/ Design Patterns / Behavioral patterns / Memento

Memento in C++

Back to **Memento** description

Memento design pattern

Discussion. A memento is an object that stores a snapshot of the internal state of another object. It can be leveraged to support multi-level undo of the Command pattern. In this example, before a command is run against the Number object, Number's current state is saved in Command's static memento history list, and the command itself is saved in the static command history list. `Undo()` simply "pops" the memento history list and reinstates Number's state from the memento. `Redo()` "pops" the command history list. Note that Number's encapsulation is preserved, and Memento is wide open to Number.

```
#include <iostream.h>
class Number;

class Memento
{
  public:
    Memento(int val)
    {
        _state = val;
    }
  private:
    friend class Number; // not essential, but p287 suggests this
    int _state;
};

class Number
{
  public:
    Number(int value)
    {
        _value = value;
    }
    void dubble()
    {
        _value = 2 * _value;
    }
    void half()
    {
        _value = _value / 2;
    }
    int getValue()
    {
        return _value;
    }
    Memento *createMemento()
    {
        return new Memento(_value);
    }
    void reinstateMemento(Memento *mem)
    {
        _value = mem->_state;
    }
  private:
    int _value;
};

class Command
{
  public:
    typedef void(Number:: *Action)();
    Command(Number *receiver, Action action)
    {
        _receiver = receiver;
        _action = action;
    }
    virtual void execute()
    {
        _mementoList[_numCommands] = _receiver->createMemento();
        _commandList[_numCommands] = this;
        if (_numCommands > _highWater)
          _highWater = _numCommands;
        _numCommands++;
        (_receiver-> *_action)();
    }
    static void undo()
    {
        if (_numCommands == 0)
        {
            cout << "*** Attempt to run off the end!! ***" << endl;
            return ;
        }
        _commandList[_numCommands - 1]->_receiver->reinstateMemento
          (_mementoList[_numCommands - 1]);
        _numCommands--;
    }
    void static redo()
    {
        if (_numCommands > _highWater)
        {
            cout << "*** Attempt to run off the end!! ***" << endl;
            return ;
        }
        (_commandList[_numCommands]->_receiver->*(_commandList[_numCommands]
          ->_action))();
        _numCommands++;
    }
  protected:
    Number *_receiver;
    Action _action;
    static Command *_commandList[20];
    static Memento *_mementoList[20];
    static int _numCommands;
    static int _highWater;
};

Command *Command::_commandList[];
Memento *Command::_mementoList[];
```

```
int Command::_numCommands = 0;
int Command::_highWater = 0;

int main()
{
  int i;
  cout << "Integer: ";
  cin >> i;
  Number *object = new Number(i);

  Command *commands[3];
  commands[1] = new Command(object, &Number::dubble);
  commands[2] = new Command(object, &Number::half);

  cout << "Exit[0], Double[1], Half[2], Undo[3], Redo[4]: ";
  cin >> i;

  while (i)
  {
    if (i == 3)
      Command::undo();
    else if (i == 4)
      Command::redo();
    else
      commands[i]->execute();
    cout << "   " << object->getValue() << endl;
    cout << "Exit[0], Double[1], Half[2], Undo[3], Redo[4]: ";
    cin >> i;
  }
}
```

Output

```
Integer: 11
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 2
   5
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 1
   10
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 2
   5
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 3
   10
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 3
   5
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 3
   11
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 3
*** Attempt to run off the end!! ***
   11
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 4
   5
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 4
   10
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 4
   5
Exit[0], Double[1], Half[2], Undo[3], Redo[4]: 4
*** Attempt to run off the end!! ***
   5
```

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.

Learn more

Code examples

Java	Memento in Java	
C++	Memento in C++	
PHP	Memento in PHP	
Delphi	Memento in Delphi	Memento in Delphi

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us

© 2007-2018 SourceMaking.com
All rights reserved.

Terms / Privacy policy