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

Компоновщик (Composite)

Паттерн Composite относится к структурным паттернам уровня объекта.

Паттерн Composite группирует схожие объекты в древовидные структуры.

Для построения дерева будут использоваться массивы, представляющие ветви дерева.

Требуется для реализации:

1. Базовый абстрактный класс Component который предоставляет интерфейс, как для ветвей, так и для листьев дерева;
2. Класс Composite, реализующий интерфейс Component и являющийся ветвью дерева;
3. Класс Leaf, реализующий интерфейс Component и являющийся листом дерева.

Обратите внимание, что лист дерева является классом листовых узлов и не может иметь потомков (из листа не может вырасти ветвь или другой лист).

Ветви дерева задают поведение объектов, входящих в структуру дерева, у которых есть потомки, а также сами хранит в себе компоненты дерева. Другим словами ветви могут содержать другие ветви и листья.

Основным назначением паттерна, является обеспечение единого интерфейса как к составному (ветви) так и конечному (листу) объекту, что бы клиент не задумывался над тем, с каким объектом он работает.

[!] 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.

```
//composite.go
// Package composite is an example of the Composite Pattern.
```

```
package composite

// Component provides an interface for branches and leaves of a tree.
type Component interface {
	Add(child Component)
	Name() string
	Child() []Component
	Print(prefix string) string
}

// Directory implements branches of a tree
type Directory struct {
	name   string
	childs []Component
}

// Add appends an element to the tree branch.
func (d *Directory) Add(child Component) {
	d.childs = append(d.childs, child)
}

// Name returns name of the Component.
func (d *Directory) Name() string {
	return d.name
}

// Child returns child elements.
func (d *Directory) Child() []Component {
	return d.childs
}

// Print returns the branche in string representation.
func (d *Directory) Print(prefix string) string {
	result := prefix + "/" + d.Name() + "\n"
	for _, val := range d.Child() {
		result += val.Print(prefix + "/" + d.Name())
	}
	return result
}
```

```
// File implements a leaves of a tree
type File struct {
	name string
}

// Add implementation.
func (f *File) Add(child Component) {
}

// Name returns name of the Component.
func (f *File) Name() string {
	return f.name
}

// Child implementation.
func (f *File) Child() []Component {
	return []Component{}
}

// Print returns the leave in string representation.
func (f *File) Print(prefix string) string {
	return prefix + "/" + f.Name() + "\n"
}

// NewDirectory is constructor.
func NewDirectory(name string) *Directory {
	return &Directory{
		name: name,
	}
}

// NewFile is constructor.
func NewFile(name string) *File {
	return &File{
		name: name,
	}
}
```

```
//composite_test.go
package composite
```

```
import (
	"testing"
)

func TestComposite(t *testing.T) {

	expect := "/root\n/root/usr\n/root/usr/B\n/root/A\n"

	rootDir := NewDirectory("root")

	usrDir := NewDirectory("usr")
	fileA := NewFile("A")

	rootDir.Add(usrDir)
	rootDir.Add(fileA)

	fileB := NewFile("B")

	usrDir.Add(fileB)

	result := rootDir.Print("")

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

Revision #1
Created 2022-07-03 10:29:50 UTC by gasick
Updated 2022-07-03 10:30:34 UTC by gasick