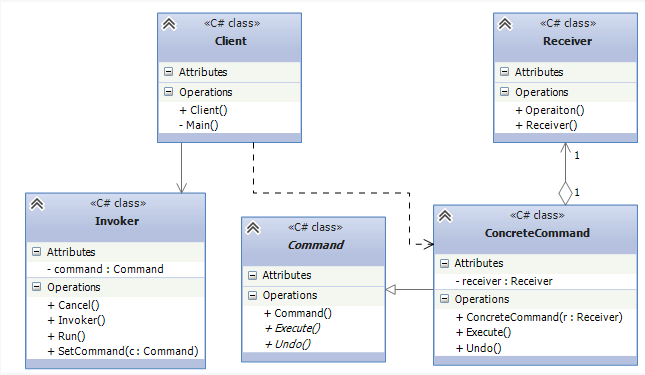
Поведенческий паттерн – **Команда**

# Определение

**Команда (Comand)** – это поведенческий паттерн проектирования, который превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

Хорошие программы обычно структурированы в виде слоёв. Самый распространённый пример — слои интерфейса и бизнес-логики. Первый всего лишь рисует красивую картинку для пользователя. Но когда нужно сделать что-то важное, интерфейс «просит» слой бизнес-логики заняться этим.

# UML-диаграмма



**Command**: интерфейс, представляющий команду. Обычно определяет метод **Execute**() для выполнения действия, а также нередко включает метод **Undo**(), реализация которого должна заключаться в отмене действия команды

**ConcreteCommand**: конкретная реализация команды, реализует метод **Execute**(), в котором вызывается определенный метод, определенный в классе **Receiver**

**Receiver**: получатель команды. Определяет действия, которые должны выполняться в результате запроса.

**Invoker**: инициатор команды - вызывает команду для выполнения определенного запроса

**Client**: клиент - создает команду и устанавливает ее получателя с помощью метода **SetCommand**()

# Особенности

* Убирает прямую зависимость между объектами, вызывающими операции и объектами, которые их непосредственно выполняют.
* Позволяет реализовать простую отмену и повтор операций.
* Позволяет реализовать отложенный запуск команд.
* Позволяет собирать сложные команды из простых.
* Реализует принцип открытости/закрытости.
* Усложняет код программы за счёт дополнительных классов.

# Список источников

1. <https://metanit.com/sharp/patterns/3.3.php>
2. <https://refactoring.guru/ru/design-patterns/command>

# Пример – Пульты и устройства

В этом примере паттерн Команда служит для ведения истории выполненных операций, позволяя, отменять их, если потребуется.

Код примера взят с <https://refactoring.guru/ru/design-patterns/command>

Адаптация под C# <https://github.com/IlyaPershin/Comand>

// Абстрактная команда задаёт общий интерфейс для

// всех команд.

**abstract class** **Command** **is**

**protected** **field** app: Application

**protected** **field** editor: Editor

**protected** **field** backup: text

**constructor** Command(app: Application, editor: Editor) **is**

**this**.app = app

**this**.editor = editor

// Сохраняем состояние редактора.

**method** saveBackup() **is**

backup = editor.text

// Восстанавливаем состояние редактора.

**method** undo() **is**

editor.text = backup

// Главный метод команды остаётся абстрактным, чтобы

// каждая конкретная команда определила его по-своему.

// Метод должен возвратить true или false, в зависимости

// о того, изменила ли команда состояние редактора, а

// значит, нужно ли её сохранить в истории.

**abstract** **method** execute()

// Конкретные команды.

**class** **CopyCommand** **extends** Command **is**

// Команда копирования не записывается в историю, так

// как она не меняет состояние редактора.

**method** execute() **is**

app.clipboard = editor.getSelection()

**return** **false**

**class** **CutCommand** **extends** Command **is**

// Команды, меняющие состояние редактора, сохраняют

// состояние редактора перед своим действием и

// сигнализируют об изменении, возвращая true.

**method** execute() **is**

saveBackup()

app.clipboard = editor.getSelection()

editor.deleteSelection()

**return** **true**

**class** **PasteCommand** **extends** Command **is**

**method** execute() **is**

saveBackup()

editor.replaceSelection(app.clipboard)

**return** **true**

// Отмена это тоже команда.

**class** **UndoCommand** **extends** Command **is**

**method** execute() **is**

app.undo()

**return** **false**

// Глобальная история команд — это стек.

**class** **CommandHistory** **is**

**private** **field** history: array of Command

// Последний зашедший...

**method** push(c: Command) **is**

Push command to the end of history array.

// ...выходит первым.

**method** pop():Command **is**

Get the most recent command from history.

// Класс редактора содержит непосредственные операции над

// текстом. Он отыгрывает роль получателя – команды

// делегируют ему свои действия.

**class** **Editor** **is**

**field** text: string

**method** getSelection() **is**

**Return** selected text.

**method** deleteSelection() **is**

Delete selected text.

**method** replaceSelection(text) **is**

Insert clipboard contents at current position.

// Класс приложения настраивает объекты для совместной

// работы. Он выступает в роли отправителя — создаёт

// команды, чтобы выполнить какие-то действия.

**class** **Application** **is**

**field** clipboard: string

**field** editors: array of Editors

**field** activeEditor: Editor

**field** history: CommandHistory

// Код, привязывающий команды к элементам интерфейса

// может выглядеть примерно так.

**method** createUI() **is**

*// ...*

copy = function() {executeCommand(

**new** CopyCommand(**this**, activeEditor)) }

copyButton.setCommand(copy)

shortcuts.onKeyPress("Ctrl+C", copy)

cut = function() { executeCommand(

**new** CutCommand(**this**, activeEditor)) }

cutButton.setCommand(cut)

shortcuts.onKeyPress("Ctrl+X", cut)

paste = function() { executeCommand(

**new** PasteCommand(**this**, activeEditor)) }

pasteButton.setCommand(paste)

shortcuts.onKeyPress("Ctrl+V", paste)

undo = function() { executeCommand(

**new** UndoCommand(**this**, activeEditor)) }

undoButton.setCommand(undo)

shortcuts.onKeyPress("Ctrl+Z", undo)

// Запускаем команду и проверяем, надо ли добавить её

// в историю.

**method** executeCommand(command) **is**

**if** (command.execute)

history.push(command)

// Берём последнюю команду из истории и заставляем её

// все отменить. Мы не знаем конкретный тип команды, но

// это и не важно, так как каждая команда знает как

// отменить своё действие.

**method** undo() **is**

command = history.pop()

**if** (command != **null**)

command.undo()