ВОЩИЛО ЮРИЙ

РАЗРАБОТКА КОРПОРАТИВНЫХ РЕШЕНИЙ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ JAVA

ОСНОВНЫЕ КОНЦЕПЦИИ И ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

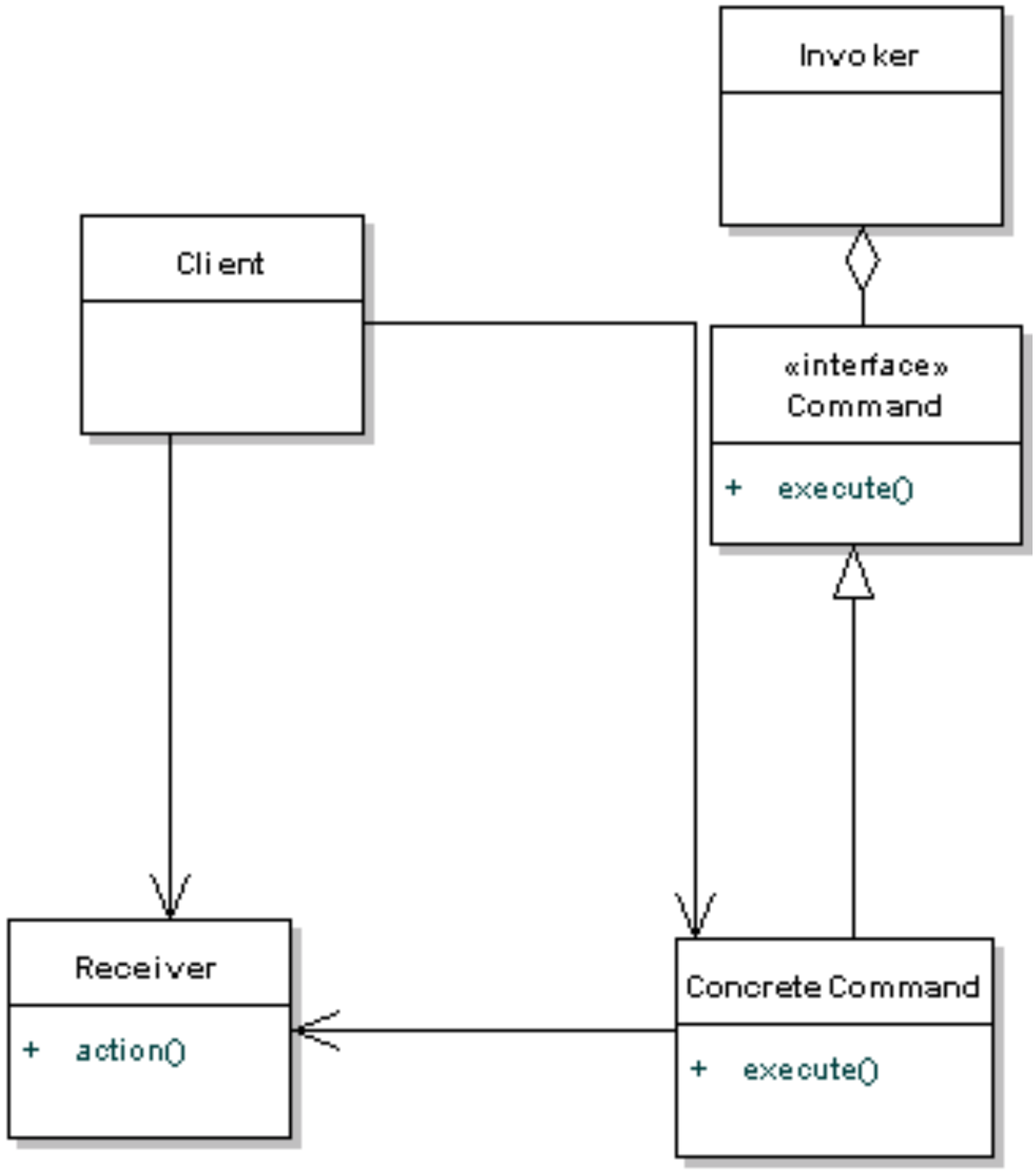
1. Command
2. Builder
3. Strategy

COMMAND

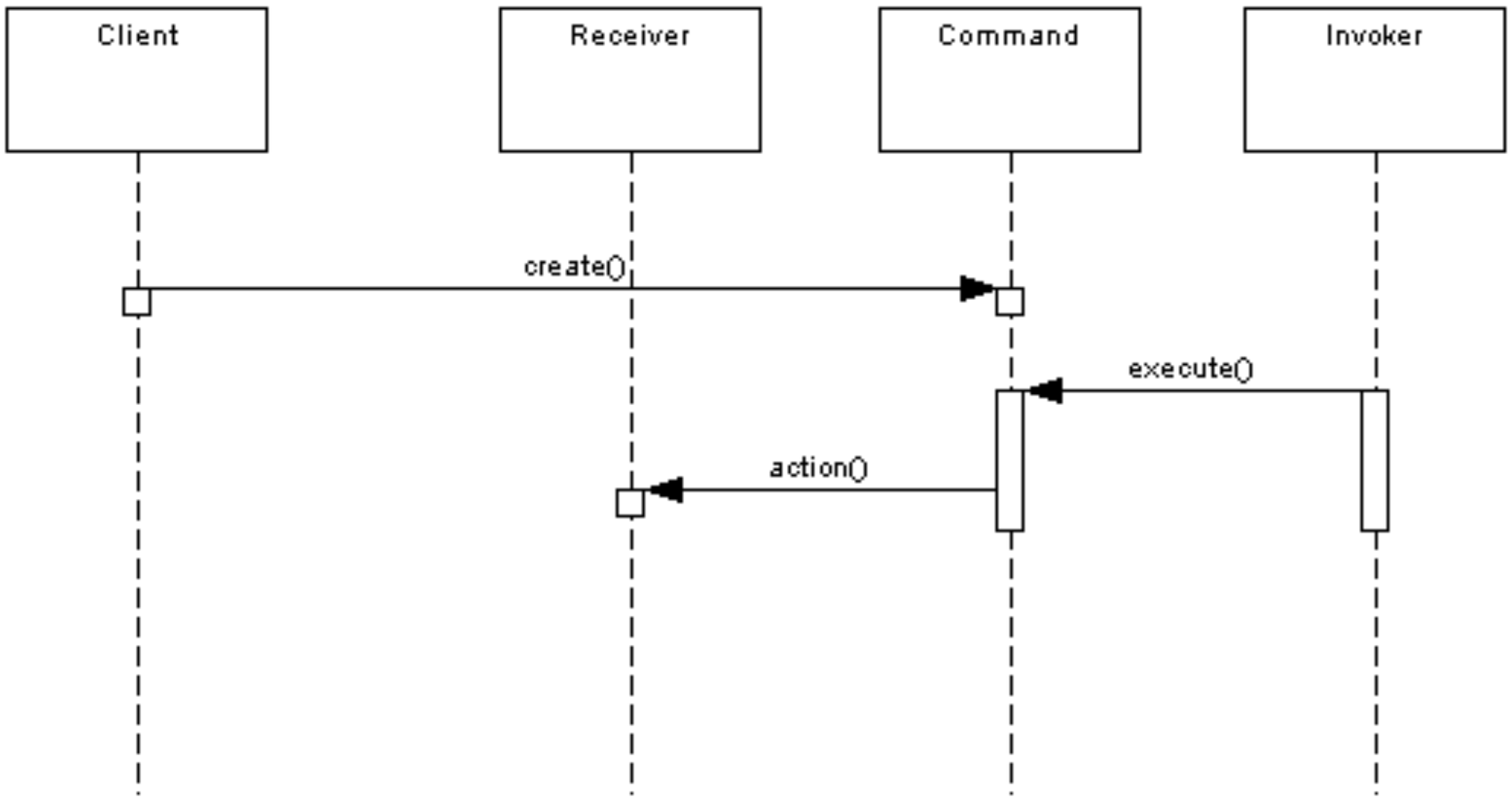
Команда (англ. Command) – поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие.

Объект команды заключает в себе само действие и его параметры.

COMMAND


Invoker
Client
«interface»
Command
+ execute()
Receiver
+ action()
Concrete Command
+ execute()

COMMAND



ЦЕЛЬ COMMAND

Создание структуры, в которой класс-отправитель и класс-получатель не зависят друг от друга напрямую. Организация обратного вызова к классу, который включает в себя класс-отправитель.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Кнопки пользовательского интерфейса и пункты меню

В Swing и Borland Delphi Action (действие) является объектом команды. В дополнение к способности выполнить нужную команду, Action может иметь связанную с ним иконку, сочетание клавиш, текст всплывающей подсказки и так далее. Кнопка на панели инструментов или пункт меню могут быть полностью инициализированы с использованием только объекта Action.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Запись макросов

Если все действия пользователя представлены в виде объектов команды, программа может записать последовательность действий, просто сохраняя список командных объектов в том порядке, в котором они выполняются. Затем она может «воспроизвести» одни и те же действия, выполняя те же объекты команд в той же последовательности.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Многоуровневая отмена операций (Undo)

Если все действия пользователя в программе реализованы в виде командных объектов, программа может сохранить стек последних выполненных команд. Когда пользователь хочет отменить команду, программа просто выталкивает последний объект команды и выполняет его метод undo().

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Сети

Можно отправить объекты команд по сети для выполнения на другой машине, например действие игрока в компьютерной игре.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Индикаторы выполнения

Предположим, что программа имеет последовательность команд, которые она выполняет по порядку. Если каждый объект команды имеет метод getEstimatedDuration() (получить оценочную длительность), программа может легко оценить общую продолжительность процесса. Она может показать индикатор выполнения, который отражает, насколько близка программа к завершению всех задач.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Пулы потоков

Типичный класс пула потоков общего назначения может иметь метод addTask(), который добавляет рабочий элемент к внутренней очереди заданий ожидающих своего выполнения. Он поддерживает пул потоков, которые выполняют команды из очереди. Элементы в очереди являются объектами команд. Как правило, эти объекты реализуют общий интерфейс, такой как java.lang.Runnable, что позволяет пулу потоков запустить команды на выполнение, даже если он сам был написан без каких-либо знаний о конкретных задачах, для которых он будет использоваться.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

Транзакции

Аналогично операции «отмена» система управления базами данных (СУБД) или установщик программного обеспечения может хранить список операций, которые были или будут выполнены. Если одна из них закончится неудачей, то все остальные могут быть отменены или быть отброшены (обычно называется откат). Например, если две связанные между собой таблицы базы данных должны быть обновлены, а второе обновление терпит неудачу, то транзакция может быть откачена, чтобы первая таблица не содержала недопустимую ссылку.

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ COMMAND

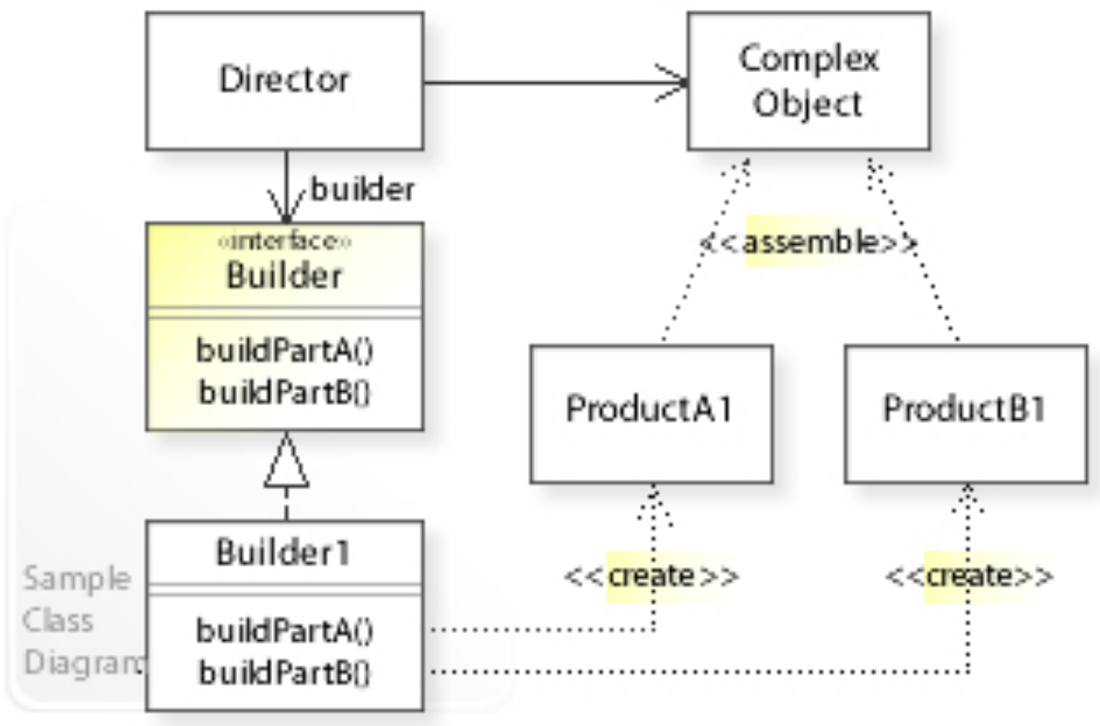
Мастера

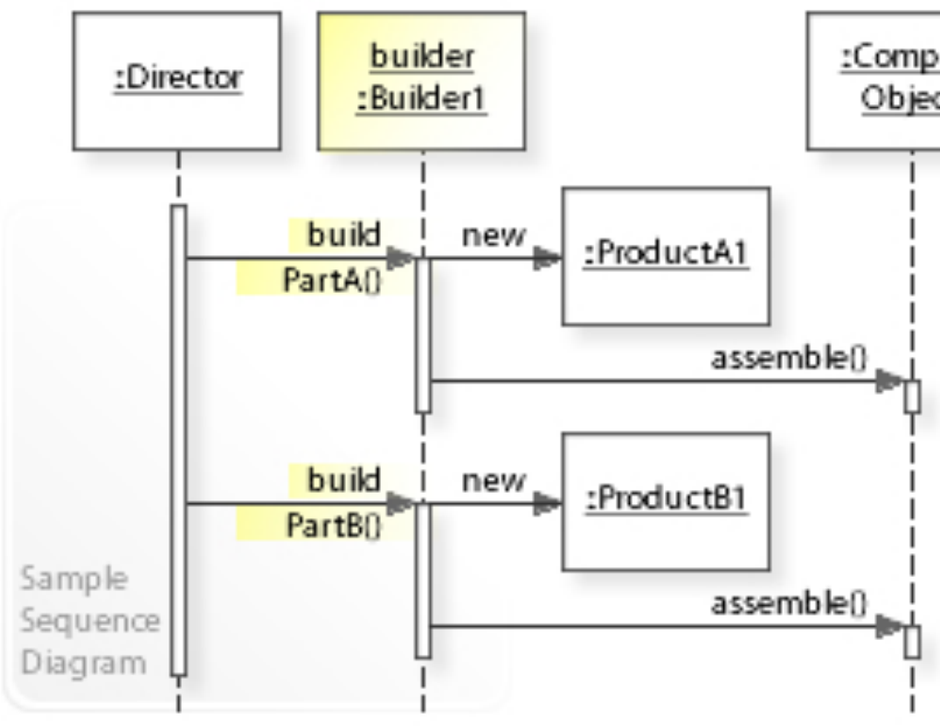
Часто мастер (мастер установки или любой другой) представляет несколько страниц конфигурации для одного действия, которое происходит только тогда, когда пользователь нажимает на кнопку «Готово» на последней странице. В этих случаях, естественный способ отделить код пользовательского интерфейса от кода приложения является реализация мастера с помощью объекта команд. Объект команда создается при первом отображении мастера. Каждая страница мастера сохраняет свои изменения в объекте команды, поэтому объект заполняется по мере перехода пользователя. Кнопка «Готово» просто запускает метод execute() на выполнение.

BUILDER

Строитель (англ. Builder) — порождающий шаблон проектирования предоставляет способ создания составного объекта.

BUILDER


Director
builder
«interface»
Builder
buildPartA()
buildPartB()
Builder1
buildPartA()
buildPartB()
Complex
Object
<<assemble>>
ProductA1
ProductB1
<<create>>
<<create>>
Sample
Class
Diagram


:Director
builder
:Builder1
:Complex
Object
build
PartA()
new
:ProductA1
assemble()
build
PartB()
new
:ProductB1
assemble()
Sample
Sequence
Diagram

ЦЕЛЬ BUILDER

Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

- позволяет изменять внутреннее представление продукта;
- изолирует код,
- реализующий конструирование и представление;
- дает более тонкий контроль над процессом конструирования.

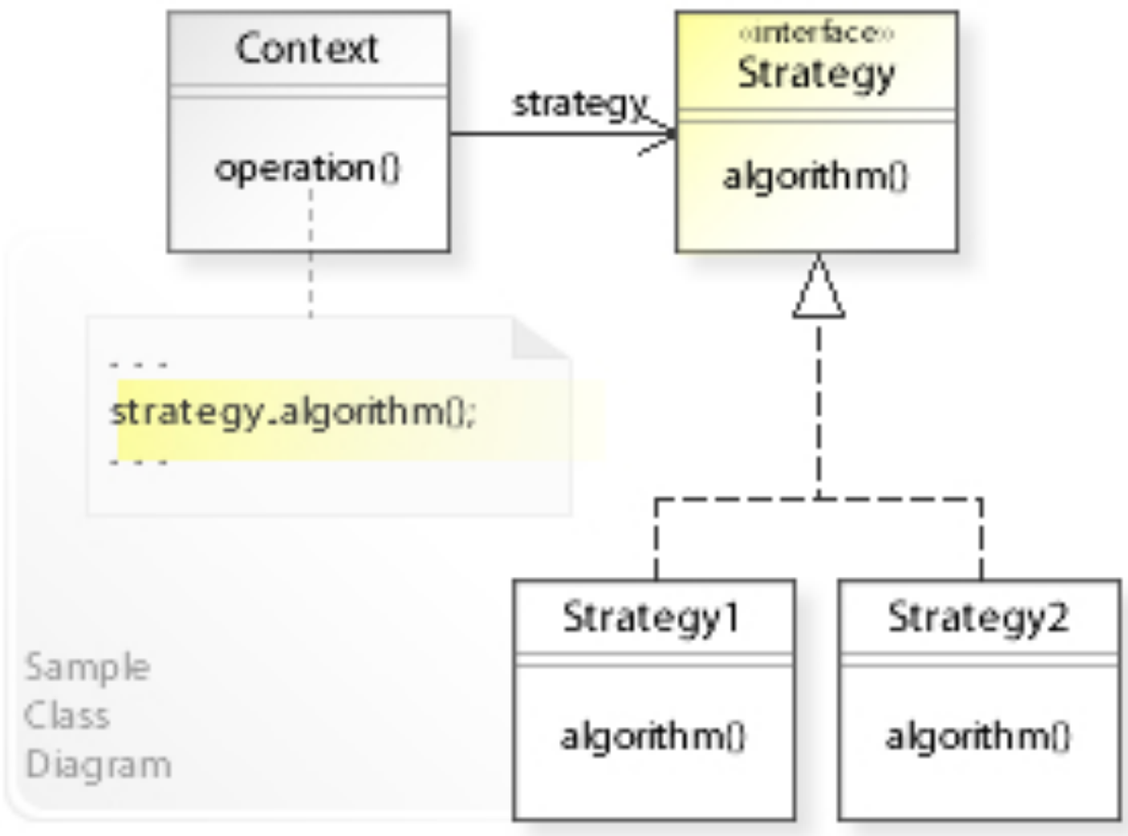
STRATEGY

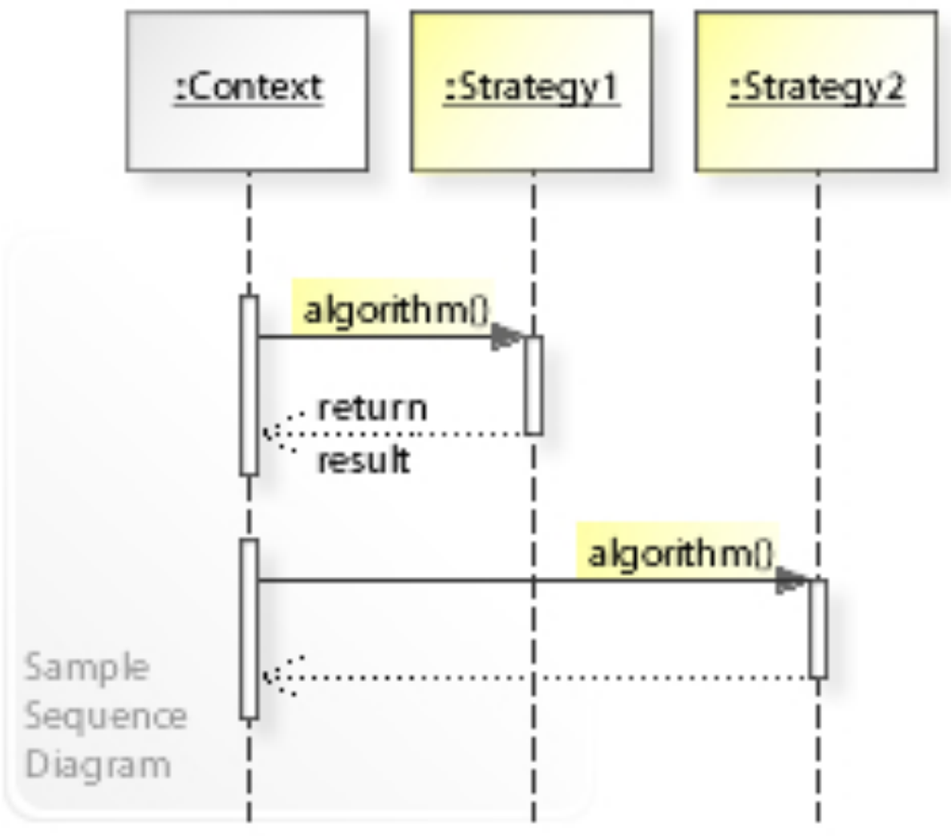
Стратегия (англ. Strategy) – поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости.

Это позволяет выбирать алгоритм путём определения соответствующего класса.

Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.

STRATEGY


Context
operation()
strategy
«interface»
Strategy
algorithm()
strategy.algorithm();
Strategy1
algorithm()
Strategy2
algorithm()
Sample
Class
Diagram


:Context
:Strategy1
:Strategy2
algorithm()
return
result
algorithm()
Sample
Sequence
Diagram

ЦЕЛЬ STRATEGY

По типу клиента (или по типу обрабатываемых данных) выбрать подходящий алгоритм, который следует применить. Если используется правило, которое не подвержено изменениям, нет необходимости обращаться к шаблону «стратегия».

- Программа должна обеспечивать различные варианты алгоритма или поведения
- Нужно изменять поведение каждого экземпляра класса
- Необходимо изменять поведение объектов на стадии выполнения
- Введение интерфейса позволяет классам-клиентам ничего не знать о классах, реализующих этот интерфейс и инкапсулирующих в себе конкретные алгоритмы