# Software Engineering and Design Task 13

## Strategy Pattern

# Inhalt

* Konzept
* Verwendung
* UML Diagramm
* Code Beispiel
* Pro und Contra
* Fragen

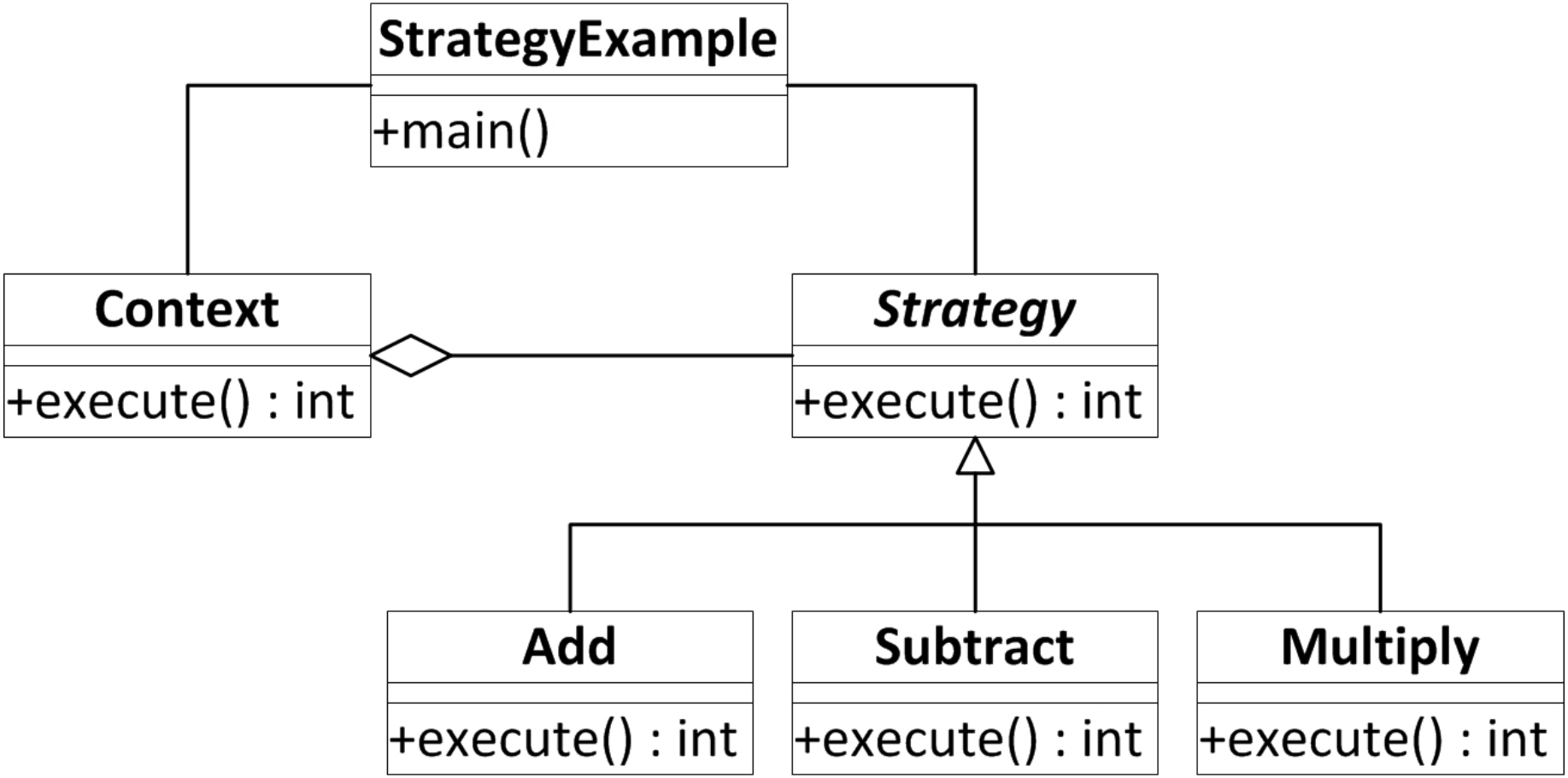
# Konzept

* Kategorie: Verhaltensmuster
* Unterschiedliche Algorithmen sollen zur Laufzeit schnell ausgewählt werden können

# Verwendung

* Wenn sich viele verwandte Klassen nur in ihrem Verhalten unterscheiden
* Wenn unterschiedliche (austauschbare) Varianten eines Algorithmus benötigt werden
* Wenn Daten innerhalb eines Algorithmus vor Klienten verborgen werden sollen
* Wenn verschiedene Verhaltensweisen innerhalb einer Klasse fest integriert sind (meist über Mehrfachverzweigungen) aber
  - die verwendeten Algorithmen wiederverwendet werden sollen bzw.
  - die Klasse flexibler gestaltet werden soll

# UML Diagramm



# Code Beispiel

```
interface Strategy {
    int execute(int a, int b);
}


class Context {
    private Strategy strategy;

    public Context(Strategy strategy) {
        this.strategy = strategy;
    }

    public int execute(int a, int b) {
        return strategy.execute(a, b);
    }
}
```

# Code Beispiel

```
class Add implements Strategy {
    public int execute(int a, int b) {
        System.out.println("Called Add's execute()");
        return a + b;  // Do an addition with a and b
    }
}

class Subtract implements Strategy {
    public int execute(int a, int b) {
        System.out.println("Called Subtract's execute()");
        return a - b;  // Do a subtraction with a and b
    }
}

class Multiply implements Strategy {
    public int execute(int a, int b) {
        System.out.println("Called Multiply's execute()");
        return a * b;   // Do a multiplication with a and b
    }
}
```

# Code Beispiel

```
class StrategyExample {
    public static void main(String[] args) {
        Context context;

        // Three contexts following different strategies
        context = new Context(new Add());
        int resultA = context.execute(2,5);

        context = new Context(new Subtract());
        int resultB = context.execute(2,5);

        context = new Context(new Multiply());
        int resultC = context.execute(2,5);

        System.out.println("Result A : " + resultA );
        System.out.println("Result B : " + resultB );
        System.out.println("Result C : " + resultC );
    }
}
```

# Pro und Contra

Pro

* Einfach erweiterbar
* Einfaches wechseln zwischen verschiedenen Algorithmen während der Laufzeit
* Es wird die Auswahl aus verschiedenen Implementierungen ermöglicht und dadurch erhöhen sich die Flexibilität und die Wiederverwendbarkeit
* Mehrfachverzweigungen können vermieden werden
  -> dies erhöht die Übersicht des Codes

# Pro und Contra

Contra

* Der Client muss die verschiedenen Strategien kennen
* Die Anzahl der Objekte nimmt stark zu
* Gegenüber der Implementierung der Algorithmen im Kontext entsteht hier ein zusätzlicher Kommunikationsaufwand zwischen Strategie und Kontext

# Fragen