40. Memento

Memento is a design pattern that is used specifically for storing intermediate states of some data, so the actions can be undone and the state can be reverted.

The Memento design pattern can be summarized as follows:

- There is an **Originator** object, which acts as the origin of the data that can change its state at will.
- There is a **Memento** object, which represents a single snapshot of the data.
- **Originator** object can generate a new instance of **Memento** when some action is performed, or accept an instance of **Memento** to reset its data.
- There's also a **Caretaker** object, which stores a collection of **Memento** objects in a stack. This stack represents the history of changes.
- The **Caretaker** object also coordinates the history of changes with the **Originator** and allows the consumer to undo the actions.

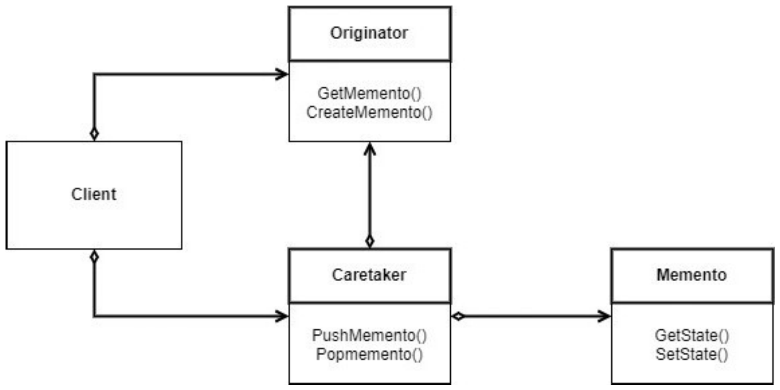


Figure 40.1 - Memento UML diagram

We will now go through an example implementation of Memento. The complete solution can be found via the link below:

https://github.com/fiodarsazanavets/design-patterns-in-csharp/tree/main/Behavioral_Patterns/Memento

Prerequisites

In order to be able to implement the code samples below, you need the following installed on your machine:

- .NET 8 SDK (or newer)
- A suitable IDE or a code editor (Visual Studio, Visual Studio Code, JetBrains Rider)

Memento implementation example

After creating a project based on a console application template, we will add the following interface to it, which will represent the public methods available on the **Memento** object:

```
namespace Memento_Demo;

internal interface IMemento
{
    string GetState();
    DateTimeOffset GetCreatedDate();
}
```

The concrete **Memento** implementation will look as follows:

```
namespace Memento_Demo;

internal class TextEditorMemento(
    string state) :  IMemento
{
        private readonly
        string state = state;
        private readonly
        DateTimeOffset created =
            DateTimeOffset.Now;

    public string GetState()
    {
                return state;
    }

        public DateTimeOffset GetCreatedDate()
    {
                return created;
    }
}
```

Because this object is only ever meant to represent an immutable snapshot, both its state and the timestamp are read-only. They only get populated via the class constructor.

We will now add our **Originator** object. This will be a class that represents a text editor. In this text editor, you can update text at any point. When you save text, a **Memento** object is generated. Also, at any point, the internal `state` can be restored to one of the previous snapshots by passing a **Memento** object into the `SetState` method.

```
namespace Memento_Demo;

internal class TextEditor
{
        private string state;

        public TextEditor()
        {
                state = string.Empty;
        }

        public string GetCurrentText()
    {
                return state;
    }

        public void UpdateText(
        string updatedText)
    {
                state = updatedText;
        }

        public IMemento Save()
    {
                Console.WriteLine(
            "Saving state.");
                return new TextEditorMemento(
            state);
    }

        public void SetState(
        IMemento memento)
        {
                state = memento.GetState();
                Console.WriteLine(
            $"Restored the state from the snapshot create\
d at {
                memento.GetCreatedDate()}.");
    }
}
```

Next, we will add a **Caretaker** object. This object stores the history of previous **Memento** objects on a stack and coordinates this history with the **Originator** object.

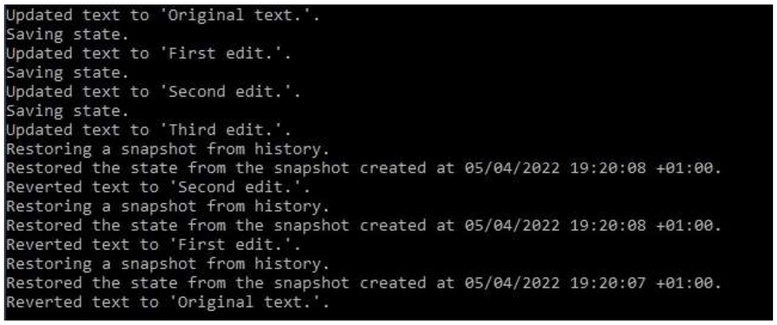
```
namespace Memento_Demo;

internal class Caretaker
{
        private TextEditor textEditor;
        private Stack<IMemento> history;

        public Caretaker(
        TextEditor textEditor)
        {
                this.textEditor =
            textEditor;
                history =
            new Stack<IMemento>();
        }

        public void Backup()
    {
```

```
                history.Push(
            textEditor.Save());
        }

        public void Revert()
    {
                Console.WriteLine(
            "Restoring a snapshot from history.");

                if (history.Count == 0)
        {
                        Console.WriteLine(
                "No snapshots to restore.");
                        return;
        }

                textEditor.SetState(history.Pop());
        }
}
```

Finally, we will replace the content of the `Program.cs` file with the following. We create some **Memento** snapshots and see what happens if we attempt to revert them.

```
using Memento_Demo;

var textEditor =
    new TextEditor();
var caretaker =
    new Caretaker(textEditor);

textEditor.UpdateText(
    "Original text.");
Console.WriteLine(
    $"Updated text to '{
        textEditor.GetCurrentText()}'.");
caretaker.Backup();
textEditor.UpdateText(
    "First edit.");
Console.WriteLine(
    $"Updated text to '{
        textEditor.GetCurrentText()}'.");
caretaker.Backup();
textEditor.UpdateText(
    "Second edit.");
Console.WriteLine(
    $"Updated text to '{
        textEditor.GetCurrentText()}'.");
caretaker.Backup();

textEditor.UpdateText(
    "Third edit.");
Console.WriteLine(
    $"Updated text to '{
        textEditor.GetCurrentText()}'.");

caretaker.Revert();
Console.WriteLine(
    $"Reverted text to '{
        textEditor.GetCurrentText()}'.");
caretaker.Revert();
Console.WriteLine(
    $"Reverted text to '{
        textEditor.GetCurrentText()}'.");
caretaker.Revert();
Console.WriteLine(
    $"Reverted text to '{
        textEditor.GetCurrentText()}'.");

Console.ReadKey();
```

And, as we can see from the screenshot below, we are able to revert the text in the text editor to its previous state.



```
Updated text to 'Original text.'.
Saving state.
Updated text to 'First edit.'.
Saving state.
Updated text to 'Second edit.'.
Saving state.
Updated text to 'Third edit.'.
Restoring a snapshot from history.
Restored the state from the snapshot created at 05/04/2022 19:20:08 +01:00.
Reverted text to 'Second edit.'.
Restoring a snapshot from history.
Restored the state from the snapshot created at 05/04/2022 19:20:08 +01:00.
Reverted text to 'First edit.'.
Restoring a snapshot from history.
Restored the state from the snapshot created at 05/04/2022 19:20:07 +01:00.
Reverted text to 'Original text.'.
```

Figure 40.2 - Undoing actions by using Memento

Now, let's summarize the benefits of using the Memento design pattern.

Benefits of using Memento

Memento design pattern gives you the following key benefits:

- Because snapshot **Memento** objects are separate from the internal state of the **Originator**, you can get them to store only as much or as little information as possible.
- The history of changes is easy to construct and revert.
- Because there is a separate **Caretaker** object involved, the single responsibility principle is well maintained.

But, as effective as the Memento design pattern is for enabling *undo* action, there are some things to look out for while using it. And this is what we will have a look at next.

Caveats of using Memento

The main thing to look out for while using the Memento design pattern is that your **Memento** stack doesn't consume too much memory, which can happen if the history grows too large. You can mitigate this by making sure that you only save those elements of the state that you definitely need to keep track of and nothing else. You can add some rules that would remove older entries in the history. Or perhaps you can even combine Memento with the Flyweight design pattern, saving common states in shared objects.

Also, in some situations, the **Originator** object may complete its lifecycle and get out of the scope of the program. In this case, the history of its snapshots will serve no purpose, while it will still be occupying the memory. To mitigate against this, you will need to keep track of the **Originator's** lifecycle inside the **Caretaker** object and clear the history when appropriate.