Memento Design Pattern

Intent

- Without violating encapsulation, capture and externalize an object's internal state so that the object can be returned to this state later.
- A magic cookie that encapsulates a "check point" capability.
- Promote undo or rollback to full object status.

Problem

Need to restore an object back to its previous state (e.g. "undo" or "rollback" operations).

Discussion

The client requests a Memento from the source object when it needs to checkpoint the source object's state. The source object initializes the Memento with a characterization of its state. The client is the "care-taker" of the Memento, but only the source object can store and retrieve information from the Memento (the Memento is "opaque" to the client and all other objects). If the client subsequently needs to "rollback" the source object's state, it hands the Memento back to the source object for reinstatement.

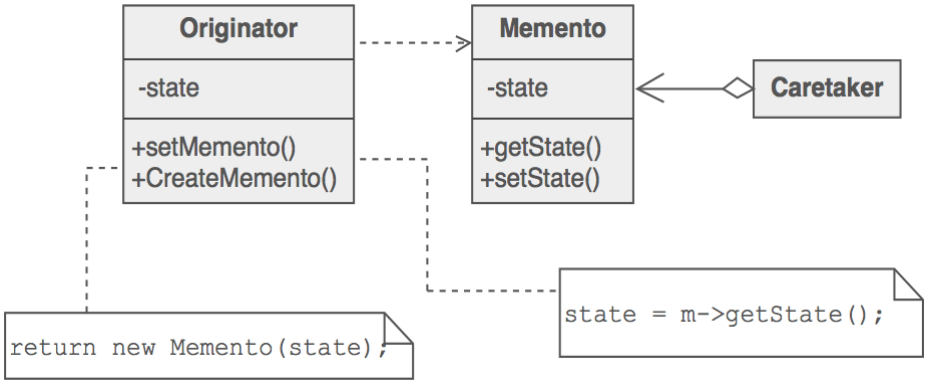
An unlimited "undo" and "redo" capability can be readily implemented with a stack of Command objects and a stack of Memento objects.

The Memento design pattern defines three distinct roles:

1. Originator - the object that knows how to save itself.
2. Caretaker - the object that knows why and when the Originator needs to save and restore itself.

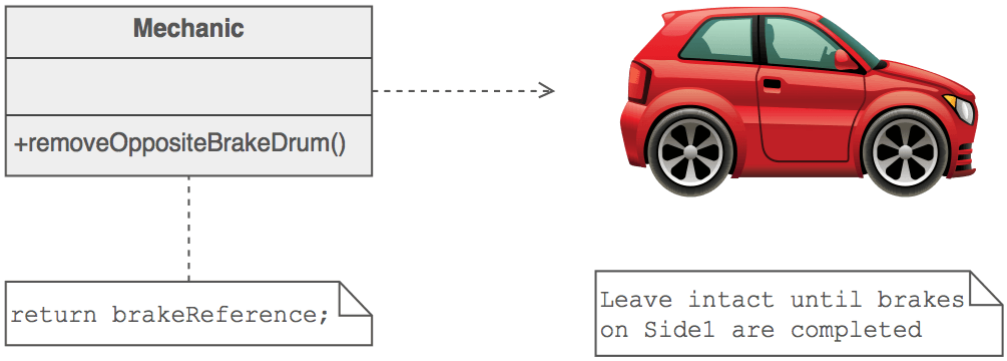
3. Memento - the lock box that is written and read by the Originator, and shepherded by the Caretaker.

Structure



Example

The Memento captures and externalizes an object's internal state so that the object can later be restored to that state. This pattern is common among do-it-yourself mechanics repairing drum brakes on their cars. The drums are removed from both sides, exposing both the right and left brakes. Only one side is disassembled and the other serves as a Memento of how the brake parts fit together. Only after the job has been completed on one side is the other side disassembled. When the second side is disassembled, the first side acts as the Memento.



Check list

1. Identify the roles of “caretaker” and “originator”.
2. Create a Memento class and declare the originator a friend.
3. Caretaker knows when to "check point" the originator.
4. Originator creates a Memento and copies its state to that Memento.
5. Caretaker holds on to (but cannot peek into) the Memento.
6. Caretaker knows when to "roll back" the originator.
7. Originator reinstates itself using the saved state in the Memento.

Rules of thumb

- Command and Memento act as magic tokens to be passed around and invoked at a later time. In Command, the token represents a request; in Memento, it represents the internal state of an object at a particular time. Polymorphism is important to Command, but not to Memento because its interface is so narrow that a memento can only be passed as a value.
- Command can use Memento to maintain the state required for an undo operation.
- Memento is often used in conjunction with Iterator. An Iterator can use a Memento to capture the state of an iteration. The Iterator stores the Memento internally.

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.

Learn more

Code examples

Java	**Memento in Java**	
C++	**Memento in C++**	
PHP	**Memento in PHP**	
Delphi	**Memento in Delphi**	**Memento in Delphi**
Python	**Memento in Python**	

READ NEXT

Null Object

RETURN

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us