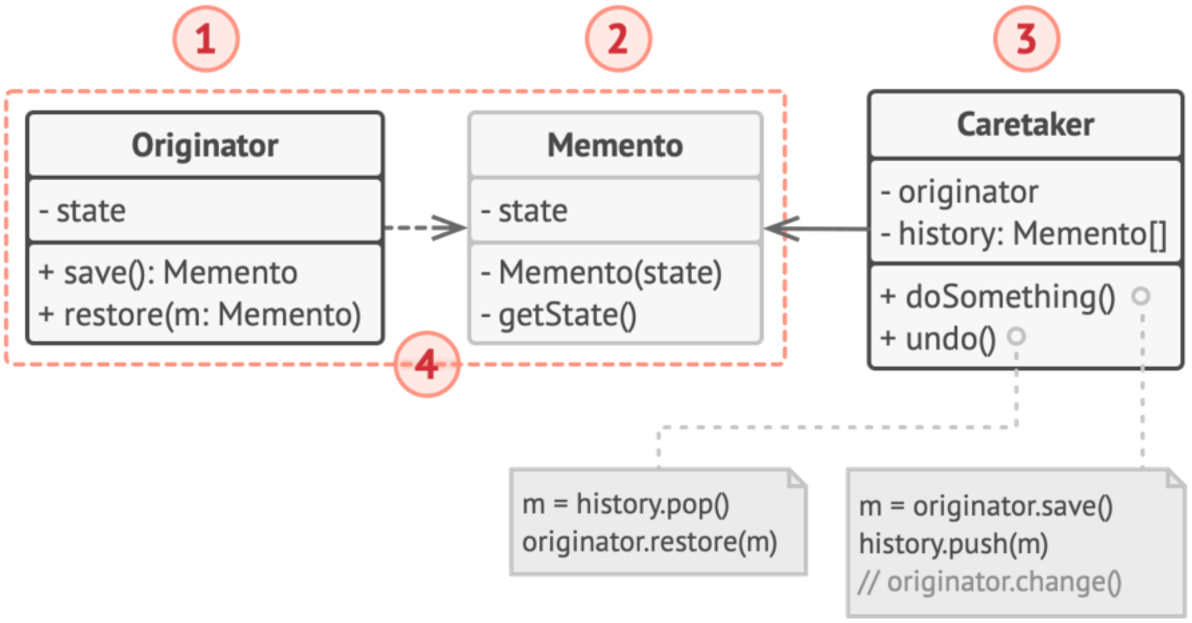
Intent

Memento is a behavioral design pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.

Structure

Implementation based on nested classes

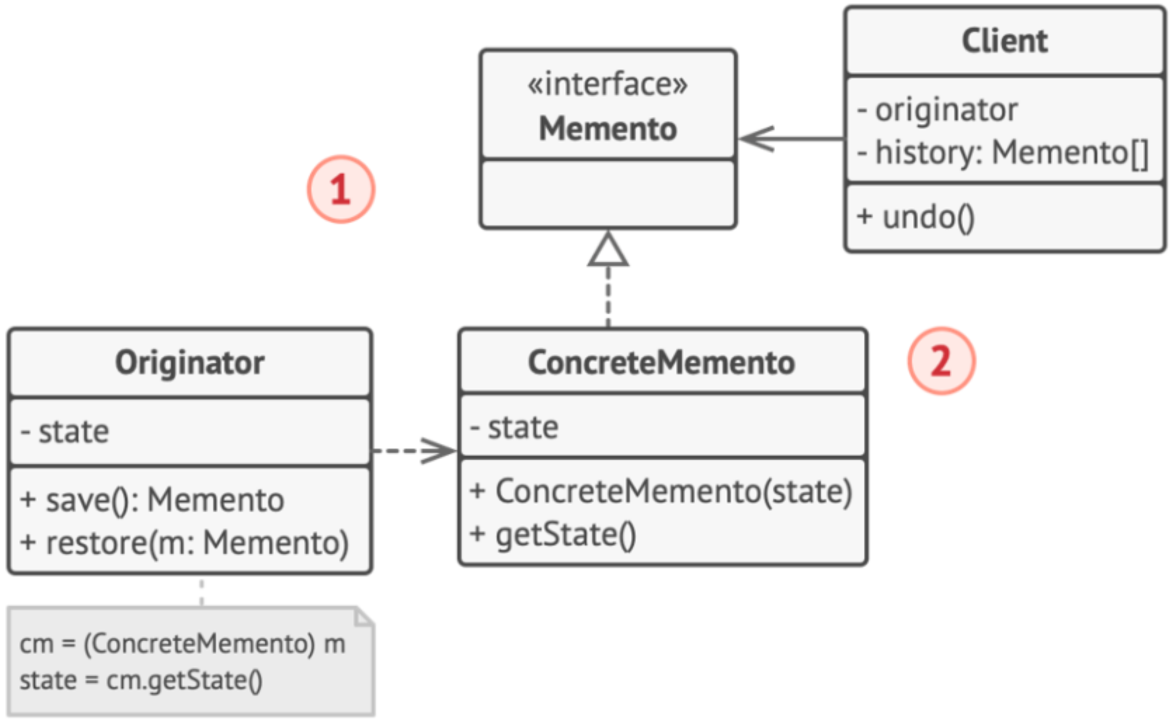


1. The **Originator** class can produce snapshots of its own state, as well as restore its state from snapshots when needed.
2. The **Memento** is a value object that acts as a snapshot of the originator's state. It's a common practice to make the memento immutable and pass it the data only once, via the constructor.
3. The **Caretaker** knows not only "when" and "why" to capture the originator's state, but also when the state should be restored. A caretaker can keep track of the originator's history by storing a stack of mementos. When the originator has to travel back in history, the caretaker fetches the topmost memento from the stack and passes it to the originator's restoration method.

4. In this implementation, the memento class is nested inside the originator. This lets the originator access the fields and methods of the memento, even though they’ re declared private. On the other hand, the caretaker has very limited access to the memento’ s fields and methods, which lets it store mementos in a stack but not tamper with their state.

Implementation based on an intermediate interface

some language does not support nested class



1. In the absence of nested classes, you can restrict access to the memento’ s fields by establishing a convention that caretakers can work with a memento only through an explicitly declared intermediary interface, which would only declare methods related to the memento’ s metadata.
2. On the other hand, originators can work with a memento object directly, accessing fields and methods declared in the memento class. The downside of this approach is that you need to declare all members of the memento public.

Applicability

- Use the Memento pattern when you want to produce snapshots of the object’s state to be able to restore a previous state of the object.
- Use the pattern when direct access to the object’s fields/getters/setters violates its encapsulation.