Если вы видите что-то необычное, просто сообщите мне.

??????? (Command)

Паттерн Command относится к поведенческим паттернам уровня объекта.

Паттерн Command позволяет представить запрос в виде объекта. Из этого следует, что команда - это объект. Такие запросы, например, можно ставить в очередь, отменять или возобновлять.

В этом паттерне мы оперируем следующими понятиями: Command - запрос в виде объекта на выполнение; Receiver - объект-получатель запроса, который будет обрабатывать нашу команду; Invoker - объект-инициатор запроса.

Паттерн Command отделяет объект, инициирующий операцию, от объекта, который знает, как ее выполнить. Единственное, что должен знать инициатор, это как отправить команду.

Требуется для реализации:

1. Базовый абстрактный класс Command описывающий интерфейс команды;
2. Класс ConcreteCommand, реализующий команду;
3. Класс Invoker, реализующий инициатора, записывающий команду и провоцирующий её выполнение;
4. Класс Receiver, реализующий получателя и имеющий набор действий, которые команда можем запрашивать;

Invoker умеет складывать команды в стопку и инициировать их выполнение по какому-то событию. Обратившись к Invoker можно отменить команду, пока та не выполнена.

ConcreteCommand содержит в себе запросы к Receiver, которые тот должен выполнять. В свою очередь Receiver содержит только набор действий (Actions), которые выполняются при обращении к ним из ConcreteCommand.

[!] В описании паттерна применяются общие понятия, такие как Класс, Объект, Абстрактный класс. Применимо к языку Go, это Пользовательский Тип, Значение этого Типа и Интерфейс.

Также в языке Go за место общепринятого наследования используется агрегирование и встраивание.

```
//command.go
// Package command is an example of the Command Pattern.
package command

// Command provides a command interface.
type Command interface {
	Execute() string
}

// ToggleOnCommand implements the Command interface.
type ToggleOnCommand struct {
	receiver *Receiver
}

// Execute command.
func (c *ToggleOnCommand) Execute() string {
	return c.receiver.ToggleOn()
}

// ToggleOffCommand implements the Command interface.
type ToggleOffCommand struct {
	receiver *Receiver
}

// Execute command.
func (c *ToggleOffCommand) Execute() string {
	return c.receiver.ToggleOff()
}

// Receiver implementation.
type Receiver struct {
}

// ToggleOn implementation.
func (r *Receiver) ToggleOn() string {
	return "Toggle On"
}
```

```
// ToggleOff implementation.
func (r *Receiver) ToggleOff() string {
	return "Toggle Off"
}

// Invoker implementation.
type Invoker struct {
	commands []Command
}

// StoreCommand adds command.
func (i *Invoker) StoreCommand(command Command) {
	i.commands = append(i.commands, command)
}

// UnStoreCommand removes command.
func (i *Invoker) UnStoreCommand() {
	if len(i.commands) != 0 {
		i.commands = i.commands[:len(i.commands)-1]
	}
}

// Execute all commands.
func (i *Invoker) Execute() string {
	var result string
	for _, command := range i.commands {
		result += command.Execute() + "\n"
	}
	return result
}
```

```
//command_test.go
package command

import (
	"testing"
)

func TestCommand(t *testing.T) {
```

```
	expect := "Toggle On\n" +
		"Toggle Off\n"

	invoker := &Invoker{}
	receiver := &Receiver{}

	invoker.StoreCommand(&ToggleOnCommand{receiver: receiver})
	invoker.StoreCommand(&ToggleOffCommand{receiver: receiver})

	result := invoker.Execute()

	if result != expect {
		t.Errorf("Expect result to equal %s, but %s.\n", expect, result)
	}
}
```

Revision #1

Created 2022-07-03 10:12:22 UTC by gasick

Updated 2022-07-03 10:13:16 UTC by gasick