Memento Design Pattern

The Memento Pattern is a behavioral design pattern that allows you to capture and externalize an object's internal state without violating encapsulation, so that the object can be restored to this state later. This pattern is particularly useful for implementing undo mechanisms.

Use Case:

Imagine you are developing a text editor where users can type text and undo their changes. You need a way to save the state of the text at different points in time, so that the user can revert to a previous state if needed.

Components:

1. Originator (TextEditor): Creates a memento containing a snapshot of its current internal state and uses the memento to restore its state.
2. Memento (TextMemento): Stores the internal state of the originator object at a specific moment in time. The memento is immutable and does not expose its state to other objects.
3. Caretaker (TextEditorCaretaker): Manages the mementos and is responsible for storing and restoring the originator's state from the memento.

Example: Text Editor with Undo Functionality

1. Originator (TextEditor):

```java
public class TextEditor {
    private StringBuilder text;

    public TextEditor() {
        this.text = new StringBuilder();
    }
```

```
    public void write(String newText) {
        text.append(newText);
    }

    public String getText() {
        return text.toString();
    }

    public TextMemento save() {
        return new TextMemento(text.toString());
    }

    public void restore(TextMemento memento) {
        this.text = new StringBuilder(memento.getText());
    }
}
```

2. Memento (TextMemento):

```java
public class TextMemento {
    private final String text;

    public TextMemento(String text) {
        this.text = text;
    }
```

```
    public String getText() {
        return text;
    }
}
```

3. Caretaker (TextEditorCaretaker):

```java
import java.util.Stack;

public class TextEditorCaretaker {
    private Stack<TextMemento> history = new Stack<>();

    public void save(TextEditor editor) {
        history.push(editor.save());
    }

    public void undo(TextEditor editor) {
        if (!history.isEmpty()) {
            editor.restore(history.pop());
        } else {
            System.out.println("No states to undo.");
        }
    }
}
```

4. Client Code:

```java
public class MementoPatternDemo {
    public static void main(String[] args) {
        TextEditor editor = new TextEditor();
        TextEditorCaretaker caretaker = new TextEditorCaretaker();

        editor.write("Hello, ");
        caretaker.save(editor);

        editor.write("World!");
        caretaker.save(editor);

        System.out.println("Current text: " + editor.getText());

        caretaker.undo(editor);
        System.out.println("After undo: " + editor.getText());

        caretaker.undo(editor);
        System.out.println("After another undo: " + editor.getText());
    }
}
```

Key Points:

- State Preservation: The Memento Pattern allows you to save and restore an object's state, making it ideal for implementing undo functionality.

- Encapsulation: The internal state of the object is preserved without exposing its details to other objects.

- Rollback Mechanism: This pattern is useful for scenarios where you need to rollback changes or revert to a previous state.