Command in Go

Command is behavioral design pattern that converts requests or simple operations into objects.

The conversion allows deferred or remote execution of commands, storing command history, etc.

Learn more about Command →

Navigation

Conceptual Example

Let’s look at the Command pattern with the case of a TV. A TV can be turned ON by either:

- ON Button on the remote;
- ON Button on the actual TV.

We can start by implementing the ON command object with the TV as a receiver. When the execution method is called on this command, it, in turn, calls the `TV.on` function. The last part is defining an invoker. We’ll actually have two invokers: the remote and the TV itself. Both will embed the ON command object.

Notice how we have wrapped the same request into multiple invokers. The same way we can do with other commands. The benefit of creating a separate command object is that we decouple the UI logic from underlying business logic. There’s no need to develop different handlers for each of the invokers. The command object contains all the information it needs to execute. Hence it can also be used for delayed execution.

button.go: Invoker

```go
package main

type Button struct {
    command Command
}

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

command.go: Command interface

```go
package main

type Command interface {
    execute()
}
```

onCommand.go: Concrete command

```go
package main

type OnCommand struct {
    device Device
}

func (c *OnCommand) execute() {
    c.device.on()
}
```

offCommand.go: Concrete command

```go
package main

type OffCommand struct {
    device Device
}

func (c *OffCommand) execute() {
    c.device.off()
}
```

device.go: Receiver interface

```go
package main

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

tv.go: Concrete receiver

```go
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: Client code

```go
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: Execution result

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

Based on: ***Golang By Example***

RETURN

← Chain of Responsibility in Go

READ NEXT

Iterator in Go →

Command in Other Languages