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

Декоратор (Decorator)

Паттерн Decorator относится к структурным паттернам уровня объекта.

Паттерн Decorator используется для расширения функциональности объектов путем динамического добавления объекту новых возможностей. При реализации паттерна используется отношение композиции.

Сущность работы декоратора заключается в обёртывании готового объекта новым функционалом, при этом весь оригинальный интерфейс объекта остается доступным, путем передачи декоратором всех запросов обернутому объекту.

Требуется для реализации:

1. Базовый абстрактный класс Component который предоставляет интерфейс для класса декоратора и компонента;
2. Класс ConcreteDecorator, реализующий интерфейс Component и перезагружающий все методы компонента, по необходимости к ним добавляется функционал;
3. Класс ConcreteComponent реализующий интерфейс Component и который будет обернут декоратором.

При такой структуре нам не важно является ли компонент декоратором или конкретной реализацией, так как интерфейс у них совпадает, и мы можем делать цепочки декораторов. Тем самым динамически менять состояние и поведение объекта.

Я слышал пример с Калсоном и мне он очень понравился. У нас есть Карлсон, мы на него одеваем комбинезон тем самым меняя его состояние, потом на штаны одеваем пропеллер тем самым меняем поведение. Пропеллер в зависимости от ситуации можно снять, изменив поведение на обратное или можно одеть другой комбинезон с другими свойствами.

[!] 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.

```
//decorator.go
// Package decorator is an example of the Decorator Pattern.
package decorator

// Component provides an interface for a decorator and component.
type Component interface {
	Operation() string
}

// ConcreteComponent implements a component.
type ConcreteComponent struct {
}

// Operation implementation.
func (c *ConcreteComponent) Operation() string {
	return "I am component!"
}

// ConcreteDecorator implements a decorator.
type ConcreteDecorator struct {
	component Component
}

// Operation wraps operation of component
func (d *ConcreteDecorator) Operation() string {
	return "<strong>" + d.component.Operation() + "</strong>"
}
```

```
//decorator_test.go
package decorator

import (
	"testing"
)

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

```
    expect := "<strong>I am component!</strong>"

    decorator := &ConcreteDecorator{&ConcreteComponent{}}

    result := decorator.Operation()

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

Revision #1

Created 2022-07-03 10:30:41 UTC by gasick

Updated 2022-07-03 10:31:29 UTC by gasick