**СТРАТЕГИЯ** - определяет семейство алгоритмов, инкапсулирует и обеспечивает их взаимозаменяемость, а также позволяет модифицировать алгоритмы независимо от их использования на стороне клиента.

ПРИМЕНЕНИЕ:

- когда нужно использовать разные вариации какого-то алгоритма внутри одного объекта

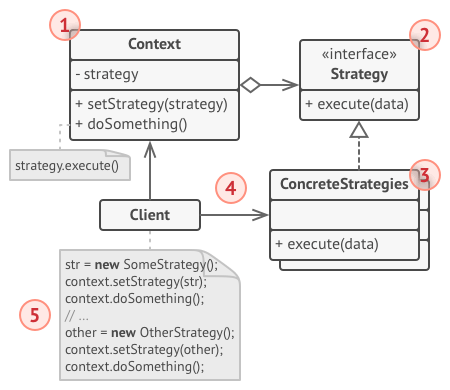
- когда есть множество похожих классов, отличающихся только некоторым поведением

- когда не хотите обнажать детали реализации алгоритмов для других классов

- когда различные вариации алгоритмов реализованы в виде развесистого условного оператора

- каждая ветка такого оператора представляет собой вариацию алгоритм

РЕАЛИЗАЦИЯ:

**1. КОНТЕКСТ**: хранит ссылку на объект конкретной стратегии, работая с ним через общий интерфейс стратегий.

**2. СТРАТЕГИЯ**: определяет интерфейс, общий для всех вариаций алгоритма. Контекст использует этот интерфейс для вызова алгоритма. Для контекста неважно, какая именно вариация алгоритма будет выбрана, так как все они имеют одинаковый интерфейс.

**3. КОНКРЕТНЫЕ СТРАТЕГИИ**: реализуют различные вариации алгоритма.

**4**. Во время выполнения программы контекст получает вызовы от клиента и делегирует их объекту конкретной стратегии.

**5**. Клиент должен создать объект конкретной стратегии и передать его в конструктор контекста. Кроме этого, клиент должен иметь возможность заменить стратегию на лету, используя сеттер. Благодаря этому, контекст не будет знать о том, какая именно стратегия сейчас выбрана.

ПЛЮСЫ:

- горячая замена алгоритмов на лету

- изолирует код и данные алгоритмов от остальных классов

- уход от наследования к делегированию

- реализует принцип открытости/закрытости

МИНУСЫ:

- усложняет программу за счёт дополнительных классов

- клиент должен знать, в чём состоит разница между стратегиями, чтобы выбрать подходящую