Strategy

Department of Computer Science
University of Pretoria

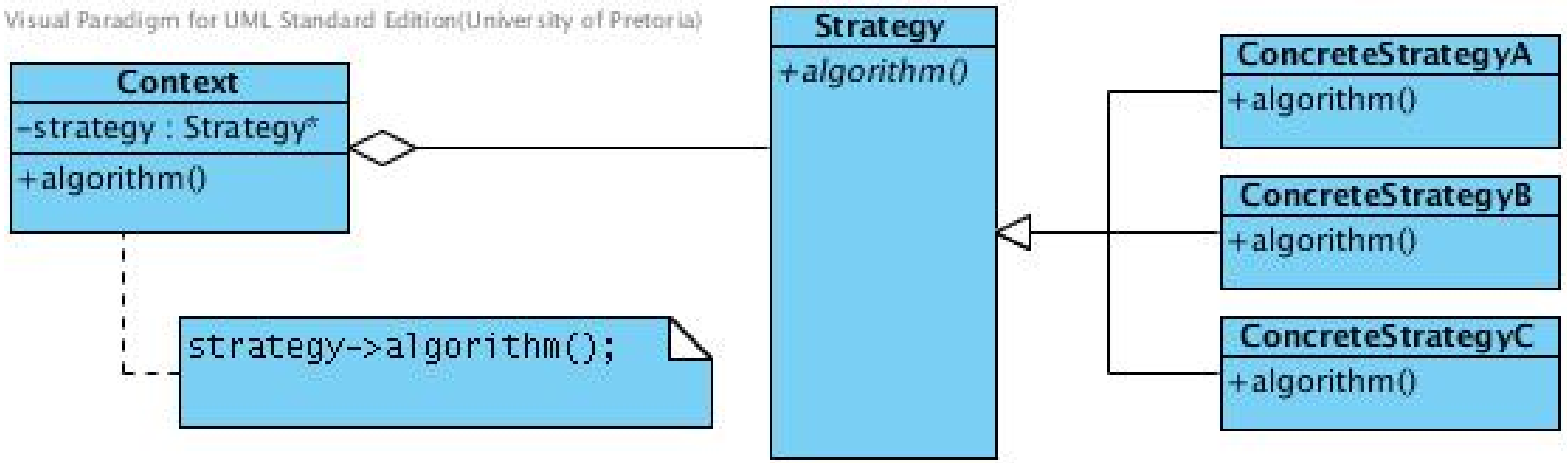
11 August 2023

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 of 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 creates an object.

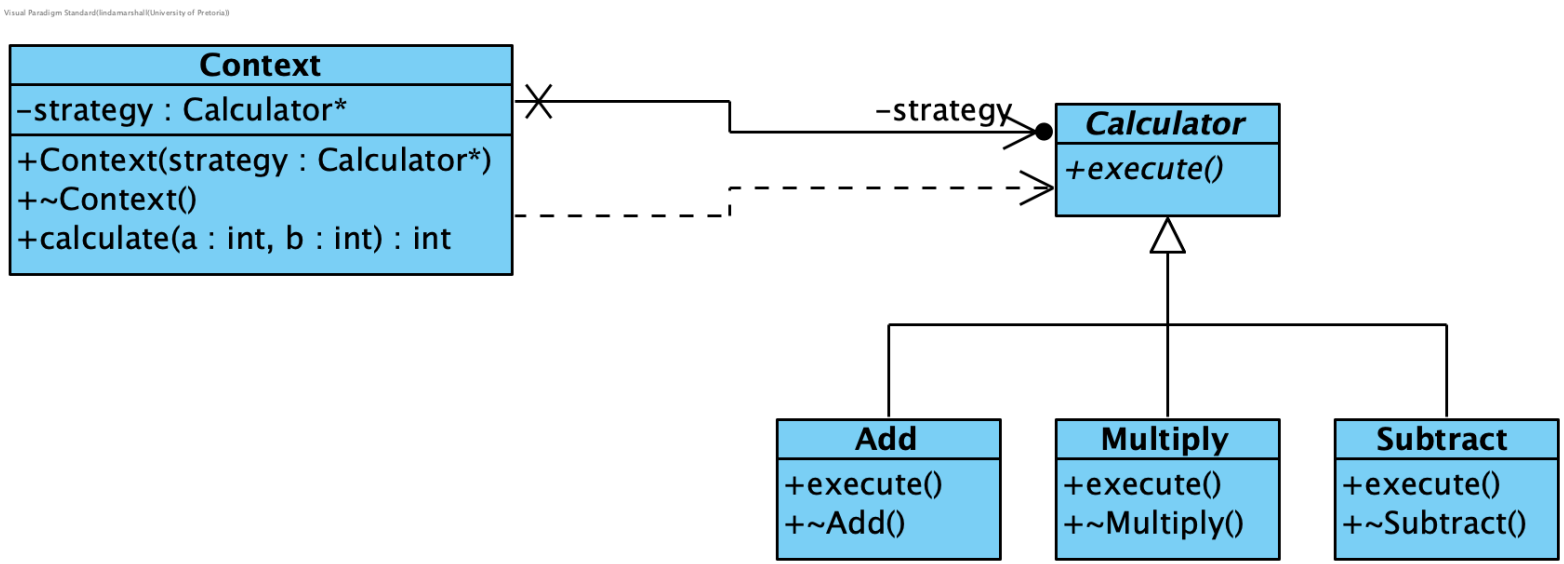
Related Patterns cont.

- **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



```
int main() {
    // Sorting algorithms
    Merge merge;
    Quick quick;
    Heap heap;

    // Searching algorithms
    Sequential sequential;
    BinaryTree binaryTree;
    HashTable hashTable;

    Collection colA;  // Context A
    colA.set_sort(&merge);
    colA.sort();

    Collection colB; // Context B
    colB.set_search(&binaryTree);
    colB.search();
}
```