**Propósito**

**Memento** es un patrón de diseño de comportamiento que te permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.

**Problema**

Imagina que estás creando una aplicación de edición de texto. Además de editar texto, tu programa puede formatearlo, asi como insertar imágenes en línea, etc.

En cierto momento, decides permitir a los usuarios deshacer cualquier operación realizada en el texto. Esta función se ha vuelto tan habitual en los últimos años que hoy en día todo el mundo espera que todas las aplicaciones la tengan. Para la implementación eliges la solución directa. Antes de realizar cualquier operación, la aplicación registra el estado de todos los objetos y lo guarda en un almacenamiento. Más tarde, cuando un usuario decide revertir una acción, la aplicación extrae la última instantánea del historial y la utiliza para restaurar el estado de todos los objetos.

Pensemos en estas instantáneas de estado. ¿Cómo producirías una, exactamente? Probablemente tengas que recorrer todos los campos de un objeto y copiar sus valores en el almacenamiento. Sin embargo, esto sólo funcionará si el objeto tiene unas restricciones bastante laxas al acceso a sus contenidos. Lamentablemente, la mayoría de objetos reales no permite a otros asomarse a su interior fácilmente, y esconden todos los datos significativos en campos privados.

Ignora ese problema por ahora y asumamos que nuestros objetos se comportan como hippies: prefieren relaciones abiertas y mantienen su estado público. Aunque esta solución resolvería el problema inmediato y te permitiría producir instantáneas de estados de objetos a voluntad, sigue teniendo algunos inconvenientes serios. En el futuro, puede que decidas refactorizar algunas de las clases editoras, o añadir o eliminar algunos de los campos. Parece fácil, pero esto también exige cambiar las clases responsables de copiar el estado de los objetos afectados.

Pero aún hay más. Pensemos en las propias “instantáneas” del estado del editor. ¿Qué datos contienen? Como mínimo, deben contener el texto, las coordenadas del cursor, la posición actual de desplazamiento, etc. Para realizar una instantánea debes recopilar estos valores y meterlos en algún tipo de contenedor.

Probablemente almacenarás muchos de estos objetos de contenedor dentro de una lista que represente el historial. Por lo tanto, probablemente los contenedores acaben siendo objetos de una clase. La clase no tendrá apenas métodos, pero sí muchos campos que reflejen el estado del editor. Para permitir que otros objetos escriban y lean datos a y desde una instantánea, es probable que tengas que hacer sus campos públicos. Esto expondrá todos los estados del editor, privados o no. Otras clases se volverán dependientes de cada pequeño cambio en la clase de la instantánea, que de otra forma ocurriría dentro de campos y métodos privados sin afectar a clases externas.

Parece que hemos llegado a un callejón sin salida: o bien expones todos los detalles internos de las clases, haciéndolas demasiado frágiles, o restringes el acceso a su estado, haciendo imposible producir instantáneas. ¿Hay alguna otra forma de implementar el "deshacer"?

**Solución**

Todos los problemas que hemos experimentado han sido provocados por una encapsulación fragmentada. Algunos objetos intentan hacer más de lo que deben. Para recopilar los datos necesarios para realizar una acción, invaden el espacio privado de otros objetos en lugar de permitir a esos objetos realizar la propia acción.

El patrón Memento delega la creación de instantáneas de estado al propietario de ese estado, el objeto originador. Por lo tanto, en lugar de que haya otros objetos intentando copiar el estado del editor desde el “exterior”, la propia clase editora puede hacer la instantánea, ya que tiene pleno acceso a su propio estado.

El patrón sugiere almacenar la copia del estado del objeto en un objeto especial llamado memento. Los contenidos del memento no son accesibles para ningún otro objeto excepto el que lo produjo. Otros objetos deben comunicarse con mementos utilizando una interfaz limitada que pueda permitir extraer los metadatos de la instantánea (tiempo de creación, el nombre de la operación realizada, etc.), pero no el estado del objeto original contenido en la instantánea.

Una política tan restrictiva te permite almacenar mementos dentro de otros objetos, normalmente llamados cuidadores. Debido a que el cuidador trabaja con el memento únicamente a través de la interfaz limitada, no puede manipular el estado almacenado dentro del memento. Al mismo tiempo, el originador tiene acceso a todos los campos dentro del memento, permitiéndole restaurar su estado previo a voluntad.

En nuestro ejemplo del editor de texto, podemos crear una clase separada de historial que actúe como cuidadora. Una pila de mementos almacenados dentro de la cuidadora crecerá cada vez que el editor vaya a ejecutar una operación. Puedes incluso presentar esta pila dentro de la UI de la aplicación, mostrando a un usuario el historial de operaciones previamente realizadas.

Cuando un usuario activa la función Deshacer, el historial toma el memento más reciente de la pila y lo pasa de vuelta al editor, solicitando una restauración. Debido a que el editor tiene pleno acceso al memento, cambia su propio estado con los valores tomados del memento.

**Aplicabilidad**

 Utiliza el patrón Memento cuando quieras producir instantáneas del estado del objeto para poder restaurar un estado previo del objeto.

 El patrón Memento te permite realizar copias completas del estado de un objeto, incluyendo campos privados, y almacenarlos independientemente del objeto. Aunque la mayoría de la gente recuerda este patrón gracias al caso de la función Deshacer, también es indispensable a la hora de tratar con transacciones (por ejemplo, si debes volver atrás sobre un error en una operación).

 Utiliza el patrón cuando el acceso directo a los campos, consultores o modificadores del objeto viole su encapsulación.

 El Memento hace al propio objeto responsable de la creación de una instantánea de su estado. Ningún otro objeto puede leer la instantánea, lo que hace que los datos del estado del objeto original queden seguros.

**Pros**

* Puedes producir instantáneas del estado del objeto sin violar su encapsulación.
* Puedes simplificar el código de la originadora permitiendo que la cuidadora mantenga el historial del estado de la originadora.

## Contras

* La aplicación puede consumir mucha memoria RAM si los clientes crean mementos muy a menudo.
* Las cuidadoras deben rastrear el ciclo de vida de la originadora para poder destruir mementos obsoletos.
* La mayoría de los lenguajes de programación dinámicos, como PHP, Python y JavaScript, no pueden garantizar que el estado dentro del memento se mantenga intacto.