Технологии программирования

Поведенческие паттерны. Chain of responsibility, Strategy, Command, Memento

Александр Михалевич

Типы паттернов проектирования

ПОРОЖДАЮЩИЕ

СТРУКТУРНЫЕ

ПОВЕДЕНЧЕСКИЕ

Поведенческие паттерны

CoR

STRATEGY

COMMAND

MEDIATOR

MEMENTO

INTERPRETER

Поведенческие паттерны

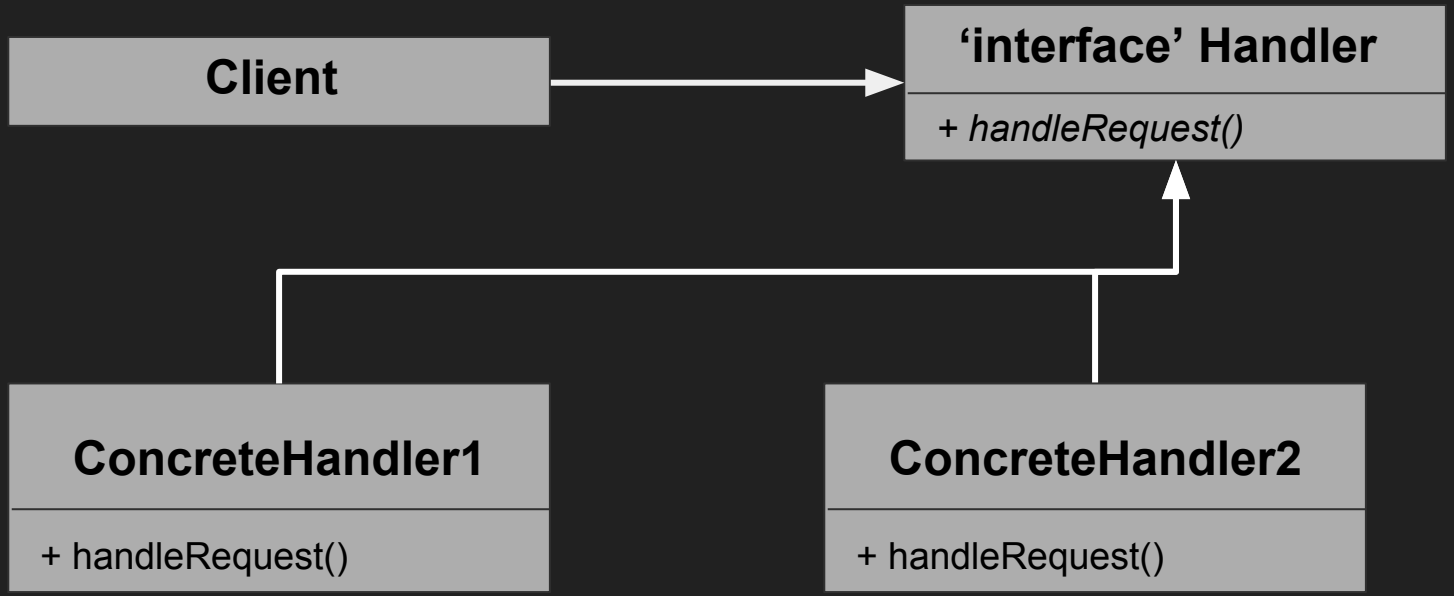
ITERATOR

STATE

OBSERVER

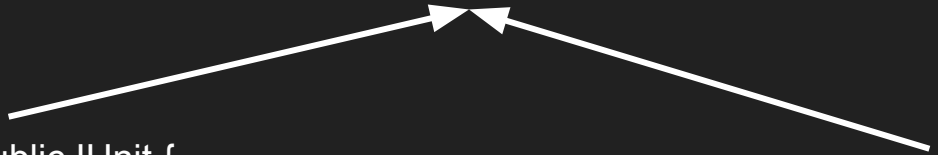
VISITOR

Chain of responsibility



Chain of responsibility example

```
class IUnit {
public:
        virtual result_t fight(CEnemy* enemy) = 0;
        virtual IUnit* set_next(IUnit* next) = 0;
};
```



```
class CFirstTypeUnit : public IUnit {
public:
        virtual result_t fight(CEnemy* enemy) {
                enemy.health -= m_attack *
                                enemy.resistance;
                m_health -= enemy.attack;
                if (enemy.alive) next.fight(enemy);
        }
        virtual IUnit* set_next(IUnit* next) { … }
};
```

```
class CSecondTypeUnit : public IUnit {
public:
        virtual result_t fight(CEnemy* enemy) {
                enemy.health -= m_attack * rand(100);
                m_health -= enemy.attack;
                if (enemy.alive) next.fight(enemy);
        }
        virtual IUnit* set_next(IUnit* next) { … }
};
```

Chain of responsibility example

```
client code
…
CFirstTypeUnit* unit = new CFirstTypeUnit;
IUnit* next = unit;

for (int i = 0; i < 100; ++i) {
        next = next->set_next(new CFirstTypeUnit);

for (int i = 0; i < 10; ++i)
        next = next->set_next(new CSecondTypeUnit);

CEnemy enemy;
unit->fight(enemy);
```

CoR vs Decorator

CoR	Decorator
выполняют независимые действия	расширяют какое-то конкретное действие
в любой момент передача по цепочке может быть прервана	не прерывают работу остальных декораторов

Relations

CoR & Composite

CoR vs Decorator

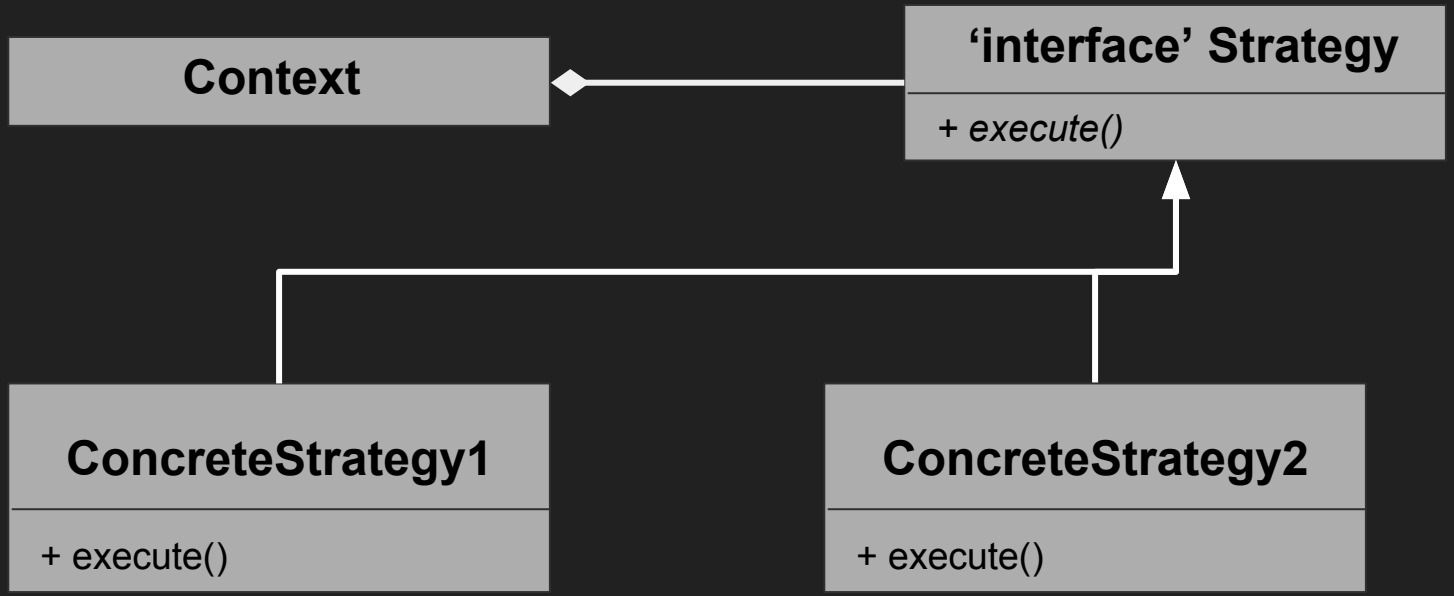
Когда применять?

есть несколько обработчиков; заранее неизвестно какой нужен

важна очередность выполнения обработчиков

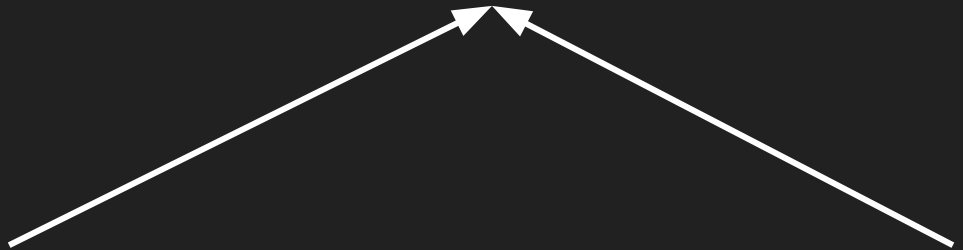
нужно менять цепь обработчиков в runtime

Strategy


Context
‘interface’ Strategy
+ execute()
ConcreteStrategy1
+ execute()
ConcreteStrategy2
+ execute()

Strategy example

```
class IMathEngine {
public:
        virtual int calculate(int first, int second) = 0;
};
```



```
class CAddOperation : public IMathEngine {
public:
        int calculate(int first, int second) {
                return first + second;
        }
};
```

```
class CSubstractOperation : public IMathEngine {
public:
        int calculate(int first, int second) {
                return first - second;
        }
};
```

Strategy example

```
class CContext {
public:
...
        int process(const Expression& expr) {
                auto id = expr.get_next_operation_id();
                auto strategy = CStrategyFactory::get(id);
                strategy->calculate(expr.next_left(), expr.next_right());

        }
...
};
```

Strategy example * (policy)

```
template<class T>
class CNewCreatorStrategy {
public:
        static T* create() { return new T; }
};
```

```
template<class T>
class CMallocCreatorStrategy {
public:
        static T* create() {
                void* buf = malloc(sizeof(T));
                if (!buf) return 0;
                return new(buf) T;
        }
};
```

```
template<class T>
class CPrototypeCreatorStrategy {
private:
        T* m_prototype;
public:
        CPrototypeCreatorStrategy(...) { ... }
        static T* create() {
                return m_prototype ?
                        m_prototype->clone() :
                        0;
        }
        // … get/set methods
};
```

Strategy example * (policy)

```
// код библиотеки
template<class CreationPolicy>
class CWidgetManager : public CreationPolicy {
...
};
```

← host class

```
// код приложения
typedef CWidgetManager< CNewCreatorStrategy<Widget> > MyWidgetManager;
```

Strategy example * (policy)

```
// код библиотеки
template<template <class Created> class CreationPolicy>
class CWidgetManager : public CreationPolicy<Widget> {
...
};
```

← host class

```
// код приложения
typedef CWidgetManager< CNewCreatorStrategy > MyWidgetManager;
```

Relations

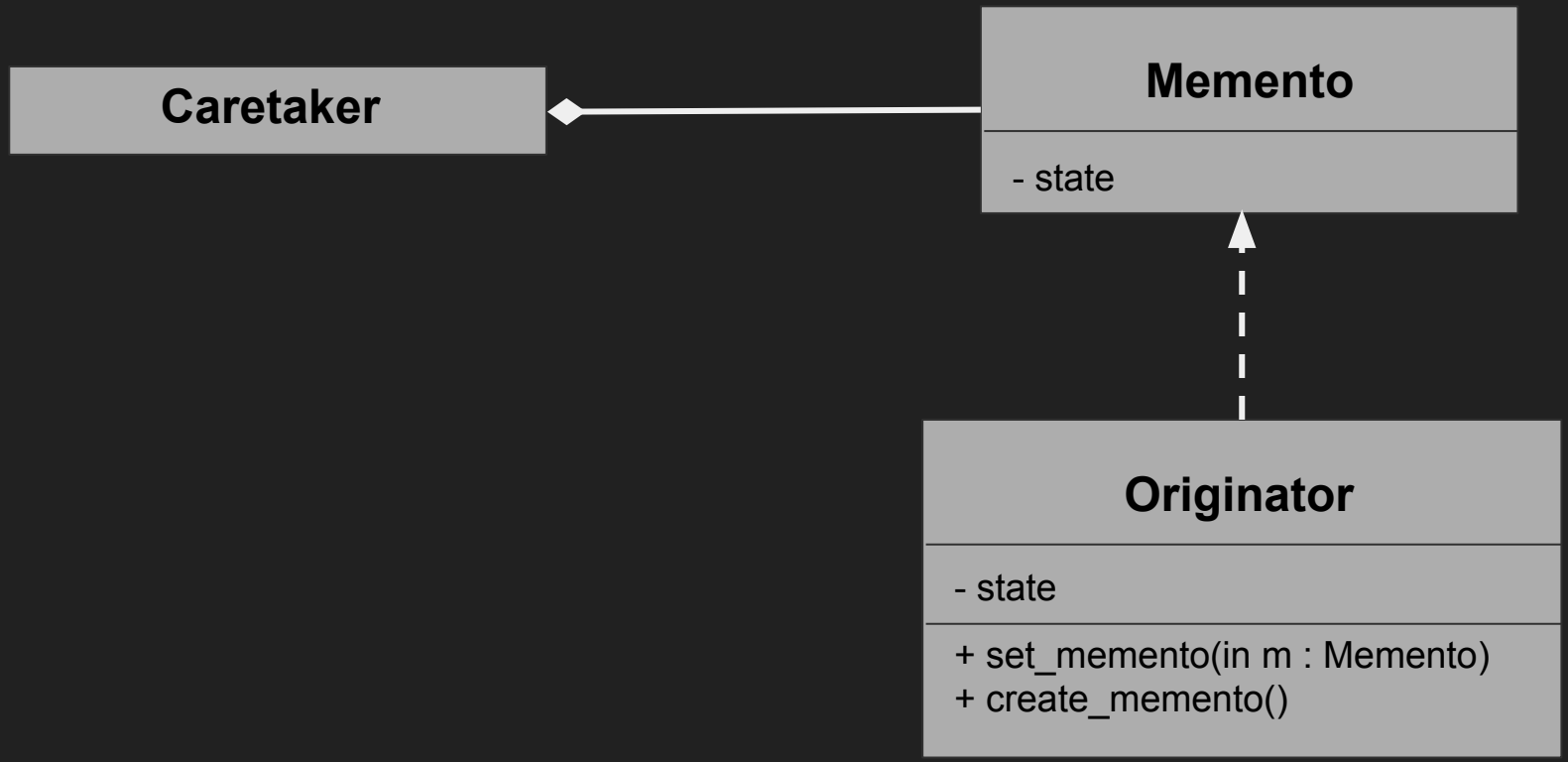
Strategy vs Decorator

Когда применять?

нужно использовать разные вариации одного алгоритма

есть много похожих классов, отличающихся только поведением

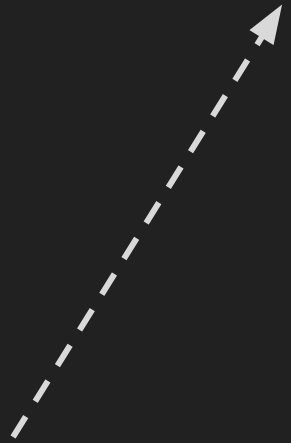
Memento


Caretaker
Memento
- state
Originator
- state
+ set_memento(in m : Memento)
+ create_memento()

Memento example

```
class IOriginator {
public:
        virtual ~IOriginator() {}
        virtual void set_memento(Memento* memento) = 0;
        virtual Memento* get_memento() = 0;
};
```

```
class CSpaceship : public IOriginator {    // originator
private:
        unsigned int m_health;
public:
        virtual ~CSpaceship () { }
        virtual void set_memento(Memento* memento) {
                m_health = memento->get_health();
        };
        virtual Memento* get_memento() {
                Memento* ret = new Memento();
                ret->set_health(m_health);
                return ret;
        }
};
```



```
class Memento {                                    // memento
private:
        unsigned int m_health;
public:
        void set_health(unsigned int health) { … }
        unsigned int get_health() { … }
};
```

```
class CGameContext {                                        // caretaker
private:
        Memento* m_memento;
public:
        void save_state(IOriginator* originator) {
                m_memento = originator->get_memento();
                ...
        }
        void load_state(IOriginator* originator) {
                originator->set_memento(m_memento);
                ...
        }
};
```