Observer

Propósito

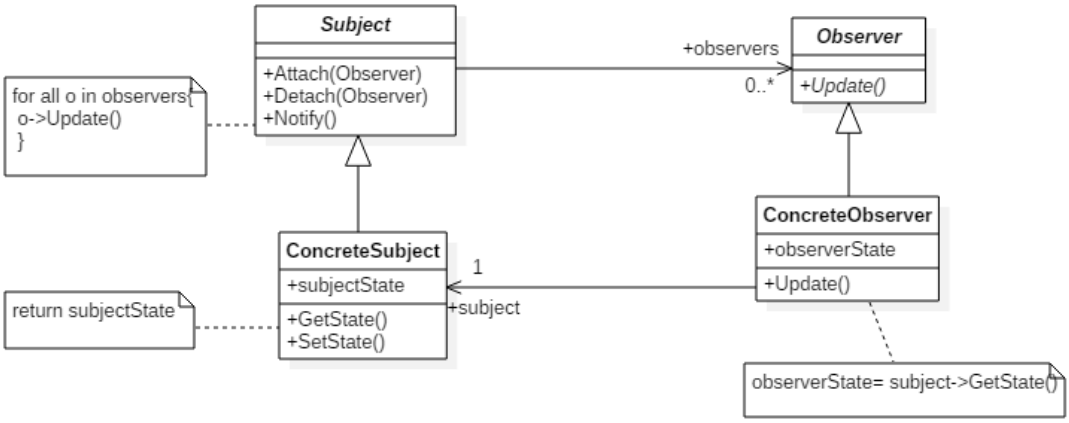
Define dependencias uno a muchos entre objetos, de modo que cuando cambia el estado de un objeto, todos los objetos dependientes reciben una notificación y se actualizan automáticamente.

Aplicabilidad

El patrón Observer se utiliza cuando:

- Cuando una abstracción tiene dos aspectos, una depende de la otra. Encapsular estos aspectos en objetos separados nos permite cambiarlos y reutilizarlos de forma independiente
- Cuando un cambio en un objeto implica cambios en otros y se desconoce cuántos objetos deben cambiar
- Cuando un objeto tenga que ser capaz de avisar a otros objetos sin tener que conocerlos. En otras palabras, no queremos que esos objetos estén fuertemente acoplados

Estructura



Participantes

- Subject: conoce a sus observadores. Cualquier número de objectos Observer puede observar al sujeto.
    - Proporciona una interface para suscribir o desuscribir Observer
- Observer: define una interface para que los objetos actualicen cuando reciben una notificación de cambio
- ConcreteSubject:
    - Guarda el estado que es objeto de interés para los objetos ConcreteObserver
    - Envía notificaciones a sus observadores cuando cambia el estado
- ConcreteObserver
    - Tiene una referencia al objeto ConcreteSubject
    - Almacena el estado que debería ser consistente con el del sujeto

- Implementa la interface de actualización que le permite mantener la consistencia con el estado del sujeto

Colaboraciones

- ConcreteSubject notifica a sus observadores siempre que ocurre un cambio que puede hacer inconsistente el estado de los observadores
- Después de recibir una notificación, un objeto ConcreteObserver realiza una consulta el estado del sujeto para actualizar su propia información

Conviene hacer notar que aunque el Observer provoque el cambio de estado en el sujeto, espera a recibir la notificación para consultar el nuevo estado, ya que el cambio de estado lo puede provocar cualquier otro objeto observador.

Consecuencias

El patrón Observer nos permite variar los sujetos y observadores de forma independiente. Puedes reutilizar los sujetos sin reutilizar sus observadores y viceversa. Este nos permite agregar observadores sin modificar los sujetos u otros observadores.

Veamos otros beneficios y efectos del patrón Observer:

1. Acoplamiento abstracto entre Subject y Observer. Todo lo que un sujeto conoce es que tiene una lista de observadores, cada uno sigue la interface definida por la clase abstracta Observer. El sujeto no conoce la clase concreta de ningún observador. De este modo el acoplamiento entre sujetos y observadores es abstracto y mínimo. Como Subject y Observer no están fuertemente acoplados, pueden pertenecer a diferentes capas de abstracción de un sistema. Un sujeto de una capa de bajo nivel puede comunicar e informar a un observador de una capa de alto nivel, manteniendo así la estructura en capas del sistema.
2. Soporte para la comunicación de difusión (broadcast). A diferencia de las peticiones normales, la notificación que envía un sujeto no necesita especificar su receptor. La notificación se envía a todos los objetos interesados que se suscribieron al sujeto. El sujeto no se preocupa de cuántos interesados hay, solamente se encarga de notificar a todos. Esto proporciona libertad de agregar o quitar observadores en cualquier momento. El observador decide si gestionar o ignorar la notificación.
3. Actualizaciones inesperadas. Como los observadores no se conocen entre sí, actúan ciegamente en cuanto al coste de modificar el sujeto. Un cambio inocuo en el sujeto puede provocar una cascada de actualizaciones en los observadores. El problema se agrava por el hecho de que el protocolo de actualización no detalla qué es lo que ha cambiado en el sujeto.