Contenido

[1. PROPOSITO DEL PATRON OBSERVADOR 1](#_Toc484563539)

[2. APLICABILIDAD 1](#_Toc484563540)

[2.1. DEBE SER UTILIZADO CUANDO: 1](#_Toc484563541)

[3. DIAGRAMA DE CLASES GENERICO (PATRON OBSERVADOR) 2](#_Toc484563542)

[3.1. PARTICIPANTES 2](#_Toc484563543)

[4. PROBLEMA 3](#_Toc484563544)

[5. DIAGRAMA DE CLASES UTILIZANDO EL PATRON 4](#_Toc484563545)

# PROPOSITO DEL PATRON OBSERVADOR

El patrón *observador*, es flexible y sencillo. Se utiliza cuando uno quiere notificar a otros objetos de un evento. En principio, lo que sucede es que un Objeto (llamémoslo *Observador*) se inscribe a otro Objeto (llamémoslo *Sujeto*) y este le avisa cuando un evento es disparado (o cuando el estado del *Sujeto* ha cambiado).

Se pueden inscribir varios *Observadores*, en ese caso (generalmente) la notificación del cambio de estado va de acuerdo al orden en cómo se suscribieron los *Observadores*.

“Básicamente debemos saber que el patrón **Observer** es un patrón de comportamiento que permite relacionar diferentes objetos entre si en torno a uno Principal, así cada vez que este ultimo cambie su estado, los demás también cambiaran de forma automática....”

# APLICABILIDAD

Puede pensarse en aplicar este patrón cuando una modificación en el estado de un objeto requiere cambios de otros, y no deseamos que se conozca el número de objetos que deben ser cambiados. También cuando queremos que un objeto sea capaz de notificar a otros objetos sin hacer ninguna suposición acerca de los objetos notificados y cuando una abstracción tiene dos aspectos diferentes, que dependen uno del otro; si encapsulamos estos aspectos en objetos separados permitiremos su variación y reutilización de modo independiente.

## DEBE SER UTILIZADO CUANDO:

* Un objeto necesita notificar a otros objetos cuando cambia su estado. La idea es encapsular estos aspectos en objetos diferentes permite variarlos y reutilizarlos independientemente.
* Cuando existe una relacion de dependencia de uno a muchos que puede requerir que un objeto notifique a multiples objetos que dependen de el cuando cambia su estado.

# DIAGRAMA DE CLASES GENERICO (PATRON OBSERVADOR)

## PARTICIPANTES

Tendremos sujetos concretos cuyos cambios pueden resultar interesantes a otros y observadores a los que al menos les interesa estar pendientes de un elemento y en un momento dado, reaccionar ante sus notificaciones de cambio. Todos los sujetos tienen en común que un conjunto de objetos quieren estar pendientes de ellos. Cualquier elemento que quiera ser observado tiene que permitir indicar:

1. “Estoy interesado en tus cambios”.
2. “Ya no estoy interesado en tus cambios”.

El observable tiene que tener, además, un mecanismo de aviso a los interesados.

A continuación tenemos a los participantes de forma desglosada:

* **Sujeto (Subject):**

El sujeto proporciona una interfaz para agregar (attach) y eliminar (detach) observadores. El Sujeto conoce a todos sus observadores.

* **Observador (Observer):**

Define el método que usa el sujeto para notificar cambios en su estado (update/notify).

* **Sujeto Concreto (ConcreteSubject):**

Mantiene el estado de interés para los observadores concretos y los notifica cuando cambia su estado. No tienen porque ser elementos de la misma jerarquía.

* **Observador Concreto (ConcreteObserver):**

Mantiene una referencia al sujeto concreto e implementa la interfaz de actualización, es decir, guardan la referencia del objeto que observan, así en caso de ser notificados de algún cambio, pueden preguntar sobre este cambio.

# PROBLEMA

Vamos a suponer que en una biblioteca, cada vez que un lector devuelve un libro, se registra el estado en el que se esta devolviendo el libro, en este caso asumiremos que el estado esta malo o dañado y lo se que ara sera notificar a todos los departamentos que esten interasados en esto.

# DIAGRAMA DE CLASES UTILIZANDO EL PATRON



# 

# DIAGRAMA DE CLASES UTILIZANDO EL PATRON

