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

???????? (Iterator)

Паттерн Iterator относится к поведенческим паттернам уровня объекта.

Паттерн Iterator предоставляет механизм обхода коллекций объектов не раскрывая их внутреннего представления.

Зачастую этот паттерн используется вместо массива объектов, чтобы не только предоставить доступ к элементам, но и наделить некоторой логикой.

Iterator представляет собой общий интерфейс, позволяющий реализовать произвольную логику итераций.

Требуется для реализации:

1. Интерфейс Iterator описывающий набор методов для доступа к коллекции;
2. Класс ConcreteIterator, реализующий интерфейс Iterator. Следит за позицией текущего элемента при переборе коллекции (Aggregate).;
3. Интерфейс Aggregate описывающий набор методов коллекции объектов;
4. Класс ConcreteAggregate, реализующий интерфейс Aggregate и хранящий в себе элементы коллекции.

[!] 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.

```
//iterator.go
// Package iterator is an example of the Iterator Pattern.
package iterator

// Iterator provides a iterator interface.
type Iterator interface {
	Index() int
```

```
	Value() interface{}
	Has() bool
	Next()
	Prev()
	Reset()
	End()
}

// Aggregate provides a collection interface.
type Aggregate interface {
	Iterator() Iterator
}

// BookIterator implements the Iterator interface.
type BookIterator struct {
	shelf    *BookShelf
	index    int
	internal int
}

// Index returns current index
func (i *BookIterator) Index() int {
	return i.index
}

// Value returns current value
func (i *BookIterator) Value() interface{} {
	return i.shelf.Books[i.index]
}

// Has implementation.
func (i *BookIterator) Has() bool {
	if i.internal < 0 || i.internal >= len(i.shelf.Books) {
		return false
	}
	return true
}

// Next goes to the next item.
func (i *BookIterator) Next() {
```

```
	i.internal++
	if i.Has() {
		i.index++
	}
}

// Prev goes to the previous item.
func (i *BookIterator) Prev() {
	i.internal--
	if i.Has() {
		i.index--
	}
}

// Reset resets iterator.
func (i *BookIterator) Reset() {
	i.index = 0
	i.internal = 0
}

// End goes to the last item.
func (i *BookIterator) End() {
	i.index = len(i.shelf.Books) - 1
	i.internal = i.index
}

// BookShelf implements the Aggregate interface.
type BookShelf struct {
	Books []*Book
}

// Iterator creates and returns the iterator over the collection.
func (b *BookShelf) Iterator() Iterator {
	return &BookIterator{shelf: b}
}

// Add adds an item to the collection.
func (b *BookShelf) Add(book *Book) {
	b.Books = append(b.Books, book)
}
```

```
// Book implements a item of the collection.
type Book struct {
	Name string
}
```

```
//iterator_test.go
package iterator

import (
	"testing"
)

func TestIterator(t *testing.T) {

	shelf := new(BookShelf)

	books := []string{"A", "B", "C", "D", "E", "F"}

	for _, book := range books {
		shelf.Add(&Book{Name: book})
	}

	for iterator := shelf.Iterator(); iterator.Has(); iterator.Next() {
		index, value := iterator.Index(), iterator.Value().(*Book)
		if value.Name != books[index] {
			t.Errorf("Expect Book.Name to %s, but %s", books[index], value.Name)
		}
	}
}
```

Revision #1

Created 2022-07-03 10:13:24 UTC by gasick

Updated 2022-07-03 10:14:18 UTC by gasick