




[🏠](#) / [Паттерны проектирования](#) / [Команда](#) / [Go](#)



# Команда на Go

**Команда** — это поведенческий паттерн, позволяющий заворачивать запросы или простые операции в отдельные объекты.

Это позволяет откладывать выполнение команд, выстраивать их в очереди, а также хранить историю и делать отмену.

 [Подробнее о паттерне Команда →](#)

## Навигация

 [Интро](#)

 [Концептуальный пример](#)

 [button](#)

 [command](#)


 [onCommand](#)

[Главная](#) [Рефакторинг](#) [Паттерны](#) [Премиум контент](#)  
[Форум](#) [Связаться](#)



© 2014-2023 Refactoring.Guru.

Все права защищены.

 Иллюстрации нарисовал Дмитрий Жарт

[Условия использования](#)

[Политика конфиденциальности](#)

[Использование контента](#) [About us](#)

# Концептуальный пример

Давайте рассмотрим паттерн Команда на примере телевизора. TV может быть включен двумя способами:

- кнопка ВКЛ на пульте дистанционного управления;
- кнопка ВКЛ на самом телевизоре.

Мы можем начать с реализации объекта команды ВКЛ с телевизором в роли получателя. Когда на эту команду вызывается метод `execute`, она, в свою очередь, вызывает функцию `TV.on`. Вышеуказанное определяет вызывающий объект. На самом деле мы будем иметь два вызывающих объекта: пульт и сам ТВ. Оба будут содержать объект команды ВКЛ.

Заметьте, что мы обернули один и тот же запрос в несколько вызывающих объектов. Это же можно делать и с другими командами. Преимуществом создания отдельных объектов команд является отделение логики пользовательского интерфейса от внутренней бизнес-логики. Нет нужды разрабатывать отдельные исполнители для каждого вызывающего объекта – сама команда содержит всю информацию, необходимую для ее исполнения. Соответственно, ее можно использовать для отсроченного выполнения задачи.

## **button.go: Отправитель**

```
package main

type Button struct {
    command Command
}

func (b *Button) press() {
    b.command.execute()
}
```

## **command.go: Интерфейс команды**

```
package main

type Command interface {
```

```
        execute()  
    }
```

## onCommand.go: Конкретная команда

```
package main  
  
type OnCommand struct {  
    device Device  
}  
  
func (c *OnCommand) execute() {  
    c.device.on()  
}
```

## offCommand.go: Конкретная команда

```
package main  
  
type OffCommand struct {  
    device Device  
}  
  
func (c *OffCommand) execute() {  
    c.device.off()  
}
```

## device.go: Интерфейс получателя

```
package main  
  
type Device interface {  
    on()  
    off()  
}
```

## tv.go: Конкретный получатель

```
package main

import "fmt"

type Tv struct {
    isRunning bool
}

func (t *Tv) on() {
    t.isRunning = true
    fmt.Println("Turning tv on")
}

func (t *Tv) off() {
    t.isRunning = false
    fmt.Println("Turning tv off")
}
```

## main.go: Клиентский код

```
package main

func main() {
    tv := &Tv{}

    onCommand := &OnCommand{
        device: tv,
    }

    offCommand := &OffCommand{
        device: tv,
    }

    onButton := &Button{
        command: onCommand,
    }
    onButton.press()

    offButton := &Button{
        command: offCommand,
    }
}
```

```
    offButton.press()  
}
```

## output.txt: Результат выполнения

```
Turning tv on  
Turning tv off
```

По материалам: ***Golang By Example***

ВЕРНУТЬСЯ НАЗАД

ЧИТАЕМ ДАЛЬШЕ

← Цепочка обязанностей на Go

Итератор на Go →

## Команда на других языках программирования

