

Observer

1- patrones de diseño (comportamiento)

Python

Estefania Sosa Garcia

El patrón de observador (publicación-inscripción o modelo-patrón)

Se define :

- ❖ El patrón del observador resuelve un problema bastante común:

¿qué sucede si un grupo de objetos necesita actualizarse cuando algún objeto cambia de estado?

define uno-a-muchos

dependencia entre los objetos para que cuando uno el objeto cambia de estado, todos sus dependientes son notificado y actualizado automáticamente.

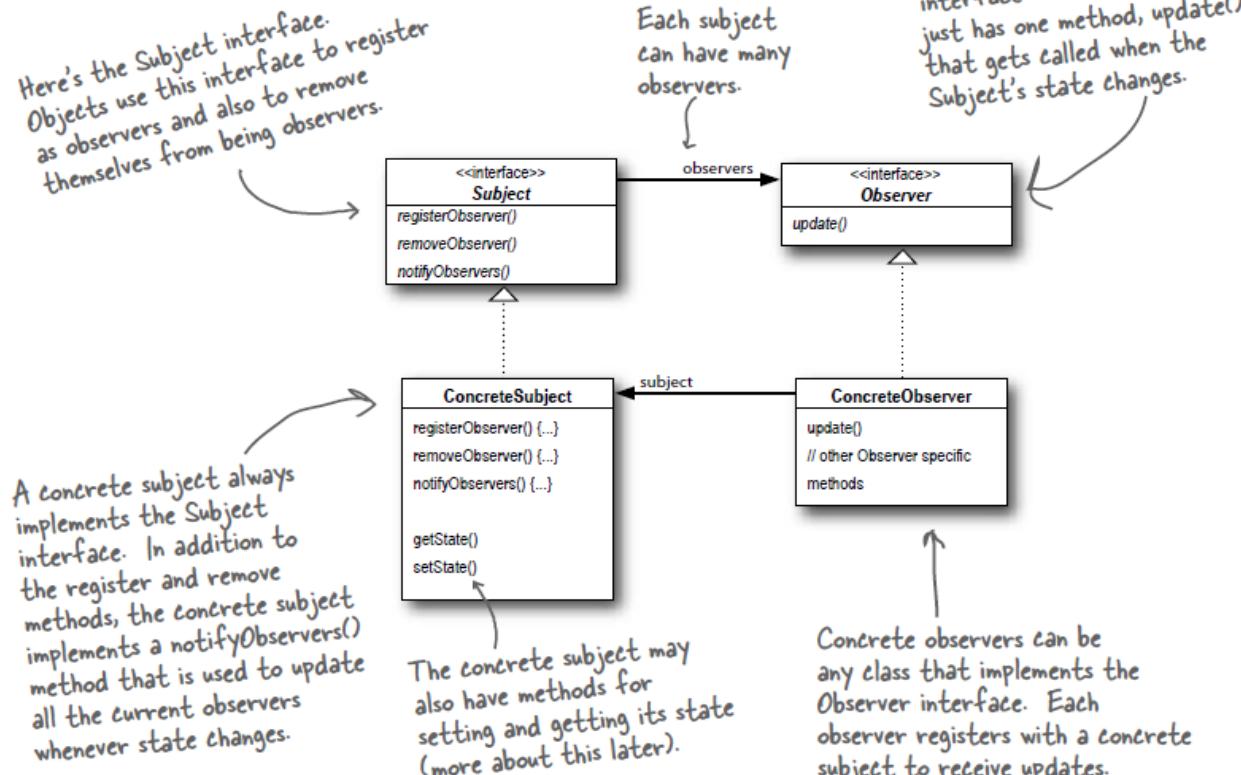
Pertenece a los patrones de comportamiento estos se basan en la asignación de responsabilidades y la comunicación entre objetos.

Identifican patrones comunes de comunicación, que ayudan a que ésta sea más flexible, y a que pueda ser llevada a cabo entre los objetos

Java

Con el patrón Observar,
El sujeto es el objeto que
contiene el
estado y lo controla. Entonces,
hay UNO
sujeto con estado. Los
observadores, en
Por otro lado, usa el estado,
incluso
si no lo poseen. Hay muchos
observadores y confían en el
sujeto
para decirles cuándo cambia su
estado.
Entonces, hay una relación
entre el
UNO Sujeto a MUCHOS
Observadores.

The Observer Pattern defined: the class diagram



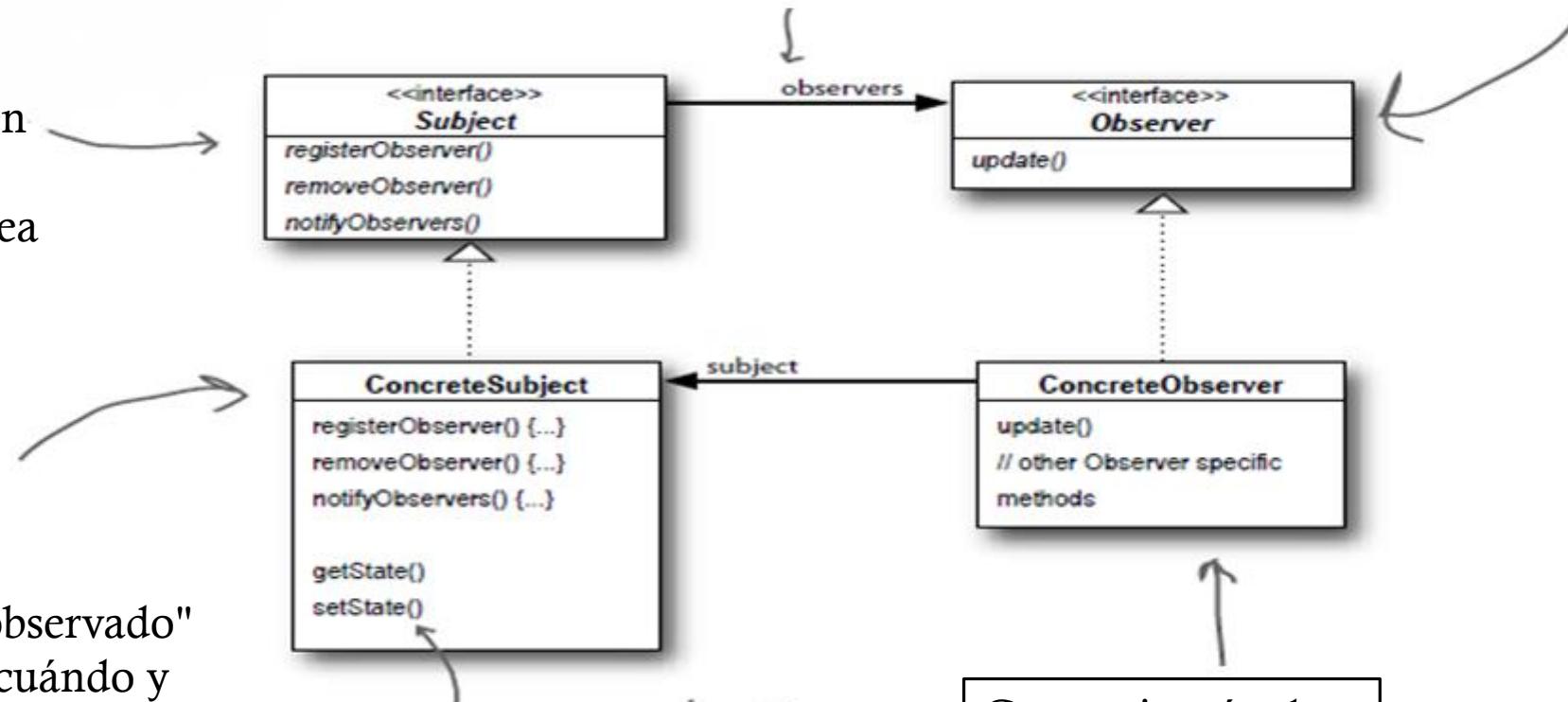
The Observer Pattern defined: the class diagram

Observadores.

La clase **Observable** realiza un seguimiento de todos los que quieren ser informados cuando ocurre un cambio, ya sea que el "estado" haya cambiado o no.

El método **notifyObservers ()** es parte de la clase base **Observable**

El "objeto observado" que decide cuándo y cómo hacer la actualización se llamará **Observable**



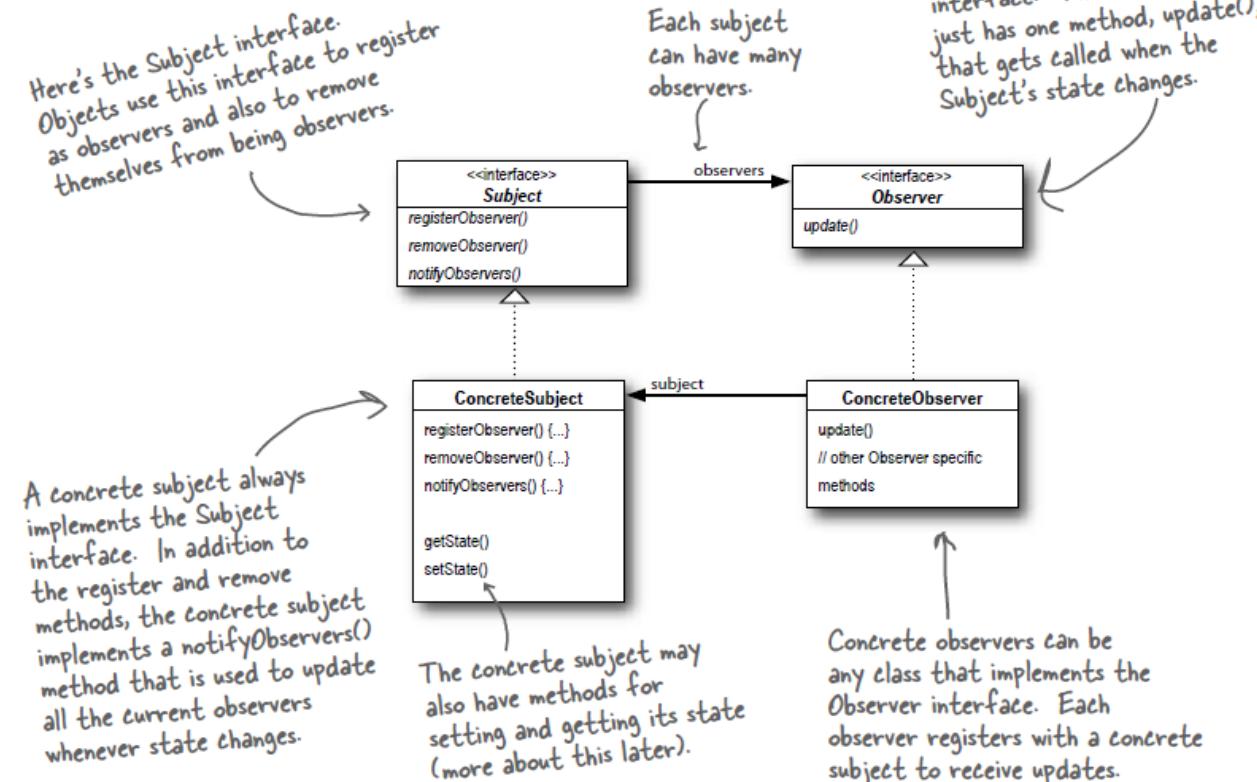
Construir método sincronización
synchronized ()

La función **synchronized ()** toma un método y lo envuelve en una función que agrega la funcionalidad mutex

Java

Con el patrón Observar,
El sujeto es el objeto que
contiene el
estado y lo controla. Entonces,
hay UNO
sujeto con estado. Los
observadores, en
Por otro lado, usa el estado,
incluso
si no lo poseen. Hay muchos
observadores y confían en el
sujeto
para decirles cuándo cambia su
estado.
Entonces, hay una relación
entre el
UNO Sujeto a MUCHOS
Observadores.

The Observer Pattern defined: the class diagram



Aplicación

Patrón de diseño Observer :

- ❖ a un código que realiza la autenticación de usuario
- ❖ CODIGO DONDE ES NECESARIA LA INTERCOMUNICACION ENTRE METODOS Y ESTADOS
 - ❖ Puede pensarse en aplicar este patrón cuando una modificación en el estado de un objeto requiere cambios de otros, y no se desea que se conozca el número de objetos que deben ser cambiados.
 - ❖ También cuando se quiere 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,

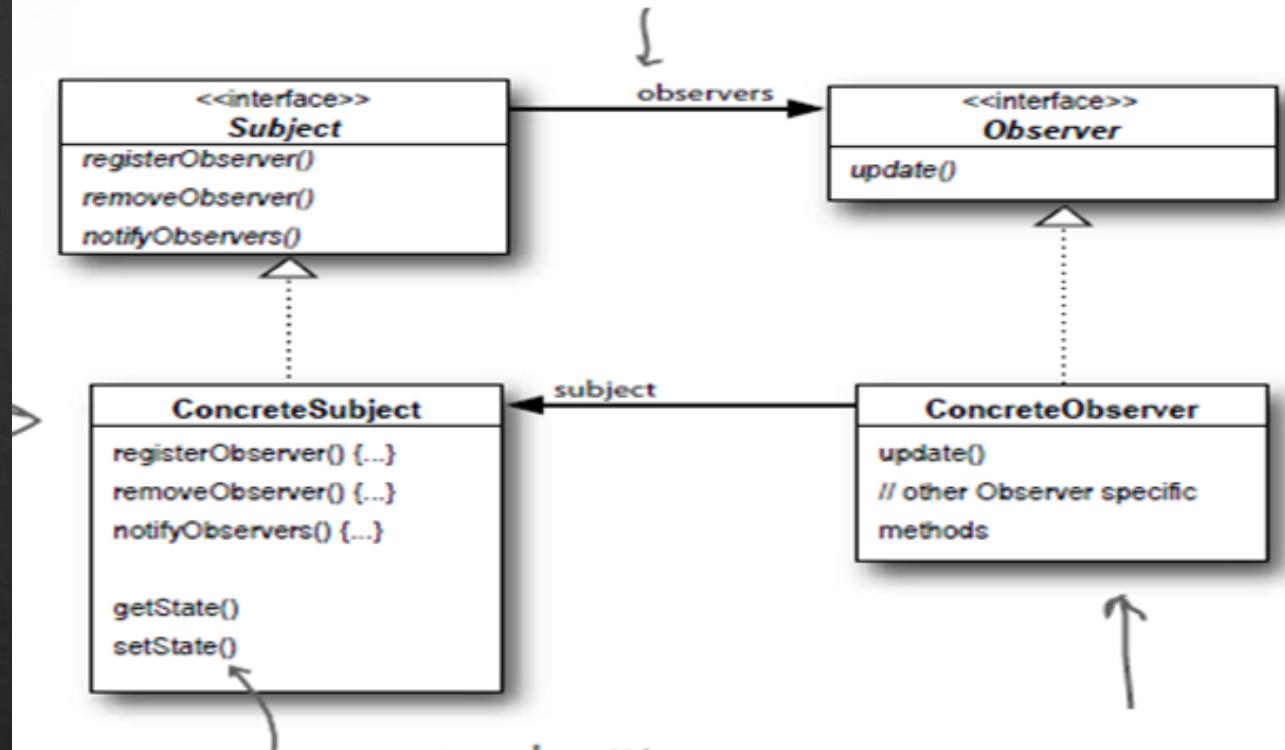
Subject: Conoce a sus observadores, Proporciona una Interfaz para que se suscriban los objetos Observer.

Observer: Define una interfaz para actualizar los objetos que deben ser notificados de cambios en el objeto Subject.

ConcreteSubject: Guarda el estado de interes para los objetos ConcreteObserver, Envia una notificacion a sus observadores cuando cambia su estado.

ConcreteObserver: Mantiene una referencia a un objeto ConcreteSubject, Guarda el estado que deberia permanecer sincronizado con el objeto observado, Implementa la interfaz Observer para mantener su estado consistente con el objeto observado.

Estructura



Ventajas y desventajas

- ❖ La principal ventaja de este patrón es que todo se logra sin recurrir a un acoplamiento estrecho. El *sujeto* solo sabe de una lista de *observadores* y en una sola llamada los notifica. Al *sujeto* no le interesa los efectos o desenlaces de los *observadores*, él simplemente emite. El resultado es código flexible y reusable.
- ❖ La única desventaja apreciable, aparece cuando un *observador* es demasiado grande. Eso puede traer consecuencias en el uso de memoria.

Observer()

+update()

Synchronization()

+f(*args)

+synchronize(Klass, name)

Observable()

+__init__()

addObserver(self,observer)

deleteObserver(self,obserer)

notifiObserver(selfObserver(self,arg))

deleteObserver()

setChange()

clearChanged()

hasChanged()

countObservablers()

synchronize(Observable,

"addObserver deleteObserver deleteObservers " +

"setChanged clearChanged hasChanged " +

"countObservers")

- ❖ Hay dos tipos de objetos usados para implementar el patrón de observador en Python.
 - ❖ La clase Observable realiza un seguimiento de todos los que quieren ser informados cuando ocurre un cambio, ya sea que el "estado" haya cambiado o no. Cuando alguien dice "OK, todos deben verificar y potencialmente actualizarse a sí mismos", la clase Observable realiza esta tarea llamando al método `notifyObservers()` para cada uno en la lista.
 - ❖ El método `notifyObservers()` es parte de la clase base Observable.
-
- ❖ En realidad, hay dos "cosas que cambian" en el patrón del observador: la cantidad de objetos de observación y la forma en que ocurre una actualización. Es decir, el patrón del observador le permite modificar ambos sin afectar el código circundante.

Código

- ❖ C:\Users\sombr\Documents\GitHub\diseño y arquitectura de software\Das_Sistemas\Ene-Jun-2018\Estefania Sosa\presentacion\ObservedFlower.py