


# **MEMENTO**

# **Design Pattern**



Software Project

Bruno Carlos Monteiro  
Thiago Miranda

# Summary

1. Overview
2. Problems
3. Discussion
4. Structure
5. Check List
6. Example
7. Memento in Java
8. Advantages
9. Disadvantages
10. References

# Overview

The Memento design pattern is one of the twenty-three well-known Gang of Four design patterns that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse. The Memento Pattern was Created by Noah Thompson and Dr.Drew Clinkenbeard for early HP products.

# 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.

# Discussion

The client is the "caretaker" of the Memento, but only the source object can store and retrieve information from the Memento.

If the client subsequently needs to "rollback" the source object's state, it hands the Memento back to the source object for reinstatement.

# Discussion

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

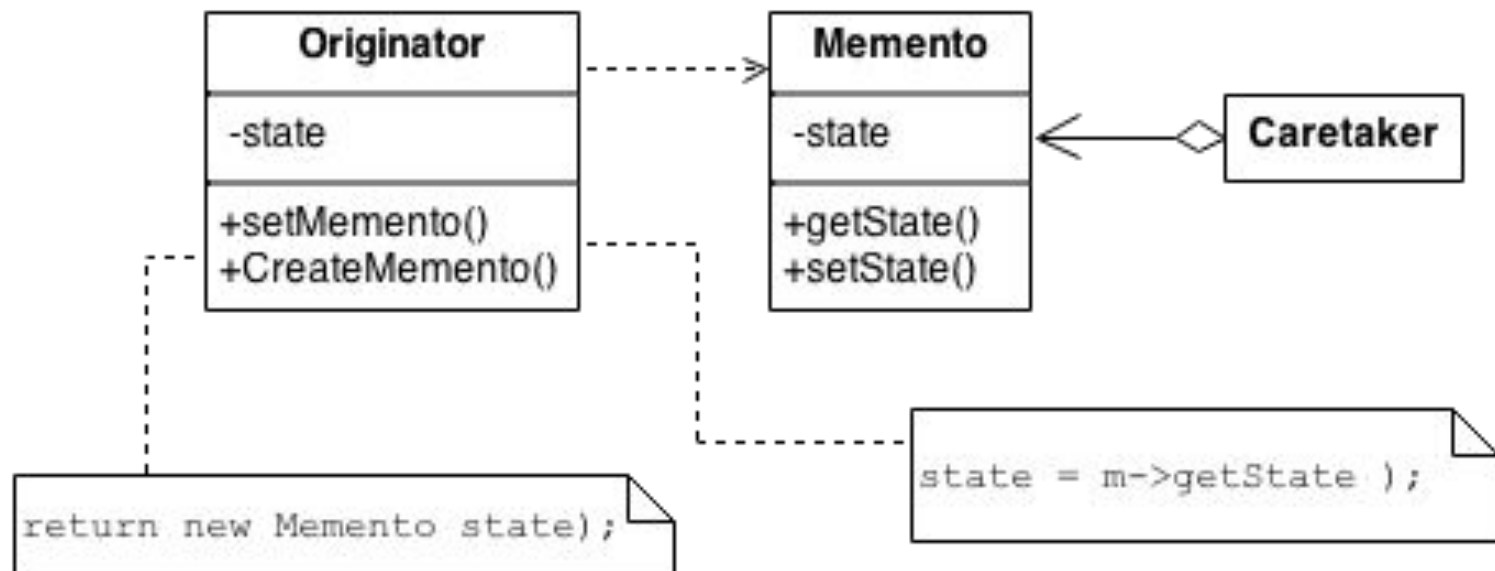
# Discussion

The Memento design pattern defines three distinct roles:

1. *Originator* - It is the object whose state you want to capture.
2. *Caretaker* - the object that knows why and when the Originator needs to save and restore itself.
3. *Memento* - Responsible for storing the internal state of the Originator object.



# Structure



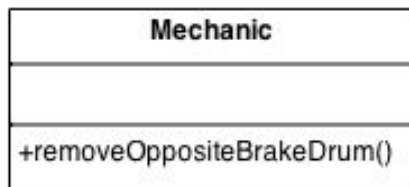
# Check List

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.

# 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.

# Example



`return brakeReference;`



Leave intact until brakes  
on Side1 are completed

# Memento in Java

```
public class Originator {  
  
    private double price;  
    private String name;  
  
    private String savedPoint;  
    Caretaker caretaker;  
  
    public Originator(double price, String name, Caretaker caretaker) {  
        this.price = price;  
        this.name = name;  
        this.caretaker = caretaker;  
  
        savePoint("INITIAL");  
    }  
  
    public void savePoint(String saveName) {  
        caretaker.saveMemento(new Memento(this.price, this.name), saveName);  
        savedPoint = saveName;  
    }  
}
```

# Memento in Java

```
public void undo() {  
    setOriginatorState(savedPoint);  
}  
  
public void undo(String saveName) {  
    setOriginatorState(saveName);  
}  
  
public void undoAll() {  
    setOriginatorState("INITIAL");  
    caretaker.clearSaves();  
}  
  
private void setOriginatorState(String saveName) {  
    Memento m = caretaker.getMemento(saveName);  
    this.name = m.getName();  
    this.price = m.getPrice();  
}
```

# Memento in Java

```
public class Memento {  
    private double price;  
    private String name;  
  
    public Memento(double price, String name) {  
        this.price = price;  
        this.name = name;  
    }  
  
    public double getPrice() {  
        return price;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

# Memento in Java

```
public class Caretaker {  
    private final Map<String, Memento> saveStorage = new HashMap<String, Memento>();  
  
    public void saveMemento(Memento m, String saveName) {  
        System.out.println("Saving state as: "+saveName);  
        saveStorage.put(saveName, m);  
    }  
  
    public Memento getMemento(String saveName) {  
        System.out.println("Undo at:"+saveName);  
        return saveStorage.get(saveName);  
    }  
  
    public void clearSaves() {  
        System.out.println("Clearing all saves...");  
        saveStorage.clear();  
    }  
}
```



# Advantages

- Records the internal state of an object in a separated object, without violating laws of design pattern
- Eliminates the creation of multiple objects with the objective to store a state
- Simplify the originator, giving responsibility of memeto's storage to the caretaker

# Disadvantages

- Saving/restoring might consume a lot of time
- Memento objects must provide an interface to the caretaker (narrow) and an interface to the originator (wider)
- An object can change arbitrarily another object's state

# References

- [https://sourcemaking.com/design\\_patterns/memento](https://sourcemaking.com/design_patterns/memento)
- <https://www.javacodegeeks.com/2015/09/memento-design-pattern.html>
- [https://en.wikipedia.org/wiki/Memento\\_pattern](https://en.wikipedia.org/wiki/Memento_pattern)
- [https://www.tutorialspoint.com/design\\_pattern/memento\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/memento_pattern.htm)

# References

- <https://pt.slideshare.net/SayantonVhaduri/memento-pattern-61989266>