26 Behavioral: Memento Pattern — Text-Editor Undo Assignment

Implement undo/redo for a simple text editor using the **Memento pattern**.  
 All classes must carry brief JavaDoc. Conclude with a concise **Reflection** on pros / cons.

.

├── analysis

│ └── memento\_need.md ← why snapshots > manual field caching

├── src/main/java

│ └── editor

│ ├── memento

│ │ └── TextEditorMemento.java

│ ├── originator

│ │ └── TextEditor.java

│ ├── caretaker

│ │ └── History.java

│ └── MementoDemo.java

├── src/test/java/editor

│ ├── UndoSingleStepTest.java

│ ├── UndoMultiStepTest.java

│ └── EmptyHistoryTest.java

├── reflection.md

└── README.md

#### **1 TextEditorMemento.java**

package editor.memento;

/\*\*

\* <p>Immutable snapshot of a {@code TextEditor}.</p>

\* Only the originator can create instances; the caretaker stores them blindly.

\*/

public final class TextEditorMemento {

private final String content;

TextEditorMemento(String content){ this.content = content; }

String getContent(){ return content; } // package-private

}

#### **2 TextEditor.java (Originator)**

package editor.originator;

import editor.memento.TextEditorMemento;

/\*\* Originator that can create and restore textual snapshots. \*/

public class TextEditor {

private String content = "";

/\*\* Replaces the entire document text. \*/

public void setContent(String text){ this.content = text; }

public String getContent(){ return content; }

/\*\* Creates a memento capturing current state. \*/

public TextEditorMemento save(){ return new TextEditorMemento(content); }

/\*\* Restores state from a previously created memento. \*/

public void restore(TextEditorMemento m){ this.content = m.getContent(); }

}

#### **3 History.java (Caretaker)**

package editor.caretaker;

import editor.memento.TextEditorMemento;

import java.util.ArrayDeque;

import java.util.Deque;

/\*\* Caretaker that stores editor snapshots in LIFO order. \*/

public class History {

private final Deque<TextEditorMemento> stack = new ArrayDeque<>();

public void push(TextEditorMemento m){ stack.push(m); }

/\*\* Returns the most-recent snapshot or {@code null} if history empty. \*/

public TextEditorMemento pop(){ return stack.poll(); }

}

#### **4 MementoDemo.java (Client)**

package editor;

import editor.originator.TextEditor;

import editor.caretaker.History;

/\*\* Demonstrates undo with the Memento pattern. \*/

public class MementoDemo {

public static void main(String[] args){

TextEditor editor = new TextEditor();

History history = new History();

editor.setContent("Version 1");

history.push(editor.save());

editor.setContent("Version 2");

history.push(editor.save());

editor.setContent("Version 3");

System.out.println("Now: " + editor.getContent()); // Version 3

editor.restore(history.pop());

System.out.println("Undo: " + editor.getContent()); // Version 2

editor.restore(history.pop());

System.out.println("Undo: " + editor.getContent()); // Version 1

}

}

Console

Now: Version 3

Undo: Version 2

Undo: Version 1

#### **5 JUnit-style snippets**

/\* UndoSingleStepTest.java \*/

TextEditor e = new TextEditor();

History h = new History();

e.setContent("A"); h.push(e.save());

e.setContent("B");

e.restore(h.pop());

assertEquals("A", e.getContent());

/\* EmptyHistoryTest.java \*/

assertNull(new History().pop());

## **reflection.md**

The Memento pattern cleanly **captures internal state** without exposing fields to outside code.

| **Benefit** | **Note** |
| --- | --- |
| Encapsulation retained | TextEditorMemento exposes no setters; only the originator can touch stored data. |
| Simple undo/redo | A stack of mementos gives instant, reversible edits. |
| Check-pointing | Same snapshots could implement “Save draft” or rollback after error. |

*Trade-offs*

* **Memory** – every version stores full text; for large docs use diff-based mementos or cap history.
* **Class overhead** – extra caretaker and memento classes for each originator.
* **Granularity** – deciding when to snapshot (every keystroke? on save?) affects performance and UX.

For editors, games, or workflows where state is lightweight yet valuable to reverse, Memento offers a tidy, maintainable solution.