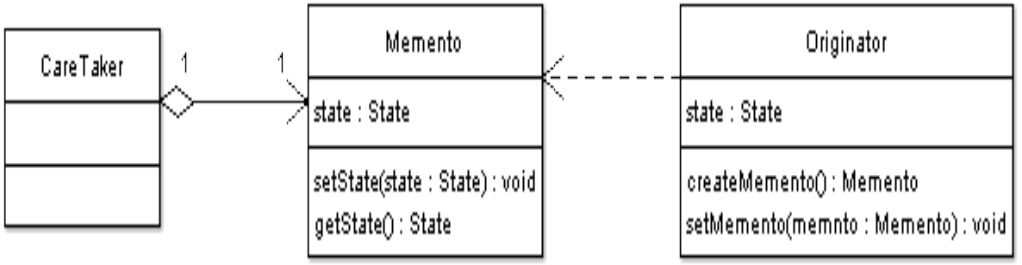
Memento Pattern

The intent of this pattern is to capture the internal state of an object without violating encapsulation and thus providing a means for restoring the object into initial state when needed. It provides an undo via rollback. Remember!

UML 1



•Memento
- Stores internal state of the Originator object. The state can include any number of state variables.
- The Memento must have two interfaces, an interface to the caretaker. This interface must not allow any operations or any access to internal state stored by the memento and thus honors encapsulation. The other interface is to the originator and allows the originator to access any state variables necessary to for the originator to restore previous state.

•Originator
- Creates a memento object capturing the originators internal state.
- Use the memento object to restore its previous state.

•Caretaker
- Responsible for keeping the memento.
- The memento is opaque to the caretaker, and the caretaker must not operate on it.

Caretaker

- A Caretaker would like to perform an operation on the Originator while having the possibility to rollback.
- The caretaker calls the createMemento() method on the originator asking the originator to pass it a memento object. At this point the originator creates a memento object saving its internal state and passes the memento to the caretaker.
- The caretaker maintains the memento object and performs the operation.
- In case of the need to undo the operation, the caretaker calls the setMemento() method on the originator passing the maintained memento object. The originator would accept the memento, using it to restore its previous state.

Java Example: Originator

```
import java.util.List;
import java.util.ArrayList;
class Originator {
    private String state;
    // The class could also contain additional data that is not part of the
    // state saved in the memento.

    public void set(String state) {
        System.out.println("Originator: Setting state to " + state);
        this.state = state;
    }

    public Memento saveToMemento() {
        System.out.println("Originator: Saving to Memento.");
        return new Memento(state);
    }

    public void restoreFromMemento(Memento memento) {
        state = memento.getSavedState();
        System.out.println("Originator: State after restoring from Memento: " + state);
    }
```

Java Example: Memento class

```
public static class Memento {
        private final String state;

        private Memento(String stateToSave) {
            state = stateToSave;
        }

        private String getSavedState() {
            return state;
        }
    }
}
```

Java Example: Caretaker

```
class Caretaker {
    public static void main(String[] args) {
        List<Originator.Memento> savedStates = new ArrayList<Originator.Memento>();

        Originator originator = new Originator();
        originator.set("State1");
        originator.set("State2");
        savedStates.add(originator.saveToMemento());
        originator.set("State3");
        // We can request multiple mementos, and choose which one to roll back to.
        savedStates.add(originator.saveToMemento());
        originator.set("State4");

        originator.restoreFromMemento(savedStates.get(1));
    }
}
```

Java Example: Output

Originator: Setting state to State1
Originator: Setting state to State2
Originator: Saving to Memento.
Originator: Setting state to State3
Originator: Saving to Memento.
Originator: Setting state to State4
Originator: State after restoring from Memento: State3

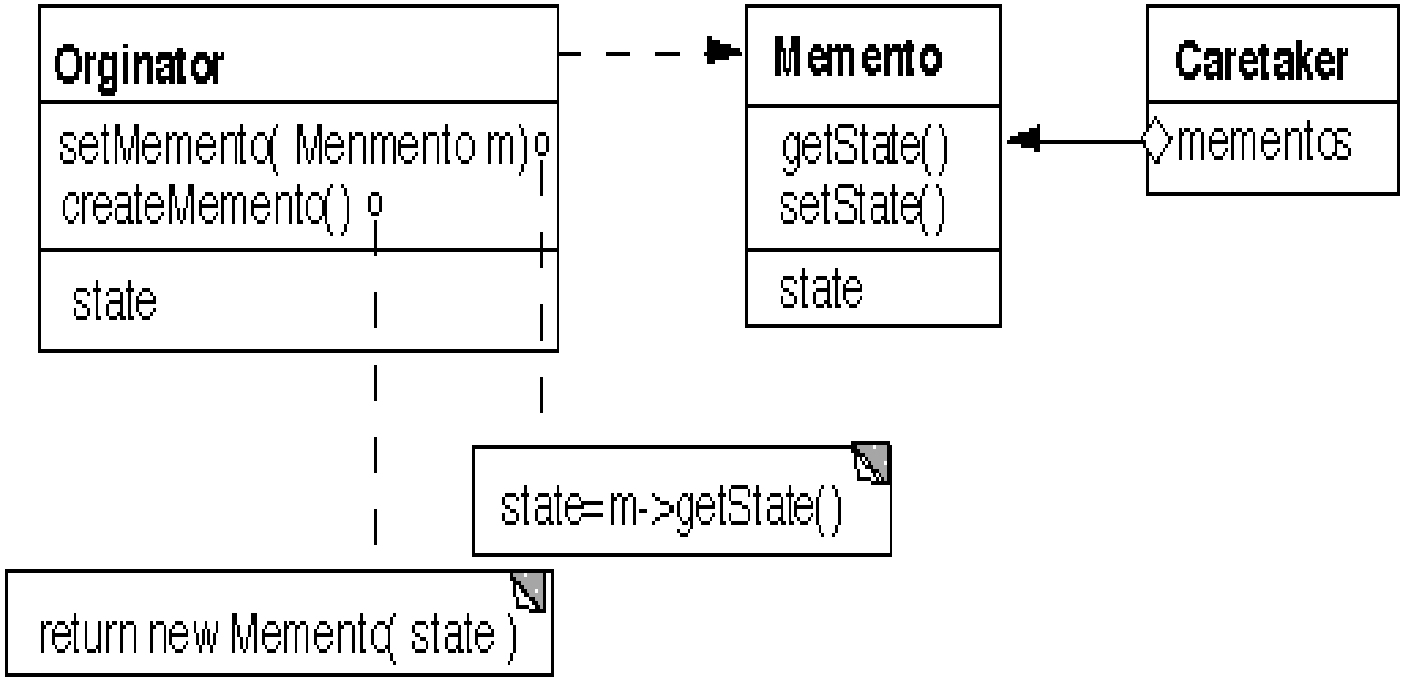
Memento

- The Memento object itself is an opaque object (one which the Caretaker cannot, or should not, change).
- When using this pattern, care should be taken if the originator may change other objects or resources - the Memento pattern operates on a single object.
- Memento is used to save **State**

Memento and Command

- Command can have an undo ability, but it involves executing something to achieve the undo
- Memento allows for reassigning State
- Memento in action: Recent Files list for an application
- Command in action in same application: Undo (Ctrl+Z) previous operation(s)
- Command pattern can use Memento to help remember state for a set of undo operations (can have a list of Mementos)

One Last UML


Orginator
setMemento(Menmento m)
createMemento()
state
Memento
getState()
setState()
state
Caretaker
mementos
state=m->getState()
return new Memento(state)

Final Thoughts

- Memento is a behavioral pattern
- Use when you:
  - Need to save all or part of the state of an object and
  - Do not wish to expose the saved state to the outside world
- Consequences
  - Simplifies Originator (It does not need to provide all the code to manage the saved state)
  - Copying state takes time and space, thus sometimes Memento is not appropriate