Strategy

Linda Marshall and Vreda Pieterse

Department of Computer Science
University of Pretoria

14 August 2015

Overview

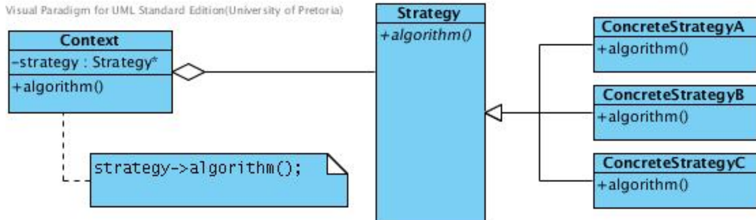
1. Identification
2. Structure
3. Participants
4. Related Patterns
5. Example
   - UML class diagram
   - Client code
   - `Context.h`

Name and Classification:

Strategy (Behavioural)

Intent:

"Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it." GoF(315)


Visual Paradigm for UML Standard Edition(University of Pretoria)
Context
-strategy : Strategy*
+algorithm()
strategy->algorithm();
Strategy
+algorithm()
ConcreteStrategyA
+algorithm()
ConcreteStrategyB
+algorithm()
ConcreteStrategyC
+algorithm()

- Context holds a pointer to a strategy object.
- The strategy object may vary in implementation in terms the ConcreteStrategy to which is being referred.
- The pattern alleviates the need for a complex conditional to select the desired strategy.

Strategy

- Declares an interface common to all supported algorithms.
- Context uses this interface to call the algorithm defined by a ConcreteStrategy.

ConcreteStrategy

- Implements the algorithm defined by the Strategy interface.

Context

- Is configured with a ConcreteStrategy object.
- Maintains a reference to a Strategy object.
- May define an interface that lets Strategy access its data.

Related Patterns

- **Factory Method** (107): Both Strategy and Factory Method use delegation through an abstract interface to concrete implementations. However, Strategy performs an operation while Factory Method create an object.

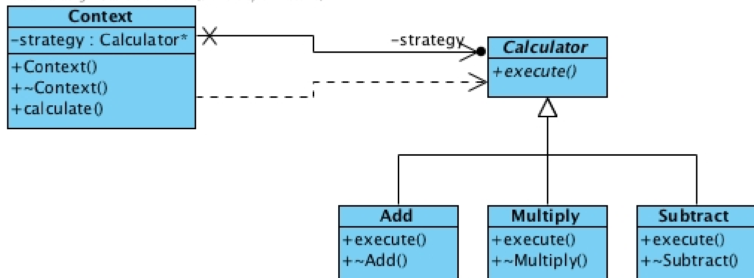
Related Patterns cont.

- **State**(305): State and Strategy have the same structure and apply the same techniques to achieve their goals, but differ in intent. Strategy is about implementations which accomplish the same result. One implementation can replace the other as the strategy requires. State is about doing different things based on the state, this relieves the caller from the burden of accommodating every possible state.

Related Patterns cont.

- **Template Method** (325): Where Template Method varies part(s) of the algorithm, Strategy varies the entire algorithm.
- **Flyweight**(195): Strategy objects often makes good flyweights.

Class diagram from existing code



```
#include <iostream>
#include "Context.h"
#include "Calculator.h"
using namespace std;
int  main() {
    Context* context[3];

    // Three contexts following different strategies
        context[0] = new Context(new Add());
        int resultA = context[0]->calculate(3,4);
        context[1] = new Context(new Subtract());
        int resultB = context[1]->calculate(3,4);
        context[2] = new Context(new Multiply());
        int resultC = context[2]->calculate(3,4);

    cout << "Result A : " << resultA << endl;
    cout << "Result B : " << resultB << endl;
    cout << "Result C : " << resultC << endl;

    for (int i = 0; i < 3; i++)  delete context[i];
    return 0;
}
```

```
#ifndef CONTEXT_H
#define CONTEXT_H
#include "Calculator.h"

class Context {
  public:
      Context(Calculator* strategy);
      ~Context();
      int calculate(int a, int b);
  private:
      Calculator* strategy;
};

#endif
```

```
#ifndef CALCULATOR_H
#define CALCULATOR_H
class Calculator {
  public:
      virtual int execute(int a, int b) = 0;
};
class Add : public Calculator {
  public:
      virtual int execute(int a, int b);
      ~Add();
};
class Subtract : public Calculator {
  public:
      virtual int execute(int a, int b);
      ~Subtract();
};
class Multiply : public Calculator {
  public:
      virtual int execute(int a, int b);
      ~Multiply();
};
#endif
```