Padrão de projeto: memento

Luis Aurélio Campos

INTRODUÇÃO

- Memento é um padrão de projeto comportamental que permite ao usuário salvar e restaurar o estado anterior de um objeto sem revelar os detalhes de sua implementação, ou seja, evitando problemas com o encapsulamento de classes de objetos.

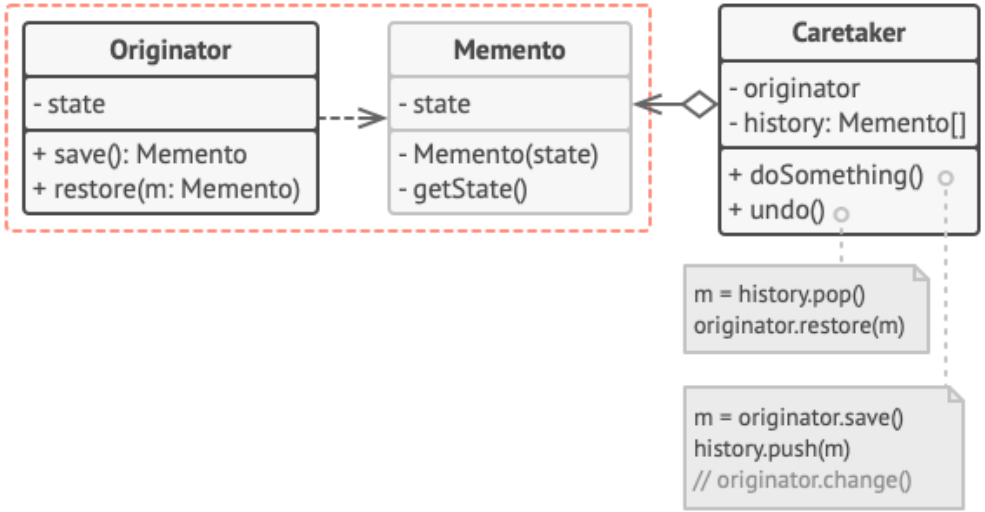
FUNCIONAMENTO

- O padrão Memento realiza a criação de snapshots do estado para o próprio dono do estado, o objeto Originador. O objeto Originador cria um novo memento a partir de si próprio, que realizará o controle de toda a execução do padrão Memento, criando uma nova instância da classe CareTaker, ou armazenador. Sempre que existir alguma modificação no objeto derivada de ações do sistema, o Originador irá criar um novo Memento.

FUNCIONAMENTO

- Com o estado salvo, o originador retorna para o armazenador, que guardará os Mementos em uma pilha, usando o LIFO, para, posteriormente, poder restaurar o objeto para qualquer estado salvo no memento, criando desta forma um histórico.

DIAGRAMA DO MEMENTO



APLICAÇÕES

- Um dos exemplos tradicionais de utilização do padrão comportamental Memento, é o editor de textos. Sempre quando há alteração do documento que está sendo trabalhado, é possível retornar ao estado anterior do texto, e ao retornar a este estado, também é possível retornar a um estado mais atual, essa funcionalidade pode ser representada muito bem pelo Memento.

IMPLEMENTAÇÃO EM C++

```
#include <iostream>
#include <vector>

/*
 * Memento
 * armazena o estado original do objeto Originator and protege
 * contra objetos de externos, fora o originator
 */

class Memento
{
private:
    // acessivel apenas pelo Originator
    friend class Originator;

    Memento(const int s) : state(s) {}

    void setState(const int s)
    {
        state = s;
    }

    int getState()
    {
        return state;
    }

private:
    int state;
};

/*
 * Originator
 * cria o memento contendo um snapshot do seu estado interno atual
 * e usa o memento para retornar a um outro estado
 */

class Originator
{
public:

    void setState(const int s)
    {
        std::cout << "Alterado o status para: " << s << "." << std::endl;
```

IMPLEMENTAÇÃO EM C++

```
        state = s;
    }

    int getState()
    {
        return state;
    }

    void setMemento(Memento* const m)
    {
        state = m->getState();
    }

    Memento *createMemento()
    {
        return new Memento(state);
    }

private:
    int state;
};

/*
 * CareTaker
 * é responsavel por armazenar o historico do memento
 * utiliza o Originator para salvar e retornar os estados
 */

class CareTaker
{
public:
    CareTaker(Originator* const o) : originator(o) {}

    ~CareTaker()
    {
        for (unsigned int i = 0; i < history.size(); i++)
        {
            delete history.at(i);
        }
        history.clear();
    }

    void salvarEstado()
    {
```

IMPLEMENTAÇÃO EM C++

```
        std::cout << "Estado salvo!" << std::endl;
        history.push_back(originator->createMemento());
    }

    void retornarEstado()
    {
        if ( history.empty() )
        {
            std::cout << "Impossivel retornar o estado!" << std::endl;
            return;
        }

        Memento *m = history.back();
        originator->setMemento(m);
        std::cout << "Estado retornado!" << std::endl;

        history.pop_back();
        delete m;
    }

private:
    Originator *originator;
    std::vector<Memento*> history;
};


int main()
{
    Originator *originator = new Originator();
    CareTaker *caretaker = new CareTaker(originator);

    originator->setState(1);
    caretaker->salvarEstado();

    originator->setState(2);
    caretaker->salvarEstado();

    originator->setState(3);

    caretaker->retornarEstado();
    std::cout << "Estado atual: " << originator->getState() << "." << std::endl;

    caretaker->retornarEstado();
    std::cout << "Estado atual: " << originator->getState() << "." << std::endl;
```

IMPLEMENTAÇÃO EM C++

```
133
134     delete originator;
135     delete caretaker;
136
137     return 0;
138 }
139
```

Obrigado!