Memento Design Pattern in Java

APRIL 2, 2018 BY PANKAJ — 13 COMMENTS

Memento design pattern is one of the behavioral design pattern. Memento design pattern is used when we want to save the state of an object so that we can restore later on. Memento pattern is used to implement this in such a way that the saved state data of the object is not accessible outside of the object, this protects the integrity of saved state data.

Table of Contents [hide]

Memento Design Pattern

Memento design pattern is implemented with two objects – `Originator` and `Caretaker`.

Originator is the object whose state needs to be saved and restored and it uses an inner class to save the state of Object. The inner class is called **Memento** and it's private, so that it can't be accessed from other objects.

Caretaker is the helper class that is responsible for storing and restoring the Originator's state through Memento object. Since Memento is private to Originator, Caretaker can't access it and it's stored as an Object within the caretaker.

Memento Design Pattern Java

One of the best real life example is the text editors where we can save it's data anytime and use undo to restore it to previous saved state.

We will implement the same feature and provide a utility where we can write and save contents to a File anytime and we can restore it to last saved state. For simplicity, I will not use any IO operations to write data into file.

Memento Pattern Originator Class

```
package com.journaldev.design.memento;

public class FileWriterUtil {
```

```
        private String fileName;
        private StringBuilder content;

        public FileWriterUtil(String file){
                this.fileName=file;
                this.content=new StringBuilder();
        }

        @Override
        public String toString(){
                return this.content.toString();
        }

        public void write(String str){
                content.append(str);
        }

        public Memento save(){
```

Notice the Memento inner class and implementation of save and undo methods. Now we can continue to implement Caretaker class.

Memento Pattern Caretaker Class

```
package com.journaldev.design.memento;

public class FileWriterCaretaker {

        private Object obj;

        public void save(FileWriterUtil fileWriter){
                this.obj=fileWriter.save();
        }

        public void undo(FileWriterUtil fileWriter){
                fileWriter.undoToLastSave(obj);
        }
}
```

Notice that caretaker object contains the saved state in the form of Object, so it can't alter its data and also it has no knowledge of it's structure.

Memento Pattern Example Test Class

Lets write a simple test program that will use our memento pattern implementation.

```
package com.journaldev.design.memento;

public class FileWriterClient {

	public static void main(String[] args) {

		FileWriterCaretaker caretaker = new FileWriterCaretaker();

		FileWriterUtil fileWriter = new FileWriterUtil("data.txt");
		fileWriter.write("First Set of Data\n");
		System.out.println(fileWriter+"\n\n");

		// lets save the file
		caretaker.save(fileWriter);
		//now write something else
		fileWriter.write("Second Set of Data\n");

		//checking file contents
		System.out.println(fileWriter+"\n\n");

		//lets undo to last save
		caretaker.undo(fileWriter);
```

Output of above memento pattern example test program is:

```
First Set of Data



First Set of Data
Second Set of Data



First Set of Data
```

Memento pattern is simple and easy to implement, one of the thing needs to take care is that Memento class should be accessible only to the Originator object. Also in client application, we should use caretaker object for saving and restoring the originator state.

Also if Originator object has properties that are not immutable, we should use deep copy or cloning to avoid data integrity issue like I have used in above example. 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.

One of the drawback is that if Originator object is very huge then Memento object size will also be huge and use a lot of memory.

« PREVIOUS
Mediator Design Pattern in Java

NEXT »
Observer Design Pattern in Java

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DESIGN PATTERNS

Comments

Prasanti says
APRIL 1, 2018 AT 9:51 PM
All patterns have been nicely explained.
Reply

ali says
APRIL 19, 2016 AT 4:57 AM
i cant understand one thing here
why we need to impl memento pattern why we dont use a tempral vars
like list or stack objects directly
Reply

John says
MAY 4, 2016 AT 3:07 PM
Maybe you missed the title of the article….
Reply

Chirag Sharma says
MARCH 31, 2016 AT 11:06 PM
Dear Pankaj
I thing i want to know that why we have save public method in FileWriteruntil because client can call this save method from writer object and after that state maintain logic will be fail.
Reply

Sachidananda Patra says
SEPTEMBER 26, 2015 AT 9:04 AM
Nice article !!
Reply

Rahul Sathe says
SEPTEMBER 16, 2015 AT 8:32 PM
Great article.
Just a question – Shouldn't the caretaker class have a stack kind of functionality to save the memento objects?
Reply

suvendu says
DECEMBER 11, 2014 AT 6:56 AM
Thanks!!
Reply

Dhemaz Sangcap says
JULY 21, 2014 AT 6:23 AM
great article. your example look different among others by making the Memento an inner class. Is it still acceptable to expose it as public class? and also is it valid without using interfaces? based on other tutorials they are creating 2 interfaces (for orginator. with access on mementos properties, and for care taker with no access on memento's properties.
Reply

johny says
DECEMBER 11, 2013 AT 9:35 PM
I have a doubt. Why do we need FileWriterCaretaker.java class. Cant we acheive the same functionality without this class. Say if I modify the FileWriterUtil.java with the following:
private Memento memento;
public void save(){
memento = new Memento(this.fileName,this.content);
}
public void undoToLastSave(){
if (memento != null) {
this.fileName= memento.fileName;
this.content=memento.content;
}
}
Anything wrong. Can you please let me know if this is valid?
Reply

Pankaj says
DECEMBER 12, 2013 AT 2:05 AM
Its done to implement the "separation of concerns". Also client code can choose not to have the undo feature by not using the caretaker class.
Reply

johny says
DECEMBER 12, 2013 AT 3:07 PM
Understood. Thanks.
Reply

sathish says
OCTOBER 1, 2015 AT 3:14 AM
Hi,
Could you please explain on the concerns that you separated in your code.
Reply

johny says
DECEMBER 11, 2013 AT 8:39 PM
Nice Article.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *
Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

DESIGN PATTERNS TUTORIAL

Java Design Patterns

Creational Design Patterns

- Singleton
- Factory
- Abstract Factory
- Builder
- Prototype

Structural Design Patterns

- Adapter
- Composite
- Proxy
- Flyweight
- Facade
- Bridge
- Decorator

Behavioral Design Patterns

- Template Method
- Mediator
- Chain of Responsibility
- Observer
- Strategy
- Command
- State
- Visitor
- Interpreter
- Iterator
- Memento

Miscellaneous Design Patterns

- Dependency Injection
- Thread Safety in Java Singleton

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial