Memento

Propósito

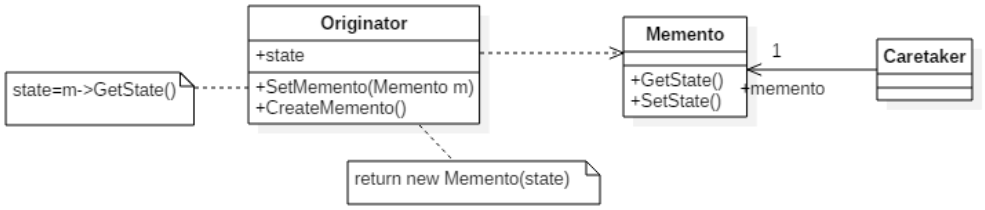
Si violar la encapsulación, captura y externaliza el estado interno de un objeto de modo que el objeto pueda recuperar ese estado más adelante.

Aplicabilidad

El Memento se utiliza cuando:

- Se desea guardar una instantánea del estado de un objeto para poderlo recuperar más adelante, y
- Una interface que permitiera obtener el estado expondría detalles de implementación, rompiéndose así la encapsulación

Estructura



Participantes

- Memento:
  - Almacena el estado interno de un objeto Originator. El Memento puede almacenar mucho o poco del estado interno de su generador (Originator)
  - Protege el acceso a objetos que no sean el Originator. Los mementos tienen dos interfaces. Caretaker "ve" una interface estrecha del Memento -sólo puede pasarle el memento a otros objetos. Originator, por el contrario, ve una interface más completa, una que le permite acceder a los datos necesarios para poder recuperar un estado previo. Idealmente, sólo debería poder acceder al estado interno del memento el originator que lo generó.
- Originator:
  - Crea un memento que contiene el estado interno actual
  - Utiliza el memento para restaurar su estado interno
- Caretaker:
  - Es responsable de guardar el memento
  - Nunca opera o examina los contenidos de un memento

Colaboraciones

- Un objeto Caretaker solicita un memento a un objeto Originator, lo guarda un tiempo, y lo devuelve al originator
- Los mementos son pasivos. Sólo el Originator que creó un memento, asignará o recuperará su estado

Consecuencias

El patrón Memento tiene las siguientes consecuencias:

1. Preserva los límites de la encapsulación. El Memento evita exponer información que solamente debería manejar el originator pero que se debe almacenar fuera del originator. El patrón protege a otros objetos de la potencial complejidad interna de un Originator, preservando así los límites de la encapsulación.
2. Simplifica a Originator. En otros diseños para preservar la encapsulación, Originator almacena versiones del estado interno que han solicitado los clientes. Esto provoca poner la carga de gestión del almacenamiento en Originator. Permitir que los clientes gestionen el estado que solicitan simplifica al Originator y además no deben notificarle cuando han terminado.
3. Utilizar mementos puede resultar costoso. Los mementos pueden incurrir en sobrecostes si el Originator tiene que almacenar mucha información o si el cliente crea y devuelve mementos al originator con demasiada frecuencia. Si la encapsulación y restauración del estado del Originator no es una operación “barata”, entonces puede que este patrón no sea adecuado.
4. Definir interfaces “estrechas” y “amplias”. En algunos lenguajes puede resultar difícil asegurar que sólo el originator tiene acceso al estado del memento.
5. Costes ocultos en el cuidado de los mementos. El objeto Caretaker es el responsable de eliminar los mementos que conserva. Sin embargo, el caretaker no sabe cuánta información supone el estado de un memento. Con lo cual, un caretaker que no sea ligero podría incurrir en un coste grande cuando almacene mementos.