References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 283-292, 185-206

Memento

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

undo, rollbacks

Only originator:
- Can access Memento’s get/set state methods
- Create Memento
Example

```java
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;
    }
}
```
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?
Some Consequences

Expensive

Narrow & Wide interfaces - Keep data hidden

Class Memento {
public:
    virtual ~Memento();
private:
    friend class Originator;
    Memento();
    void setState(State*);
    State* getState();
}
Using Clone to Save State

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
Flyweight

Use sharing to support large number of fine-grained objects efficiently
A document has many instances of the character 'a'

Character has
   Font
   width
   Height
   Ascenders
   Descenders
   Where it is in the document

Most of these are the same for all instances of 'a'

Use one object to represent all instances of 'a'
Intrinsic State

Information that is independent from the objects context

The information that can be shared among many objects
Extrinsic State

Information that is dependent on the objects context

The information that can not be shared among objects
Structure

Client

FlyweightFactory

getFlyweight( key )

Flyweight

operation( extrinsicState )

if (flyweight[key] exists)
  return existing flyweight
else
  create new flyweight
  add it to flyweight pool
  return new flyweight

ConcreteFlyweight

operation( extrinsicState )

intrinsicState

UnsharedConcreteFlyweight

operation( extrinsicState )

allState
The Hard Part

Separating state from the flyweight

Where does the extrinsic state go?

How do we get extrinsic state to correct object when it is needed?
public void testInterned() {
    String a1 = "catrat";
    String a2 = "cat";
    assertFalse(a1 == (a2 + "rat");

    String a3 = (a2 + "rat").intern();
    assertTrue(a1 == a3);
    String a4 = "cat" + "rat";
    assertTrue(a1 == a4);
    assertTrue(a3 == a4);
}