POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

19. *Memento*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

- *En la elaboración de un documento, cada vez que se guarda se pierde la posibilidad de "volver" a una versión anterior del documento.*
- *Por ese motivo, muchas personas recurren al "guardar como" para almacenar sucesivas versiones para tener un histórico de los distintos estados de la elaboración del documento y poder "volver" a cualquier estado anterior*

- *Sin violar la encapsulación, captura y externaliza el estado interno de un objeto de tal forma que el objeto puede ser restaurado a este estado después*

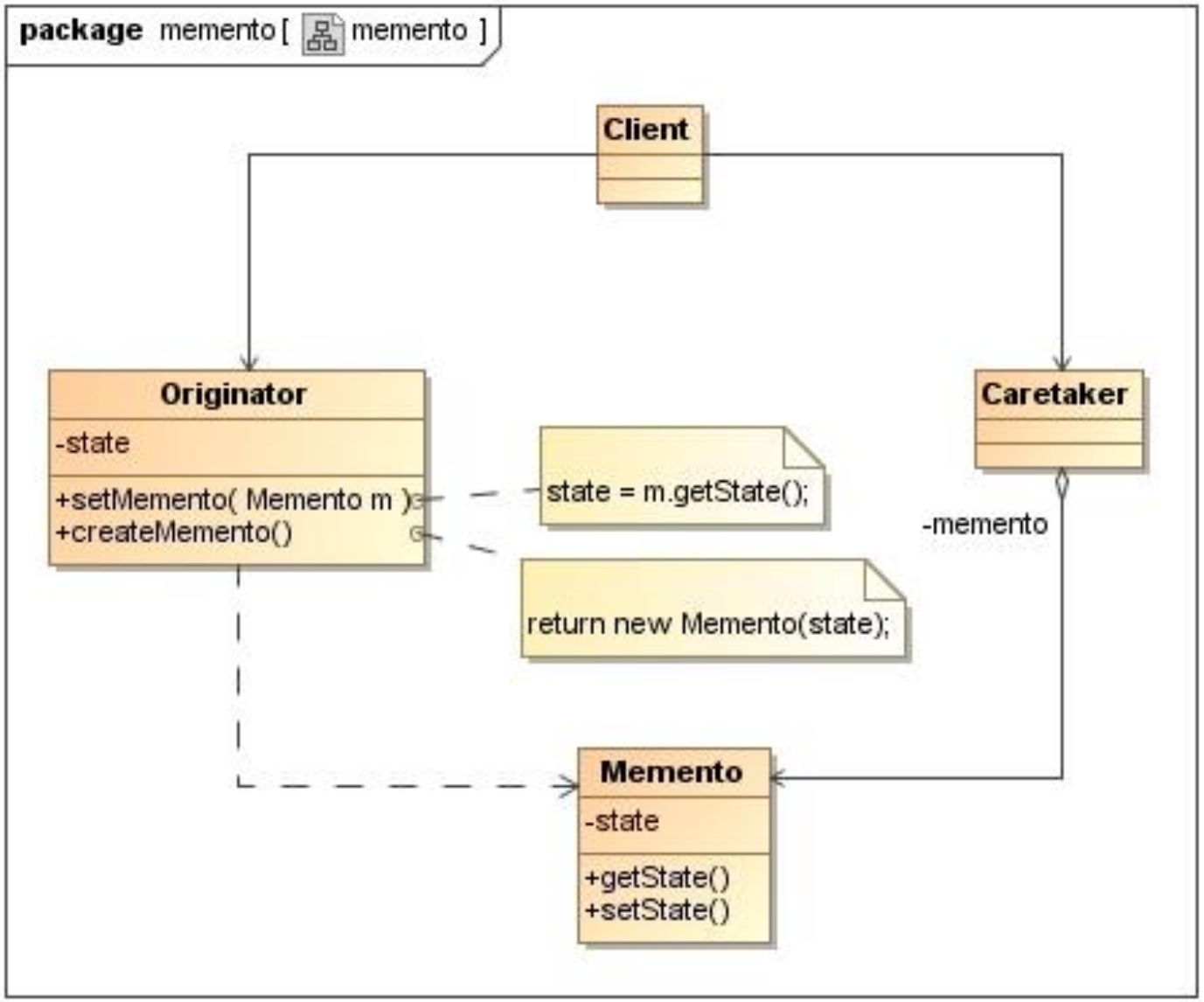
1. Problema

- A veces es necesario guardar el estado interno de un objeto. Esto es necesario cuando se implementan casillas de verificación o mecanismos de deshacer que permiten a los usuarios anular operaciones provisionales y recuperarse de los errores. Debe guardarse información del estado en algún sitio para que los objetos puedan volver a su estado anterior. Pero los objetos normalmente encapsulan parte de su estado, o todo, haciéndolo inaccesible a otros objetos e imposible de guardar externamente. Exponer este estado violará la encapsulación, lo que puede comprometer la fiabilidad y extensibilidad de la aplicación.

1. Problema

- Úsese cuando:
  - Hay que guardar una instancia del estado de un objeto (o de parte de éste) para que pueda volver posteriormente a ese estado, y
  - una interfaz directa para obtener el estado exponga detalles de implementación y rompa la encapsulación del objeto

2. Solución


package memento [memento]
Client
Originator
-state
+setMemento(Memento m)
+createMemento()
state = m.getState();
return new Memento(state);
Caretaker
-memento
Memento
-state
+getState()
+setState()

2. Solución

- Un *Memento* es un objeto que almacena una instancia del estado interno de otro objeto – el creador del *Memento* -. El mecanismo de deshacer solicitará un *Memento* al creador cuando necesite comprobar el estado de éste. El creador inicializa el *Memento* con información que representa su estado actual. Solo el creador puede almacenar y recuperar información del *Memento* – el *Memento* es "opaco" a otros objetos -.
- Los *Mementos* tienen dos interfaces: una amplia para los creadores y otra reducida para otros objetos. Lo ideal sería que el lenguaje de implementación permitiese dos niveles de protección estática. C++ permite hacer esto haciendo que Creador sea una clase amiga del *Memento* y haciendo privada la interfaz amplia del *Memento*. Solo la interfaz debería ser declarada pública

2. Solución

- Cuando los *Mementos* se crean y se devuelvan a su creador en una secuencia predecible, el *Memento* puede guardar únicamente el cambio con respecto al estado interno del creador.

3. Consecuencias

- El *Mememto* evita exponer información que sólo debería ser gestionada por un creador, pero que sin embargo debe ser guardada fuera del creador. El patrón oculta a otros objetos las interioridades, potencialmente complejas, del Creador, preservando así los límites de la encapsulación
- En otros diseños que persiguen conservar la encapsulación, el Creador mantiene las versiones de su estado interno que han sido solicitadas por los clientes. Eso asigna toda la responsabilidad de gestión del almacenamiento al Creador. Que sean los clientes quienes gestionen el estado que solicitan al Creador y evita que los clientes tengan que notificar a los creadores cuando han acabado

3. Consecuencias

- Los *Mementos* podrían producir un coste considerable si el Creador debe copiar grandes cantidades de información para guardarlas en el *Memento* o si los clientes crean y devuelven *Mementos* a su creador con mucha frecuencia. A menos que encapsular y restablecer el estado del Creador sea poco costoso, el patrón podría no ser apropiado
- En algunos lenguajes puede ser difícil garantizar que sólo el creador acceda al estado del *Memento*
- Un conserje es responsable de borrar los *Mementos* que custodia. Sin embargo, el conserje no sabe cuánto estado hay en un *Memento*. De ahí que un conserje que debería ser ligero pueda provocar grandes costes de almacenamiento cuando debe guardas *Mementos*.

4. Relaciones

- *Singleton* para que el gestor de cambios sea único y globalmente accesible

5. Comparativa

- ***Memento** vs **Command.***
  - Ver *Command vs Memento*