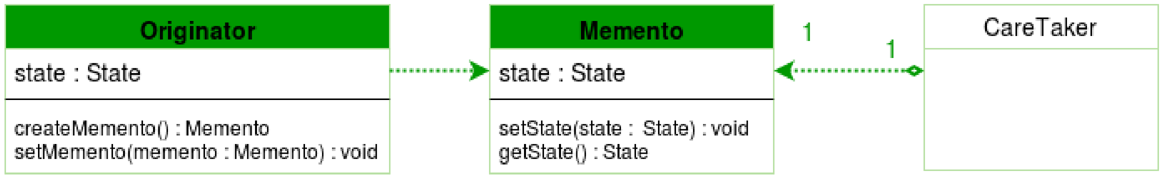
Memento design pattern

Memento pattern is a behavioral design pattern. Memento pattern is used to restore state of an object to a previous state. As your application is progressing, you may want to save checkpoints in your application and restore back to those checkpoints later.

UML Diagram Memento design pattern


Originator
state : State
createMemento() : Memento
setMemento(memento : Memento) : void
Memento
state : State
setState(state : State) : void
getState() : State
1
1
CareTaker

Design components

- **originator :** the object for which the state is to be saved. It creates the memento and uses it in future to undo.
- **memento :** the object that is going to maintain the state of originator. Its just a POJO.
- **caretaker :** the object that keeps track of multiple memento. Like maintaining savepoints.

A **Caretaker** would like to perform an operation on the **Originator** while having the possibility to rollback. The caretaker calls the **createMemento()** method on the originator asking the originator to pass it a memento object. At this point the originator creates a memento object saving its internal state and passes the memento to the caretaker. The caretaker maintains the memento object and performs the operation. In case of the need to undo the operation, the caretaker calls the **setMemento()** method on the originat passing the maintained memento object. The originator would accept the memento, using it to restore its previous state.

Let's see an example of Memento design pattern.

```
import java.util.List;
import java.util.ArrayList;

class Life
{
    private String time;

    public void set(String time)
    {
        System.out.println("Setting time to " + time);
        this.time = time;
    }

    public Memento saveToMemento()
    {
        System.out.println("Saving time to Memento");
        return new Memento(time);
    }

    public void restoreFromMemento(Memento memento)
    {
        time = memento.getSavedTime();
        System.out.println("Time restored from Memento: " + time);
    }

    public static class Memento
    {
        private final String time;

        public Memento(String timeToSave)
        {
            time = timeToSave;
        }
```

```
            public String getSavedTime()
            {
                return time;
            }
        }
    }

    class Design
    {

        public static void main(String[] args)
        {

            List<Life.Memento> savedTimes = new ArrayList<Life.Memento>();

            Life life = new Life();

            //time travel and record the eras
            life.set("1000 B.C.");
            savedTimes.add(life.saveToMemento());
            life.set("1000 A.D.");
            savedTimes.add(life.saveToMemento());
            life.set("2000 A.D.");
            savedTimes.add(life.saveToMemento());
            life.set("4000 A.D.");

            life.restoreFromMemento(savedTimes.get(0));

        }
    }
```

Output:

```
Setting time to 1000 B.C.
Saving time to Memento
Setting time to 1000 A.D.
Saving time to Memento
Setting time to 2000 A.D.
```

```
Saving time to Memento
Setting time to 4000 A.D.
Time restored from Memento: 1000 B.C.
```

Advantage

- We can use Serialization to achieve memento pattern implementation that is more generic rather than Memento pattern where every object needs to have it’s own Memento class implementation.

Disadvantage

- If Originator object is very huge then Memento object size will also be huge and use a lot of memory.

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Design Patterns | Set 1 (Introduction)

Design Patterns | Set 2 (Factory Method)

Command Pattern

Observer Pattern | Set 1 (Introduction)

Observer Pattern | Set 2 (Implementation)

Singleton Design Pattern | Implementation

Decorator Pattern | Set 1 (Background)

The Decorator Pattern | Set 2 (Introduction and Design)

Decorator Pattern | Set 3 (Coding the Design)

Strategy Pattern | Set 1 (Introduction)

Strategy Pattern | Set 2 (Implementation)

Adapter Pattern

Iterator Pattern

Curiously recurring template pattern (CRTP)

Flyweight Design Pattern

Article Tags : Design Pattern

Be the First to upvote.

☐ To-do ☐ Done

2

Based on **2** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

▲