CSCI 253

Object Oriented Design:
Memento Pattern
George Blankenship

Overview

Creational Patterns
- Singleton
- Abstract factory
- Factory Method
- Prototype
- Builder

Structural Patterns
- Composite
- Façade
- Proxy
- Flyweight
- Adapter
- Bridge
- Decorator

Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

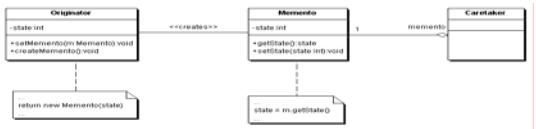
The Memento Pattern: The Problem

- The memento pattern is used to encapsulate the current state of an object in a memento object in order to be able to restore the object state later without exposing the internal representation of the object to the outside world
- An object that stores form information
  - A user can to make changes in the form
  - If they make a mistake, it can be returned to the original form values.
  - The object coule be serialized and then unserialized later but this is obviously messy and not a good solution.
  - An outside object use the form's accessor methods to pull out what you need to save the state but this causes high coupling between the class saving the state and the form; any changes in the form would require changes in the other class.
  - We need something that will allow you to save the state and restore it later without having to get involved in the details.

Memento Pattern George Blankenship 4

Example

The solution to this problem is make use of a Memento object which is responsible for storing the snapshot or current state. When a Memento object is needed, the Caretaker object, the object responsible for storing the Memento object, requests a Memento from the Originator via the createMemento() method. The state can then later be restored by passing the a Memento to the Originator.setMemento() method.



Memento Pattern George Blankenship 5

Memento Save/Load

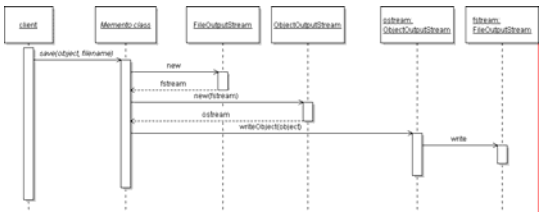
- recommendation: use a Memento class that has save/load code

```
// write the object named someObject to file "file.dat"
try {
  OutputStream os = new FileOutputStream("file.dat");
  ObjectOutputStream oos = new ObjectOutputStream(os);
  oos.writeObject(someObject);
  os.close();
} catch (IOException e) { ... }

// load the object named someObject from file "file.dat"
try {
  InputStream is = new FileInputStream("file.dat");
  ObjectInputStream ois = new ObjectInputStream(is);
  ArrayList someList = (ArrayList)ois.readObject();
  is.close();
} catch (Exception e) { ... }
```

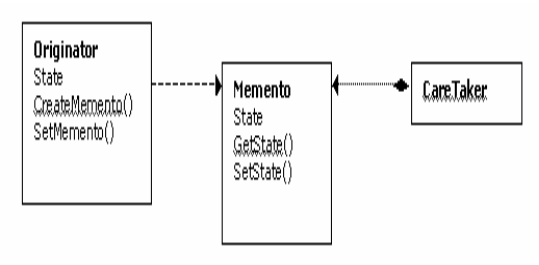
Memento Pattern George Blankenship 6

Memento Sequence Diagram



Memento Pattern George Blankenship 7

The Memento Pattern: Structure



Memento Pattern George Blankenship 8

The Memento Pattern: Participants

- Memento
  - Stores internal state of the Originator object. Protects against access by objects other than the originator. Mementos effectively have two interfaces (methods).
- Originator
  - Creates a Memento containing a snapshot of its current internal state Uses the Memento to restore its internal state
- CareTaker
  - Is responsible for the Memento's safekeeping Never operates on or examines the state of the Memento
- Generally a memento is responsible for storing a snapshot of the internal state of another object. Mementos are usually dynamic in nature

Memento Pattern George Blankenship 9

The Memento Pattern: Consequences

- Benefits
  - Since object oriented programming dictates that objects should encapsulate their state it would violate this law if objects' internal variables were accessible to external objects. The memento pattern provides a way of recording the internal state of an object in a separate object without violating this law
  - The memento eliminates the need for multiple creation of the same object for the sole purpose of saving its state.
  - The memento simplifies the Originator since the responsibility of managing Memento storage is no longer centralized at the Originator but rather distributed among the Caretakers
- Drawbacks
  - The Memento object must provide two types of interfaces: a narrow interface to the Caretaker and a wide interface to the Originator. That is, it must acts like a black box to everything except for the class that created it.
  - Using Mementos might be expensive if the Originator must store a large portion of its state information in the Memento or if the Caretakers constantly request and return the Mementos to the Originator.

Memento Pattern George Blankenship 10

The Memento Pattern: Implementation

- If you only need one Memento, combine the Originator and Caretaker into one object.
- If you need many Mementos, store only incremental changes. This will help to save space.
- Memento often used in conjunction with Command, Iterator and Singleton design patterns.
- Implementation of the Memento design pattern varies depending on the programming language. Implement the Originator as a friend class to the Memento in C++. Implement the Memento as an inner-class of the Originator in Java.

Memento Pattern George Blankenship 11