CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2006
Doc 19 Memento & Facade
Apr 26, 2007

References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 283-292, 185-206
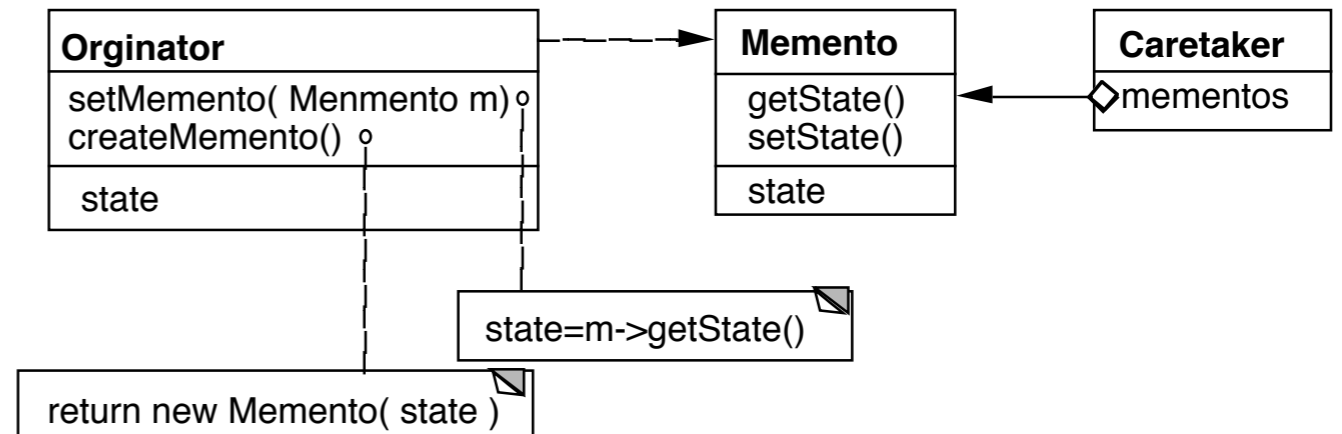
The Design Patterns Smalltalk Companion, Alpert, Brown, Woolf, 1998, pp. 297-304, 179-212

Memento

Store an object's internal state, so the object can be restored to this state later without violating encapsulation

undo, rollbacks

Orginator
setMemento(Menmento m) ∘ createMemento() ∘
state

Memento
getState() setState()
state

Caretaker
◇mementos

state=m->getState()

return new Memento(state)

Only originator:

    Can access Memento's get/set state methods
    Create Memento

Example

```
package Examples;
class Memento{
    private Hashtable savedState = new Hashtable();

    protected Memento() {}; //Give some protection

    protected void setState( String stateName, Object stateValue ) {
        savedState.put( stateName, stateValue );
    }


    protected Object getState( String stateName) {
        return savedState.get( stateName);
    }


    protected Object getState(String stateName, Object defaultValue ) {
        if ( savedState.containsKey( stateName ) )
            return savedState.get( stateName);
        else
            return defaultValue;
    }
}
```

Sample Orginator

```
package Examples;
class ComplexObject {
      private String name;
      private int someData;
      private Vector objectAsState = new Vector();

      public Memento createMemento() {
            Memento currentState = new Memento();
            currentState.setState( "name", name );
            currentState.setState( "someData", new Integer(someData) );
            currentState.setState( "objectAsState", objectAsState.clone() );
            return currentState;
      }

      public void restoreState( Memento oldState) {
            name = (String) oldState.getState( "name", name );
            objectAsState = (Vector) oldState.getState( "objectAsState" );
            Integer data = (Integer) oldState.getState( "someData");
            someData = data.intValue();
      }
```

Why not let the Orginator save its old state?

```
class ComplexObject {
        private String name;
        private int someData;
        private Vector objectAsState = new Vector();
        private Stack history;

        public createMemento() {
                Memento currentState = new Memento();
                currentState.setState( "name", name );
                currentState.setState( "someData", new Integer(someData) );
                currentState.setState( "objectAsState", objectAsState.clone() );
                history.push(currentState);
        }

        public void restoreState() {
                Memento oldState = history.pop();
                name = (String) oldState.getState( "name", name );
                objectAsState = (Vector) oldState.getState( "objectAsState" );
                Integer data = (Integer) oldState.getState( "someData");
                someData = data.intValue();
        }
```

Some Consequences

Expensive

Narrow & Wide interfaces - Keep data hidden

```
Class Memento {
public:
    virtual ~Memento();
private:
    friend class Originator;
    Memento();
    void setState(State*);
    State* GetState();
```

```
class Originator {
    private String state;

    private class Memento {
        private String state;
        public Memento(String stateToSave)
            { state = stateToSave; }
    public String getState() { return state; }
    }

    public Object memento()
        { return new Memento(state);}
```

Using Clone to Save State

```java
interface Memento extends Cloneable { }

class ComplexObject implements Memento {
    private String name;
    private int someData;

    public Memento createMemento() {
        Memento myState = null;
        try {
            myState =  (Memento) this.clone();
        }
        catch (CloneNotSupportedException notReachable) {
        }
        return myState;
    }

    public void  restoreState( Memento savedState) {
        ComplexObject myNewState = (ComplexObject)savedState;
        name = myNewState.name;
        someData = myNewState.someData;
    }
}
```
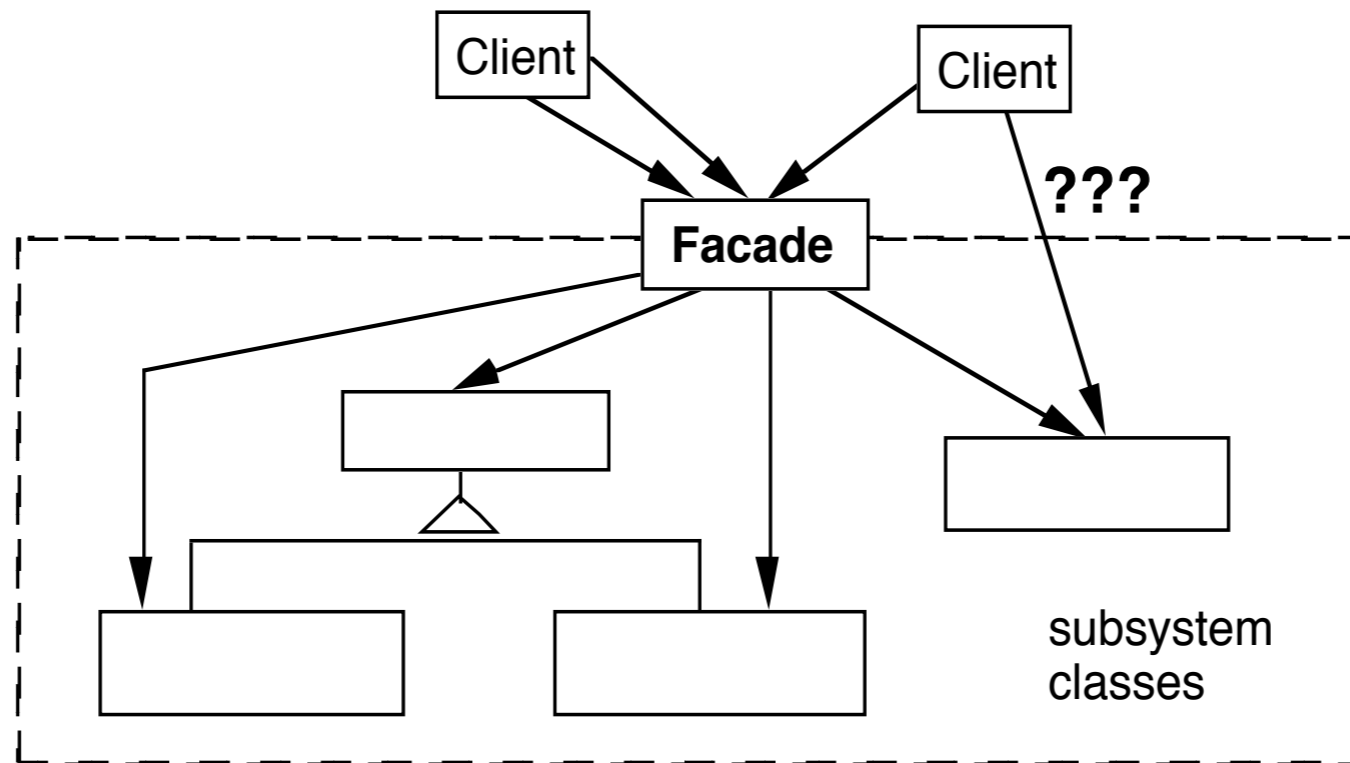
The Facade Pattern

Create a class that is the interface to the subsystem

Clients interface with the Facade class to deal with the subsystem

Consequences of Facade Pattern

It hides the implementation of the subsystem from clients

It promotes weak coupling between the subsystems and its clients

It does not prevent clients from using subsystem classes directly, should it?

Facade does not add new functionality to the subsystem

Compiler Example

The VisualWorks Smalltalk compiler system has 75 classes

Programmers only use Compiler, which uses the other classes

Compiler evaluate: '100 factorial'

```
| method compiler |
method := 'reset
    "Resets the counter to zero"
    count := 0.'.

compiler := Compiler new.
compiler
    parse:method
    in: Counter
    notifying: nil
```