

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

Состояние (State)

Паттерн State относится к поведенческим паттернам уровня объекта.

Паттерн State позволяет объекту изменять свое поведение в зависимости от внутреннего состояния и является объектно-ориентированной реализацией конечного автомата.

Поведение объекта изменяется настолько, что создается впечатление, будто изменился класс объекта.

Паттерн должен применяться:

- когда поведение объекта зависит от его состояния
- поведение объекта должно изменяться во время выполнения программы
- состояний достаточно много и использовать для этого условные операторы, разбросанные по коду, достаточно затруднительно

Требуется для реализации:

1. Класс Context, представляет собой объектно-ориентированное представление конечного автомата;
2. Абстрактный класс State, определяющий интерфейс различных состояний;
3. Класс ConcreteStateA реализует одно из поведений, ассоциированное с определенным состоянием;
4. Класс ConcreteStateB реализует одно из поведений, ассоциированное с определенным состоянием.

[!] 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 вместо наследования используется композиция.

```
//state.go  
  
// Package state is an example of the State Pattern.
```

package state

// MobileAlertStater provides a common interface for various states.

type MobileAlertStater interface {

Alert() string

}

// MobileAlert implements an alert depending on its state.

type MobileAlert struct {

state MobileAlertStater

}

// Alert returns a alert string

func (a *MobileAlert) Alert() string {

return a.state.Alert()

}

// SetState changes state

func (a *MobileAlert) SetState(state MobileAlertStater) {

a.state = state

}

// NewMobileAlert is the MobileAlert constructor.

func NewMobileAlert() *MobileAlert {

return &MobileAlert{state: &MobileAlertVibration{}}

}

// MobileAlertVibration implements vibration alert

type MobileAlertVibration struct {

}

// Alert returns a alert string

func (a *MobileAlertVibration) Alert() string {

return "Vrrr... Brrr... Vrrr..."

}

// MobileAlertSong implements beep alert

type MobileAlertSong struct {

}

```
// Alert returns a alert string
func (a *MobileAlertSong) Alert() string {
    return "Белые розы, Белые розы. Беззащитны шипы..."
}
```

```
//state_test.go
package state

import (
    "testing"
)

func TestState(t *testing.T) {

    expect := "Vrrr... Brrr... Vrrr..." +
        "Vrrr... Brrr... Vrrr..." +
        "Белые розы, Белые розы. Беззащитны шипы..."

    mobile := NewMobileAlert()

    result := mobile.Alert()
    result += mobile.Alert()

    mobile.SetState(&MobileAlertSong{ })

    result += mobile.Alert()

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

Revision #1

Created 3 July 2022 10:17:05 by gasick

Updated 3 July 2022 10:17:50 by gasick