Memento Pattern

Chapter Content

InterBit
Training & Consulting Ltd.

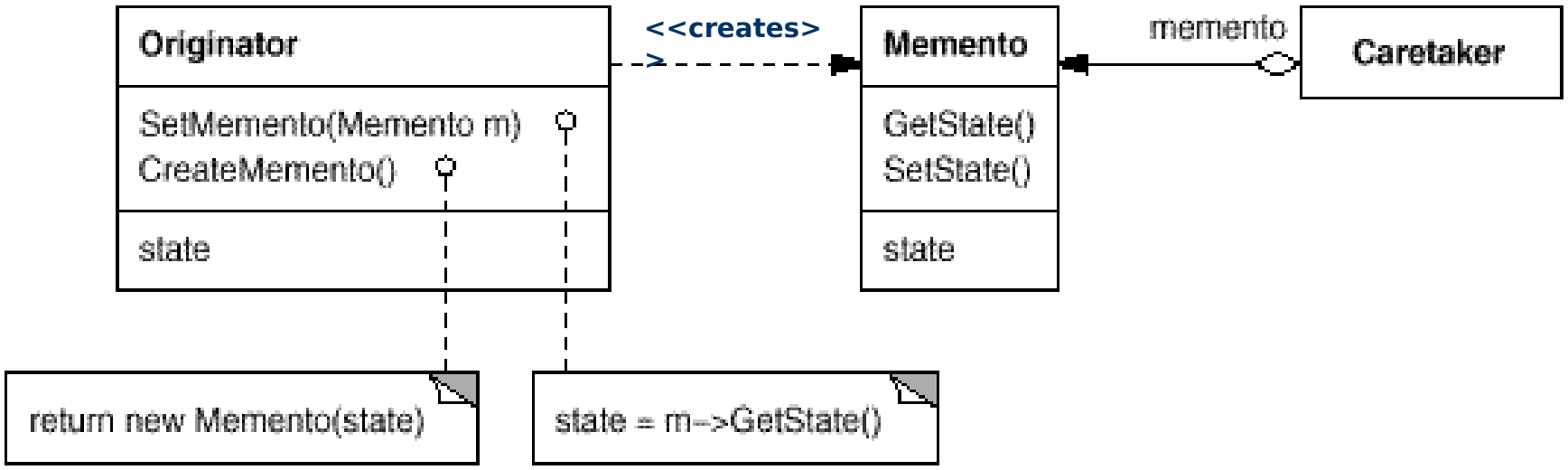
- Memento Pattern Overview
- Memento Pattern UML Diagram
- Memento Examples
- Access considerations
- Why not just clone ?

Memento Pattern Overview

InterBit Training & Consulting Ltd.

- Capture and externalize an object's internal state, so that the object can later be restored to this state.
  - Do so without violating encapsulation.
- Example: saving user preferences for a window
  - Size, which toolbars are visible…
  - Often saved to file & restored after re-booting.
  - Program manager will ask for a memento, thus hiding the window's internal data.

Memento Pattern UML Diagram



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. Uses the memento object to restore its previous state.**

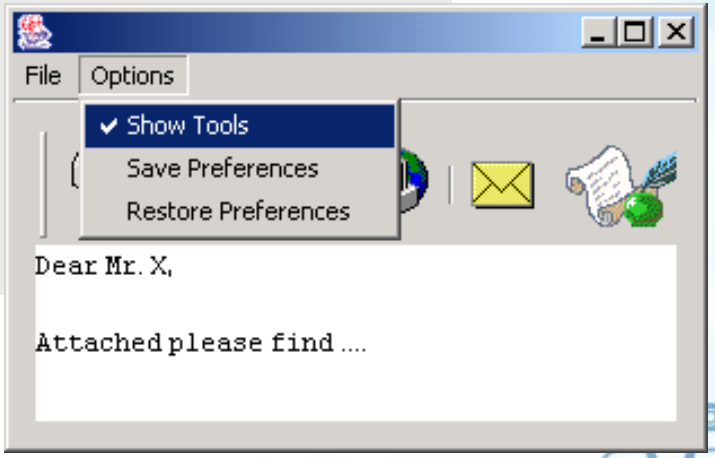
Caretaker - **Responsible for keeping the memento. The memento is opaque to the caretaker, and the caretaker must not operate on it.**

Memento Example

InterBit Training & Consulting Ltd.

```
public interface Memento {
}
```

```
public class MyWin extends JFrame{
    private JToolBar bar ;
    ...
    public MyWin(){
        ...
        restore(new MyWinMemento(DEFAULT_LOCATION, DEFAULT_SIZE, true));
    }
    private class MyWinMemento implements Memento {
        Point location;
        Dimension size;
        boolean barVisible;
    }
```



Memento Example (cont.)

InterBit Training & Consulting Ltd.

```
        public Memento getMemento(){
                return new MyWinMemento(getLocation(),getSize(),bar.isVisible());
        }
        public void restore(Memento memento){
                MyWinMemento wm=(MyWinMemento)memento;
                setLocation(wm.location);
                setSize(wm.size);
                bar.setVisible(wm.barVisible);
                JCheckBoxMenuItem showToolsItem =
                        (JCheckBoxMenuItem)getJMenuBar().getMenu(1).getItem(0);
                showToolsItem.setSelected(wm.barVisible);
        }
}
```

Usage:

```
    MyWin win= ...
    Memento memento = win.getMemento();       // when wishing to save preferences
    win.restore(memento);                     // when wishing to restore preferences
```

Access considerations

InterBit Training & Consulting Ltd.

- Note that many of the Memento attributes should be visible only to MyWin.
  - Outside managers should remember a memento, but not have access to its data.
  - We facilitate this by **hiding the concrete *MyWinMemento*** class.

Another Example

InterBit
Training & Consulting Ltd.

- Re-visiting our AI program:
  - We previously made clones of a **board,** so as to allow parallel processing.
  - For non-parallel processing, it is more memory-efficient to un-do steps.
  - One solution: remember board's **memento,** and restore it when backtracking.
  - Alternatively, on certain conditions you may prefer to define an *unexecute()* for **each step**.

Why not just clone?

InterBit
Training & Consulting Ltd.

- **Different purposes:**
  - Clone allows for duplicate copies to exist at the same time.
  - Memento restores the same object.
- **Try to avoid unnecessary cloning:**
  - Remember window size/location rather than cloning the entire graphical context.
  - Clone creates a different address; that affects pointers and mutex locks.
  - Don’t overwork the GC.

Problems and implementation

InterBit Training & Consulting Ltd.

Database Transactions

- Each transaction can succeed or fail, however if any operation fails, all operations would rollback and leave the database as if nothing has happened
- This mechanism of rolling back uses the memento design pattern.
- A transaction manager object which is responsible of performing transactions must perform operations on the table object while having the ability to undo the operation if it fails, or if any operation on any other table object fails.
- To be able to rollback, the transaction manager object would ask the table object for a memento before performing an operation and thus in case of failure, the memento object would be used to restore the table to its previous state

Why not just clone?

InterBit
Training & Consulting Ltd.

- **Different purposes:**
  - Clone allows for duplicate copies to exist at the same time.
  - Memento restores the same object.
- **Try to avoid unnecessary cloning:**
  - Remember window size/location rather than cloning the entire graphical context.
  - Clone creates a different address; that affects pointers and mutex locks.
  - Don’t overwork the GC.

Consequences

InterBit
Training & Consulting Ltd.

- Memento protects encapsulation and avoids exposing originator's internal state and implementation.
- It simplifies originator code such that the originator does not need to keep track of its previous state since this is the responsibility of the CareTaker.
- Using the memento pattern can be expensive depending on the amount of state information that has to be stored inside the memento object.
- In addition the caretaker must contain additional logic to be able to manage mementos.