

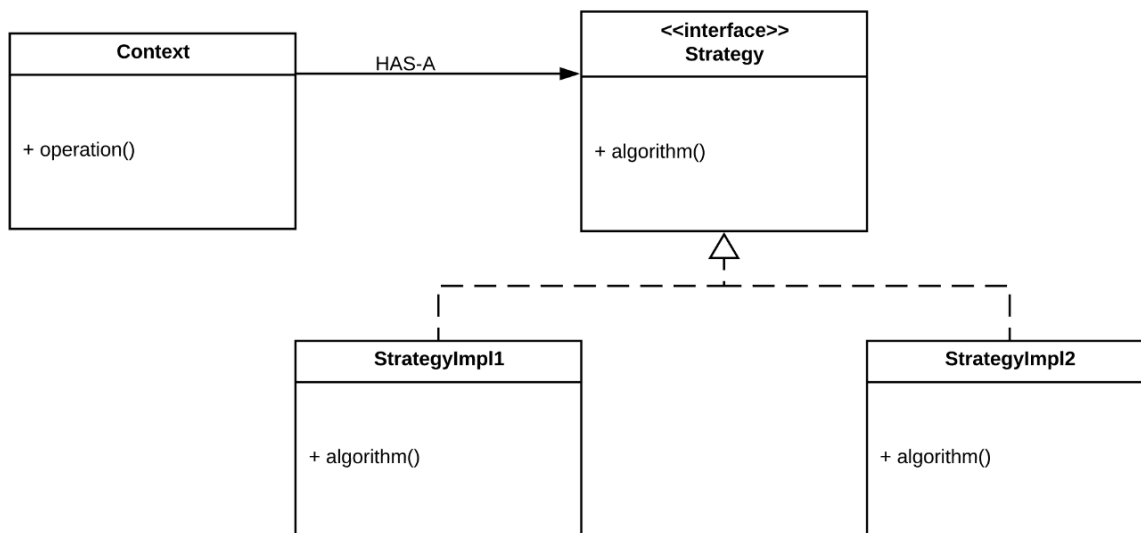


## 14. Design Patterns - day 2 of 2 (1)

### Strategy Pattern

Strategy Pattern says that "defines a family of functionality, encapsulate each one, and make them interchangeable".

The Strategy Pattern is also known as Policy.



Benefits:

- It provides a substitute to subclassing.
- It defines each behavior within its own class, eliminating the need for conditional statements.
- It makes it easier to extend and incorporate new behavior without changing the application.

Usage:

- When the multiple classes differ only in their behaviors.e.g. Servlet API.
- It is used when you need different variations of an algorithm.

```
package design.designpatterns.strategypattern;

public class StrategyClient {
```

```

    public static void main(String[] args) {

        Context context1 = new Context(new Multiply());

        System.out.print("Java new Multiply() Strategy " + context1.apply(2, 3));
    }
}

interface Strategy {
    int compute(int a, int b);
}

class Add implements Strategy {
    public int compute(int a, int b) {

        return a + b;
    }
}

class Multiply implements Strategy {
    public int compute(int a, int b) {

        return a * b;
    }
}

class Context {
    final Strategy strategy;

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

    public int apply(int a, int b) {
        return strategy.compute(a, b);
    }
}

```

## Another Example

```

import java.math.BigDecimal;

public class Client {
    public static void main(String[] args) {
        Discounter easterDiscounter = new EasterDiscounter();

        BigDecimal discountedValue = easterDiscounter
            .applyDiscount(BigDecimal.valueOf(100));
        System.out.println(discountedValue);

    }

    interface Discounter {
        BigDecimal applyDiscount(BigDecimal amount);
    }
    //Then let's say we want to apply a 50% discount at Easter and a 10% discount at Christmas. Let's implement our interface for each of t

    static class EasterDiscounter implements Discounter {
        @Override
        public BigDecimal applyDiscount(final BigDecimal amount) {
            return amount.multiply(BigDecimal.valueOf(0.5));
        }
    }
    //Java 8
    //Discounter easterDiscounter = amount -> amount.multiply(BigDecimal.valueOf(0.5));

    static class ChristmasDiscounter implements Discounter {
        @Override
        public BigDecimal applyDiscount(final BigDecimal amount) {
            return amount.multiply(BigDecimal.valueOf(0.9));
        }
    }
}

```

```
}  
}
```

References:

<https://javarevealed.wordpress.com/2013/07/05/factory-method-design-pattern/>

<https://www.geeksforgeeks.org/>