✓	Permite modificar los sujetos y observadores de forma independiente. Es posible reutilizar objetos sin volver a utilizar sus observadores o viceversa. Esto añade escalabilidad al permitir añadir observadores sin modificar el sujeto u otros observadores.
✓	A diferencia de una petición ordinaria, la notificación enviada por un sujeto no necesita especificar su receptor. Esto genera una difusión entre todos los observadores interesados que haya.
✓	Gracias a que sujeto y observador no están fuertemente acoplados, pueden pertenecer a diferentes capas de abstracción de un sistema.
✗	Una actualización aparentemente inofensiva sobre el sujeto puede generar una serie de actualizaciones en cascada de los observadores y sus objetos dependientes. Esto puede generar falsas actualizaciones muy difíciles de localizar.

2 | Diagrama UML

Diagrama Solución Patrón *Observer*

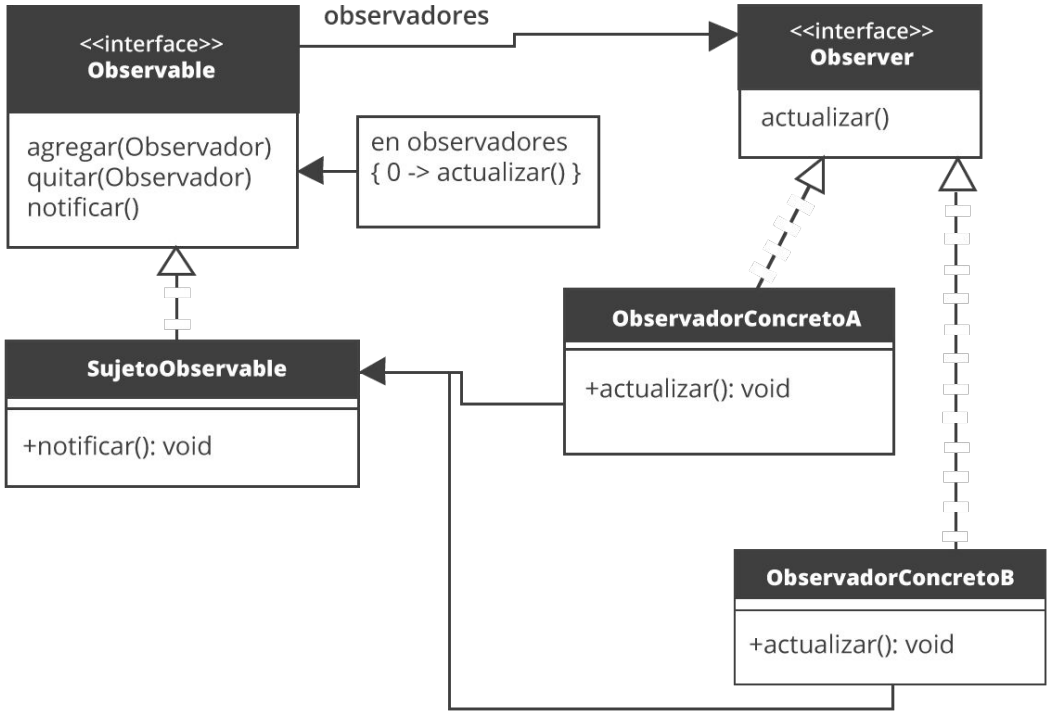
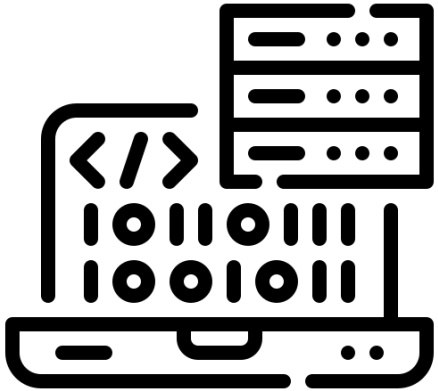


Diagrama Solución Patrón *Observer*

- **Interfaz observable (Sujeto)**: cada implementación (sujeto) conoce a sus observadores y puede ser observado por cualquier número de observadores. También proporciona una interfaz para agregar o quitar observadores.
- **Observador**: define una interfaz para que cada implementación (ObservadorConcreto) pueda actualizar los objetos que deben ser notificados de cambios en el sujeto.
- **SujetoConcreto**: envía una notificación a sus observadores cuando cambia su estado.
- **ObservadorConcreto**: mantiene referencia a un objeto **SujetoConcreto**. Guarda un estado que debería ser consistente con el del **Sujeto**. Implementa la interfaz de actualización del **Observador** para mantener su estado sincronizado con el del sujeto.

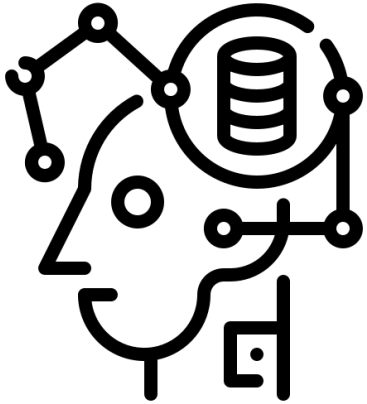
¿Cómo funciona?

Cuando el **SujetoObservable** sufre algún cambio de estado, se ejecuta el método "*notificar()*", el cual recorre una lista que contiene todos los objetos que observan al **SujetoObservable** y llama a su método "*actualizar()*". De esta manera se mantienen todos los observadores actualizados ante cualquier cambio, sin necesidad de que estén consultando constantemente si existen actualizaciones de estado.



Conclusiones

El patrón crea una dependencia directa de cada observador hacia el sujeto. Si bien esto puede llegar a traer complicaciones, es la manera en la que está estructurado el patrón.



Algo que **no debe suceder** es que el **sujeto** dependa de algún **observador**. Este comportamiento vulneraría la ignorancia que debe tener el sujeto sobre sus observadores y podría generar dependencias cíclicas.

Es conveniente especificar las modificaciones de interés explícitamente. Se puede mejorar la eficiencia extendiendo la interfaz de registro del sujeto para permitir que los observadores registren solo aquellos los eventos concretos que le interesen.


DigitalHouse>
Coding School