**КОМАНДА** - превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций. Отделяет объект, выдающий запросы, от объекта, который умеет их выполнять. Для многократной отмены, вместо ссылки на последнюю выполненную команду нужно хранить стек предыдущих команд, а метод undo() будет извлекать последнюю команду. Объекты команд могут помещаться в очередь, затем программные потоки последовательно их извлекают, вызывают их метод execute() и переходят к следующему объекту. Команды можно регистрировать в логе на диске с помощью дополнительного метода store(), а загружать в случае восстановления системы с помощью дополнительного метода load()

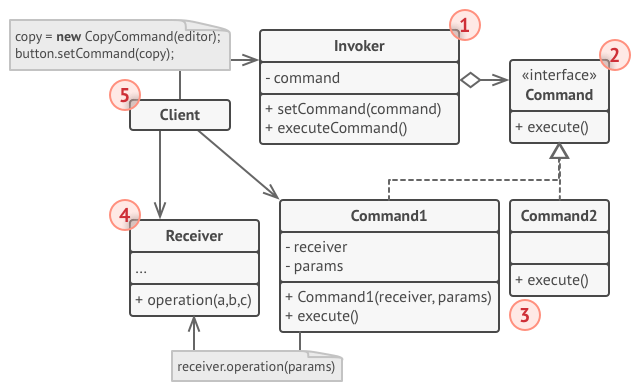
ПРИМЕНЕНИЕ:

- когда хотите параметризовать объекты выполняемым действием

- когда хотите ставить операции в очередь, выполнять их по расписанию или передавать по сети

- когда нужна операция отмены

РЕАЛИЗАЦИЯ:

**1. ИНИЦИАТОР**: хранит ссылку на объект команды и обращается к нему, когда нужно выполнить какое-то действие. Работает с командами только через их общий интерфейс, вызывая их метод execute(), т.е. не знает, какую конкретно команду использует, так как получает готовый объект команды от клиента. Например, "пульт дистанционного управления".

**2. АБСТРАКЦИЯ КОМАНДЫ**: описывает общий для всех конкретных команд интерфейс. Имеет минимум один метод execute() для запуска команды. (Опционально) методы undo(), store(), load() - для отмены изменений, сохранения команды в лог и загрузки при восстановлении соответственно. Например, "команда"

**3. КОНКРЕТНЫЕ КОМАНДЫ**: реализуют различные запросы, следуя общему интерфейсу команд. Обычно команда не делает всю работу самостоятельно, а лишь передаёт вызов получателю, которым является один из объектов бизнес-логики. Содержит ссылку на получателя, чей конкретный метод она использует в своем execute(). Например: "команда включения света", "команда выключения света", "(макро)команда включения света и воды", "пустая команда"

**4. ПОЛУЧАТЕЛЬ**: содержит бизнес-логику программы. В этой роли может выступать практически любой объект. Обычно команды перенаправляют вызовы получателям. Иногда, чтобы упростить программу, вы можете избавиться от получателей, «слив» их код в классы команд. Например, "свет", "вода"

**5. КЛИЕНТ**: создаёт объекты конкретных команд, передавая в них все необходимые параметры, среди которых могут быть и ссылки на объекты получателей. После этого клиент связывает объекты отправителей с созданными командами

ПЛЮСЫ:

- убирает прямую зависимость между объектами, вызывающими операции, и объектами, которые их непосредственно выполняют

- позволяет реализовать простую отмену и повтор операций

- позволяет реализовать отложенный запуск операций

- позволяет собирать сложные команды из простых

- реализует принцип открытости/закрытости

МИНУСЫ:

- усложняет код программы из-за введения множества дополнительных классов