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

Посетитель (Visitor)

Паттерн Visitor относится к поведенческим паттернам уровня объекта.

Паттерн Visitor позволяет обойти набор элементов (объектов) с разнородными интерфейсами, а также позволяет добавить новый метод в класс объекта, при этом, не изменяя сам класс этого объекта.

Требуется для реализации:

1. Абстрактный класс Visitor, описывающий интерфейс визитера;
2. Класс ConcreteVisitor, реализующий конкретного визитера. Реализует методы для обхода конкретного элемента;
3. Класс ObjectStructure, реализующий структуру(коллекцию), в которой хранятся элементы для обхода;
4. Абстрактный класс Element, реализующий интерфейс элементов структуры;
5. Класс ElementA, реализующий элемент структуры;
6. Класс ElementB, реализующий элемент структуры.

[!] 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.

```
//visitor.go
// Package visitor is an example of the Visitor Pattern.
package visitor

// Visitor provides a visitor interface.
type Visitor interface {
	VisitSushiBar(p *SushiBar) string
	VisitPizzeria(p *Pizzeria) string
	VisitBurgerBar(p *BurgerBar) string
```

```
}

// Place provides an interface for place that the visitor should visit.
type Place interface {
	Accept(v Visitor) string
}

// People implements the Visitor interface.
type People struct {
}

// VisitSushiBar implements visit to SushiBar.
func (v *People) VisitSushiBar(p *SushiBar) string {
	return p.BuySushi()
}

// VisitPizzeria implements visit to Pizzeria.
func (v *People) VisitPizzeria(p *Pizzeria) string {
	return p.BuyPizza()
}

// VisitBurgerBar implements visit to BurgerBar.
func (v *People) VisitBurgerBar(p *BurgerBar) string {
	return p.BuyBurger()
}

// City implements a collection of places to visit.
type City struct {
	places []Place
}

// Add appends Place to the collection.
func (c *City) Add(p Place) {
	c.places = append(c.places, p)
}

// Accept implements a visit to all places in the city.
func (c *City) Accept(v Visitor) string {
	var result string
	for _, p := range c.places {
```

```
    result += p.Accept(v)
  }
  return result
}

// SushiBar implements the Place interface.
type SushiBar struct {
}

// Accept implementation.
func (s *SushiBar) Accept(v Visitor) string {
  return v.VisitSushiBar(s)
}

// BuySushi implementation.
func (s *SushiBar) BuySushi() string {
  return "Buy sushi..."
}

// Pizzeria implements the Place interface.
type Pizzeria struct {
}

// Accept implementation.
func (p *Pizzeria) Accept(v Visitor) string {
  return v.VisitPizzeria(p)
}

// BuyPizza implementation.
func (p *Pizzeria) BuyPizza() string {
  return "Buy pizza..."
}

// BurgerBar implements the Place interface.
type BurgerBar struct {
}

// Accept implementation.
func (b *BurgerBar) Accept(v Visitor) string {
  return v.VisitBurgerBar(b)
```

```
}

// BuyBurger implementation.
func (b *BurgerBar) BuyBurger() string {
	return "Buy burger..."
}
```

```
//visitor_test.go
package visitor

import (
	"testing"
)

func TestVisitor(t *testing.T) {

	expect := "Buy sushi...Buy pizza...Buy burger..."

	city := new(City)

	city.Add(&SushiBar{})
	city.Add(&Pizzeria{})
	city.Add(&BurgerBar{})

	result := city.Accept(&People{})

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

Revision #1

Created 2022-07-03 10:21:01 UTC by gasick

Updated 2022-07-03 10:21:37 UTC by gasick