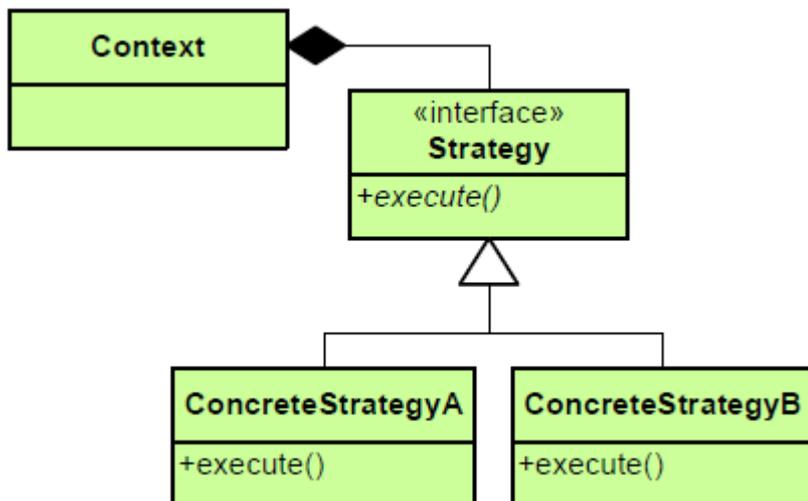# Strategy

Behavioral Pattern

strategy.cpp

```cpp
#include <iostream>
using namespace std;

class StrategyInterface
{
public:
    virtual void execute() const = 0;
};

class ConcreteStrategyA: public StrategyInterface
{
public:
```

```cpp
    virtual void execute() const
    {
        cout << "Called ConcreteStrategyA execute method" << endl;
    }
};

class ConcreteStrategyB: public StrategyInterface
{
public:
    virtual void execute() const
    {
        cout << "Called ConcreteStrategyB execute method" << endl;
    }
};

class ConcreteStrategyC: public StrategyInterface
{
public:
    virtual void execute() const
    {
        cout << "Called ConcreteStrategyC execute method" << endl;
    }
};

class Context
{
private:
    StrategyInterface * strategy_;

public:
    explicit Context(StrategyInterface *strategy):strategy_(strategy)
    {
    }

    void set_strategy(StrategyInterface *strategy)
    {
        strategy_ = strategy;
    }

    void execute() const
    {
        strategy_->execute();
    }
};

int main(int argc, char *argv[])
{
    ConcreteStrategyA concreteStrategyA;
    ConcreteStrategyB concreteStrategyB;
    ConcreteStrategyC concreteStrategyC;
```

```cpp
    Context contextA(&concreteStrategyA);
    Context contextB(&concreteStrategyB);
    Context contextC(&concreteStrategyC);

    contextA.execute(); // output: "Called ConcreteStrategyA execute
method"
    contextB.execute(); // output: "Called ConcreteStrategyB execute
method"
    contextC.execute(); // output: "Called ConcreteStrategyC execute
method"

    contextA.set_strategy(&concreteStrategyB);
    contextA.execute(); // output: "Called ConcreteStrategyB execute
method"
    contextA.set_strategy(&concreteStrategyC);
    contextA.execute(); // output: "Called ConcreteStrategyC execute
method"

    return 0;
}
```

http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Strategy