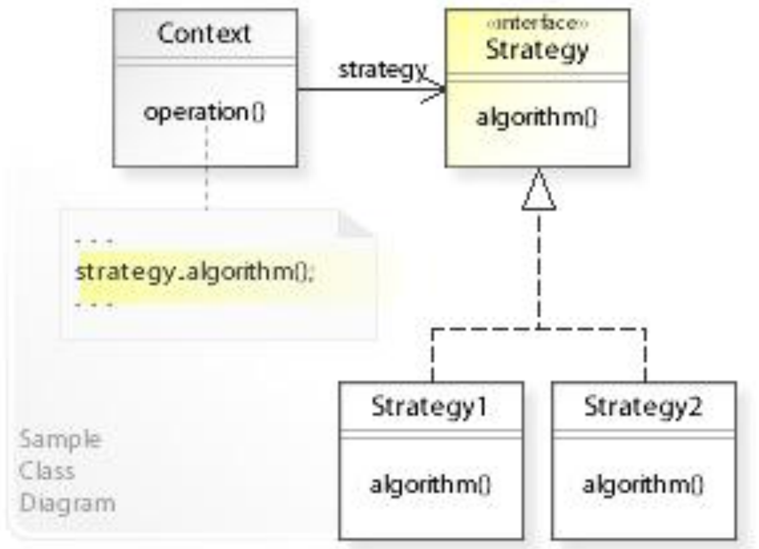
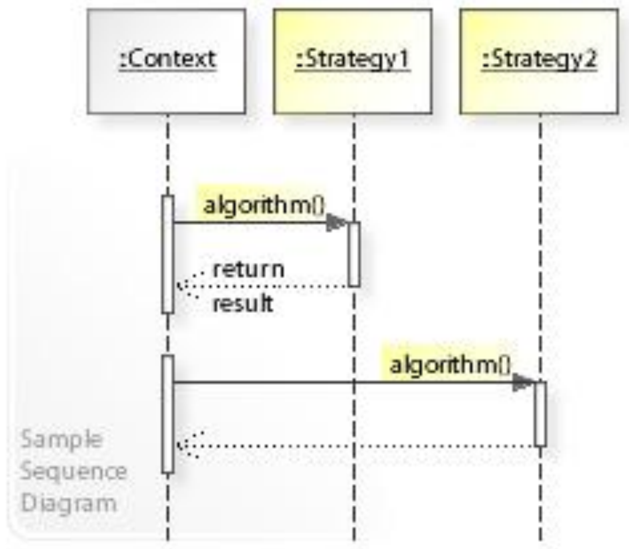
Strategy

Паттерн Стратегия

Strategy pattern (AKA Policy pattern)

- **Основная цель:** отделить логику принятия решений (decision making logic) от алгоритма (algorithm)
- **Результат:** выбор/запуск одного из нескольких алгоритмов





Strategy pattern – способы реализации

- Run-time
  - Интерфейсы
    - Лямбды (Java 8+)
      - Ссылки на методы
  - Наследование классов
  - Рефлексия
  - Ссылки на методы (не Java)
- Compile-time
  - Generics & templates - Policy-based design https://en.wikipedia.org/wiki/Policy-based_design

Выбор стратегии

- Реализацию стратегии выбирает само клиентское приложение

ВНИМАНИЕ
EDP предупреждает:
ID – плохо
ED – хорошо

```
SortStrategy sorter = new BubbleSorter();
```

- Реализацию стратегии предоставляет система/фреймворк/контейнер/… через фабрики / IoC(DI, SL, конфиги,…),…

```
Calendar calendar = Calendar.getInstance();
```

Зачем вообще нужна Стратегия?

- Вопрос:
  Зачем вообще нужна какая-то стратегия?
  Ведь если нужно что-то поменять можно сделать Ctrl-C + Ctrl-V или создать наследника и переопределить поведение?

- Ответ:
  SOLID !

Strategy + SOLID = BFF

- S – SRP - Single-responsiblity principle
  - A class should have only a single responsibility (нипаняяятна)
  - A class should have one and only one reason to change, meaning that a class should have only one job
- O – OCP - Open-closed Principle
  - Objects or entities should be open for extension, but closed for modification
- L – LSP - Liskov substitution principle
  - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program
- I – ISP - Interface segregation principle
  - Many client-specific interfaces are better than one general-purpose interface
  - Classes that implement interfaces, should not be forced to implement methods they do not use
- D – DIP - Dependency inversion principle
  - High level modules should not depend on low level modules rather both should depend on abstraction. Abstraction should not depend on details; rather detail should depend on abstraction

Strategy ♥ DRY, Strategy ♥ EDP

- DRY – Don't Repeat Yourself!
  - DRY is good ☺
  - WET is bad ☹
    - write everything twice
    - we enjoy typing
    - waste everyone's time
- EDP – Explicit Dependencies Principle
  - Methods and classes should explicitly require any collaborating objects they need in order to function correctly
    (through method parameters or constructor parameters)
    - Explicit Dependencies is good ☺
    - Implicit Dependencies is bad ☹

Coupling / Cohesion

Remember:

low in coupling and high in cohesion