Memento Pattern

Design Patterns

Motivating Example

Memento Pattern

- It would be useful to rollback one or more objects within your application to a previous state (for example, to implement Undo)
- Providing Undo functionality within one or more classes would violate the Single Responsibility Principle
- Providing full access to objects' internal state violates the encapsulation principle
- The *memento pattern* describes a way to capture objects' internal state without violating encapsulation or SRP.

Intent

Memento Pattern

"Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later."

Design Patterns

The *memento* pattern involves two objects: the *originator* and the *caretaker*. The originator represents the object whose state is being tracked. The caretaker performs operations on the originator, but needs to be able to undo the operations.

Applicability

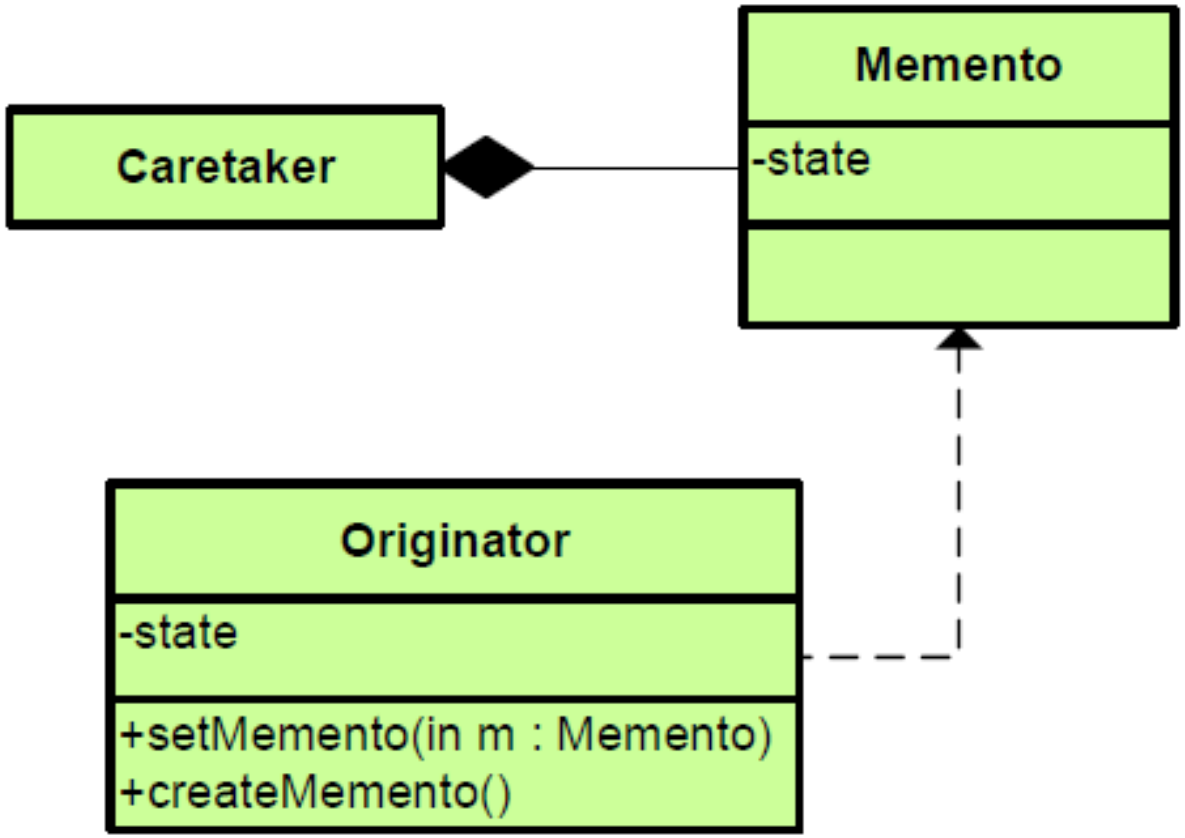
Memento Pattern

Use the Memento Pattern when:

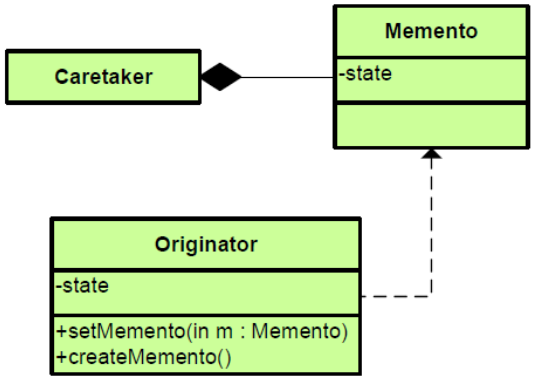
- **You need to be able to track the state of an object or your application and restore previous states as needed**
- **You want to abstract and reuse the Undo/Redo functionality**
  - Follow SRP and Don't Repeat Yourself (DRY)
  - Learn more in the Principles of Object Oriented Development course
- **You do not wish to break encapsulation and expose your classes' internal state globally**

Structure

Memento Pattern



How It Gets Used



- **The Memento itself simply holds the state of the Originator**
  - Ideally, only the originator should have access to the internal details of its state within the Memento
  - The Caretaker's interface to Memento should not allow it to access the internal state of the Originator
- **The Originator can create Mementos or restore its state given a Memento**
- **The Caretaker maintains the Memento objects without operating on their contents**

The Undo/Redo Stack

Memento Pattern

- **Multiple Undo/Redo is often implemented using a *Stack***
- **When a change is performed:**
  - A memento containing the current state is added to the Undo stack
- **When an Undo operation is performed:**
  - The current state is replaced with the state of the memento removed from the Undo stack
  - If Redo is implemented, the current state is stored in a Redo stack
- **When a Redo operation is performed:**
  - The current state is replaced with the state of the memento removed from the Redo stack
  - The current state is stored in the Undo stack

Collaboration

Memento Pattern

- A Caretaker requests a Memento from the Originator
- The Caretaker stores one or more Mementos until they are needed
- If or when required, the Caretaker passes a Memento back to the Originator.
- Mementos should be Value Objects, holding state but no behavior.
- Only the Originator that created a Memento should assign or retrieve its state

Consequences

Memento Pattern

- The Memento pattern shields other objects from potentially complex internal state of the Originator
- As opposed to the Originator maintaining versions of its internal state, the Memento pattern keeps Originator simpler
- Creating and restoring state may be expensive. Memento may not be appropriate when state contains a large amount of information.
- It may be difficult to ensure only the Originator can access a Memento's state
- Caretaker is responsible for managing Mementos, but has no idea how big or small they may be

An Alternative

Memento Pattern

- **Instead of storing state, store reverse operations**
  - May require significantly fewer resources
- **Calculator Example:**
  - Initial State: 10
  - Operation: +5
  - Reverse Operation: -5
  - New State: 15
  - Implement Undo by applying Reverse Operation
- **Many operations do not produce the same result when the reverse is applied.**
  - Graphic calculations
  - Translation

Translation Undo without State

Memento Pattern

Original Text:

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Translate English to German (undo with German to English operation):

Ohne Verletzung Kapselung, erfassen Sie und externalisieren Sie internen Zustand eines Objekts zu, so dass das Objekt später zu diesem Zustand wiederhergestellt werden kann.

Undo (German to English):

Without violating encapsulation, capture and externalize internal state of an object too, so that the object can be restored to this State later.

Iterative Memento

Memento Pattern

- Similar to storing each operation
- Store each iterative change in state, rather than complete state
- Example: Version control systems
  - Each check-in stores only the changes made to each file
- Requires fewer storage resources to maintain many state changes
- May require more resources to recreate complete state

Demo

Adding Undo to a WPF Paint Application

Related Patterns

Memento Pattern

- **Command**
  - Commands can use Mementos to provide Undo functionality
- **Iterator**
  - A memento can represent each iteration state

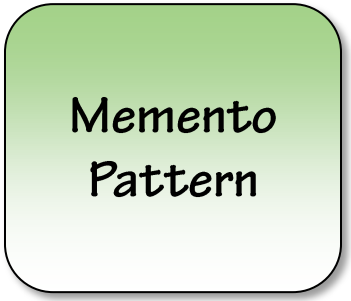
You can learn more about these patterns in the Pattern Library at PluralSight On Demand.

References

Memento Pattern

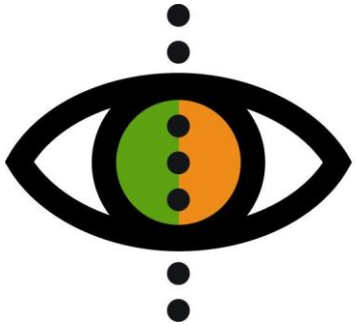
- **Books**
  - Design Patterns, http://amzn.to/95q9ux
  - Design Patterns in C#, http://amzn.to/bqJgdU
- **Online**
  - http://en.wikipedia.org/wiki/Memento_pattern
  - http://codeproject.com/Articles/186184/Memento-Design-Pattern.aspx

Summary



- The Memento pattern provides a simple way to manage different versions of a class
- Using Memento, you can implement Undo/Redo behavior without breaking encapsulation or violating SRP or DRY
- Try to avoid exposing a Memento's state to any object other than the Originator (the object whose state is represented)

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust

Blog: SteveSmithBlog.com

Twitter: @ardalis