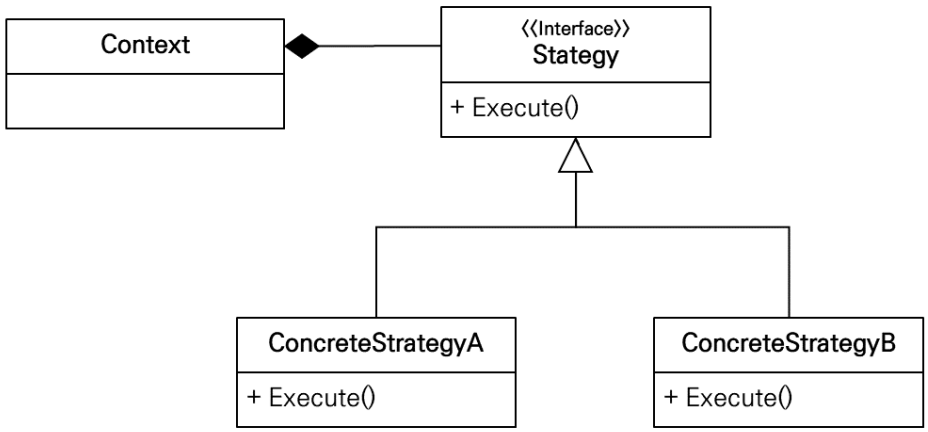
1. 스트래티지 / 전략 패턴(Strategy Pattern)

1) 스트래티지 / 전략 패턴

- 행동 패턴
- 알고리즘 군을 정의하고 각각 하나의 클래스로 캡슐화한 다음, 필요할 때 서로 교환해서 사용할 수 있게 해주는 디자인 패턴
- 스테이트 패턴과 비슷해 보이나 차이점 존재
    - 스테이트 패턴 : 상태에 따라 행위가 달라짐
    - 스트래티지 패턴 : 과정이 변경되나 결과가 동일

2) 클래스 다이어그램

Strategy Pattern_Class



3) 예제

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

class Strategy
{
public:
	virtual ~Strategy() {}
	virtual string DoAlgorithm(const vector<string> &data) const = 0;
};

class Context
{
private:
	Strategy *strategy_;

public:
	Context(Strategy *strategy = nullptr) : strategy_(strategy) { }
	~Context() { delete this->strategy_; }

	void set_strategy(Strategy *strategy)
	{
		delete this->strategy_;
		this->strategy_ = strategy;
	}

	void DoSomeBusinessLogic() const
	{
		cout << "Context: Sorting data using the strategy
			(not sure how it'll do it)\n";
		string result = this->strategy_
				->DoAlgorithm(vector<string>{"a", "e", "c", "b", "d"});
		cout << result << "\n";
	}
};

class ConcreteStrategyA : public Strategy
{
public:
	string DoAlgorithm(const vector<string> &data) const override
	{
		string result;
		for_each(begin(data), end(data), [&result](const string &letter)
		{ result += letter; }
		);
		sort(begin(result), end(result));

		return result;
	}
};
```

```
class ConcreteStrategyB : public Strategy
{
        string DoAlgorithm(const vector<string> &data) const override
        {
                string result;
                for_each(begin(data), end(data), [&result](const string &letter)
                { result += letter; }
                );

                sort(begin(result), end(result));
                for (int i = 0; i < result.size() / 2; i++)
                {
                        swap(result[i], result[result.size() - i - 1]);
                }

                return result;
        }
};
//========================================================================================

#include "Strategy.h"


void ClientCode()

{
        Context *context = new Context(new ConcreteStrategyA);

        cout << "Client: Strategy is set to normal sorting.\n";

        context->DoSomeBusinessLogic();

        cout << endl << "Client: Strategy is set to reverse sorting.\n";

        context->set_strategy(new ConcreteStrategyB);
        context->DoSomeBusinessLogic();

        delete context;
}

int main()
{
        ClientCode();
        return 0;
}
```