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

????????? (Strategy)

Паттерн Strategy относится к поведенческим паттернам уровня объекта.

Паттерн Strategy определяет набор алгоритмов схожих по роду деятельности, инкапсулирует их в отдельный класс и делает их подменяемыми. Паттерн Strategy позволяет подменять алгоритмы без участия клиентов, которые используют эти алгоритмы.

Требуется для реализации:

1. Класс Context, представляющий собой контекст выполнения той или иной стратегии;
2. Абстрактный класс Strategy, определяющий интерфейс различных стратегий;
3. Класс ConcreteStrategyA, реализует одну из стратегий представляющую собой алгоритмы, направленные на достижение определенной цели;
4. Класс ConcreteStrategyB, реализует одно из стратегий представляющую собой алгоритмы, направленные на достижение определенной цели.

[!] 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.

```
//strategy.go
// Package strategy is an example of the Strategy Pattern.
package strategy

// StrategySort provides an interface for sort algorithms.
type StrategySort interface {
Sort([]int)
}

// BubbleSort implements bubble sort algorithm.
type BubbleSort struct {
```

```
}

// Sort sorts data.
func (s *BubbleSort) Sort(a []int) {
	size := len(a)
	if size < 2 {
		return
	}
	for i := 0; i < size; i++ {
		for j := size - 1; j >= i+1; j-- {
			if a[j] < a[j-1] {
				a[j], a[j-1] = a[j-1], a[j]
			}
		}
	}
}

// InsertionSort implements insertion sort algorithm.
type InsertionSort struct {
}

// Sort sorts data.
func (s *InsertionSort) Sort(a []int) {
	size := len(a)
	if size < 2 {
		return
	}
	for i := 1; i < size; i++ {
		var j int
		var buff = a[i]
		for j = i - 1; j >= 0; j-- {
			if a[j] < buff {
				break
			}
			a[j+1] = a[j]
		}
		a[j+1] = buff
	}
}
```

```
// Context provides a context for execution of a strategy.
type Context struct {
	strategy StrategySort
}

// Algorithm replaces strategies.
func (c *Context) Algorithm(a StrategySort) {
	c.strategy = a
}

// Sort sorts data according to the chosen strategy.
func (c *Context) Sort(s []int) {
	c.strategy.Sort(s)
}
```

```
//strategy_test.go
package strategy

import (
	"strconv"
	"testing"
)

func TestStrategy(t *testing.T) {

	data1 := []int{8, 2, 6, 7, 1, 3, 9, 5, 4}
	data2 := []int{8, 2, 6, 7, 1, 3, 9, 5, 4}

	ctx := new(Context)

	ctx.Algorithm(&BubbleSort{})

	ctx.Sort(data1)

	ctx.Algorithm(&InsertionSort{})

	ctx.Sort(data2)

	expect := "1,2,3,4,5,6,7,8,9,"
```

```
	var result1 string
	for _, val := range data1 {
		result1 += strconv.Itoa(val) + ","
	}

	if result1 != expect {
		t.Errorf("Expect result1 to equal %s, but %s.\n", expect, result1)
	}

	var result2 string
	for _, val := range data2 {
		result2 += strconv.Itoa(val) + ","
	}

	if result2 != expect {
		t.Errorf("Expect result2 to equal %s, but %s.\n", expect, result2)
	}
}
```

Revision #1

Created 2022-07-03 10:17:54 UTC by gasick

Updated 2022-07-03 10:19:37 UTC by gasick