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

Фасад (Facade)

Паттерн Facade относится к структурным паттернам уровня объекта.

Паттерн Facade предоставляет высокоуровневый унифицированный интерфейс в виде набора имен методов к набору взаимосвязанных классов или объектов некоторой подсистемы, что облегчает ее использование.

Разбиение сложной системы на подсистемы позволяет упростить процесс разработки, а также помогает максимально снизить зависимости одной подсистемы от другой. Однако использовать такие подсистемы становиться довольно сложно. Один из способов решения этой проблемы является паттерн Facade. Наша задача, сделать простой, единый интерфейс, через который можно было бы взаимодействовать с подсистемами.

В качестве примера можно привести интерфейс автомобиля. Современные автомобили имеют унифицированный интерфейс для водителя, под которым скрывается сложная подсистема. Благодаря применению навороченной электроники, делающей большую часть работы за водителя, тот может с лёгкостью управлять автомобилем, не задумываясь, как там все работает.

Требуется для реализации:

1. Класс Facade предоставляющий унифицированный доступ для классов подсистемы;
2. Класс подсистемы SubSystemA;
3. Класс подсистемы SubSystemB;
4. Класс подсистемы SubSystemC.

Заметьте, что фасад не является единственной точкой доступа к подсистеме, он не ограничивает возможности, которые могут понадобиться "продвинутым" пользователям, желающим работать с подсистемой напрямую.

[!] 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.

```
//facade.go
// Package facade is an example of the Facade Pattern.
package facade

import (
	"strings"
)

// NewMan creates man.
func NewMan() *Man {
	return &Man{
		house: &House{},
		tree:  &Tree{},
		child: &Child{},
	}
}

// Man implements man and facade.
type Man struct {
	house *House
	tree  *Tree
	child *Child
}

// Todo returns that man must do.
func (m *Man) Todo() string {
	result := []string{
		m.house.Build(),
		m.tree.Grow(),
		m.child.Born(),
	}
	return strings.Join(result, "\n")
}
```

```
// House implements a subsystem "House"
type House struct {
}

// Build implementation.
func (h *House) Build() string {
	return "Build house"
}

// Tree implements a subsystem "Tree"
type Tree struct {
}

// Grow implementation.
func (t *Tree) Grow() string {
	return "Tree grow"
}

// Child implements a subsystem "Child"
type Child struct {
}

// Born implementation.
func (c *Child) Born() string {
	return "Child born"
}
```

```
//facade_test.go
package facade

import (
	"testing"
)

func TestFacade(t *testing.T) {

	expect := "Build house\nTree grow\nChild born"

	man := NewMan()

	result := man.Todo()
```

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

Revision #1

Created 2022-07-03 10:31:35 UTC by gasick

Updated 2022-07-03 10:32:24 UTC by gasick