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

Посредник (Mediator)

Паттерн Mediator относится к поведенческим паттернам уровня объекта.

Паттерн Mediator предоставляет объект-посредник, скрывающий способ взаимодействия множества других объектов-коллег. Mediator делает систему слабо связанной, избавляя объекты от необходимости ссылаться друг на друга, что позволяет изменять взаимодействие между ними независимо.

Например, у нас есть посредник между заводом производства хлебобулочных изделий, фермером и магазином сбыта. Посредник избавляет фермера от взаимодействия с заводом, который использует его сырье, а завод от взаимодействия с магазином, в который поступает продукция для сбыта.

Требуется для реализации:

1. Интерфейс Mediator - посредник описывающий организацию процесса по обмену информацией между объектами типа Colleague;
2. Класс ConcreteMediator, реализующий интерфейс Mediator;
3. Базовый абстрактный класс Colleague - коллега описывающий организацию процесса взаимодействия объектов-коллег с объектом типа Mediator;
4. Класс ConcreteColleague, реализующий интерфейс Colleague. Каждый объект-коллега знает только об объекте-медиаторе. Все объекты-коллеги обмениваются информацией только через посредника.

[!] 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.

```
//mediator.go
// Package mediator is an example of the Mediator Pattern.
```

```
package mediator

// Mediator provides a mediator interface.
type Mediator interface {
	Notify(msg string)
}

// Тип ConcreteMediator, реализует посредника
type ConcreteMediator struct {
	*Farmer
	*Cannery
	*Shop
}

// Notify implementation.
func (m *ConcreteMediator) Notify(msg string) {
	if msg == "Farmer: Tomato complete..." {
		m.Cannery.AddMoney(-15000.00)
		m.Farmer.AddMoney(15000.00)
		m.Cannery.MakeKetchup(m.Farmer.GetTomato())
	} else if msg == "Cannery: Ketchup complete..." {
		m.Shop.AddMoney(-30000.00)
		m.Cannery.AddMoney(30000.00)
		m.Shop.SellKetchup(m.Cannery.GetKetchup())
	}
}

// ConnectColleagues connects all colleagues.
func ConnectColleagues(farmer *Farmer, cannery *Cannery, shop *Shop) {
	mediator := &ConcreteMediator{
		Farmer:  farmer,
		Cannery: cannery,
		Shop:    shop,
	}

	mediator.Farmer.SetMediator(mediator)
	mediator.Cannery.SetMediator(mediator)
	mediator.Shop.SetMediator(mediator)
}
```

```
// Farmer implements a Farmer colleague
type Farmer struct {
	mediator Mediator
	tomato   int
	money    float64
}

// SetMediator sets mediator.
func (f *Farmer) SetMediator(mediator Mediator) {
	f.mediator = mediator
}

// AddMoney adds money.
func (f *Farmer) AddMoney(m float64) {
	f.money += m
}

// GrowTomato implementation.
func (f *Farmer) GrowTomato(tomato int) {
	f.tomato = tomato
	f.money -= 7500.00
	f.mediator.Notify("Farmer: Tomato complete...")
}

// GetTomato returns tomatos.
func (f *Farmer) GetTomato() int {
	return f.tomato
}

// Cannery implements a Cannery colleague.
type Cannery struct {
	mediator Mediator
	ketchup  int
	money    float64
}

// SetMediator sets mediator.
func (c *Cannery) SetMediator(mediator Mediator) {
	c.mediator = mediator
}
```

```
// AddMoney adds money.
func (c *Cannery) AddMoney(m float64) {
	c.money += m
}

// MakeKetchup implementation.
func (c *Cannery) MakeKetchup(tomato int) {
	c.ketchup = tomato
	c.mediator.Notify("Cannery: Ketchup complete...")
}

// GetKetchup returns ketchup.
func (c *Cannery) GetKetchup() int {
	return c.ketchup
}

// Shop implements a Shop colleague.
type Shop struct {
	mediator Mediator
	money    float64
}

// SetMediator sets mediator.
func (s *Shop) SetMediator(mediator Mediator) {
	s.mediator = mediator
}

// AddMoney adds money.
func (s *Shop) AddMoney(m float64) {
	s.money += m
}

// SellKetchup converts ketchup to money.
func (s *Shop) SellKetchup(ketchup int) {
	s.money = float64(ketchup) * 54.75
}

// GetMoney returns money.
func (s *Shop) GetMoney() float64 {
```

```
	return s.money
}
```

```
//mediator_test.go
package mediator

import (
	"testing"
)

func TestMediator(t *testing.T) {

	farmer := new(Farmer)
	cannery := new(Cannery)
	shop := new(Shop)

	farmer.AddMoney(7500.00)
	cannery.AddMoney(15000.00)
	shop.AddMoney(30000.00)

	ConnectColleagues(farmer, cannery, shop)

	// A farmer grows a 1000kg tomato
	// and informs the mediator about the completion of his work.
	// Next, the mediator sends the tomatoes to the cannery.
	// After the cannery produces 1000 packs of ketchup,
	// he informs the mediator about his delivery to the store.
	farmer.GrowTomato(1000)

	expect := float64(54750)
	result := shop.GetMoney()

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

Revision #1

Created 3 July 2022 10:14:24 by gasick

Updated 3 July 2022 10:15:10 by gasick