Покоряем golang

Лекция 3. Объектно-ориентированное программирование

Григорий Базилевич

04 декабря 2021 года

Как вообще объявляются типы?

```
//type *name* *type*

type MyInt int
type PrefixerFunc func(string) error

prefixer := func(prefix string) PrefixerFunc {
    return func(in string) error {
        fmt.Printf("[%s] %s", prefix , in)
        return nil
    }
}
```

Структуры в golang

```
type Person struct {
    Id int
    Name string
    Address string
}

type Account struct {
    Id int
    Name string
    SomeFunc func(string) string
    Owner Person
}
```

Структуры в golang

```
var acc = Account{
    Id: 1,
    Name: "admin",
}

acc.Owner = Person{7, "James", "London"}
```

Вложенные структуры

```
type Person struct {
    Id int
    Name string
    Address string
}

type Account struct {
    Id int
    Name string
    Owner Person
    Person
}

var acc Account
fmt.Printf("%#v\n", acc)
fmt.Println(acc.Adress)
fmt.Println(acc.Name, acc.Person.Name)
```

Методы структур

```
type Person struct {
    Id int
    Name string
}

func (p Person) UpdateName(n string) {
    p.Name = n
}

func (p *Person) SetName(n string) {
    p.Name = n
}
```

Продвинутые методы

```
type VectorInt []int

func (v *VectorInt) PushBack(x int) {
    *v = append(*v, x)
}

func (v *VectorInt) Size() int {
    return len(*v)
}

v := VectorInt([]int{1, 2, 3})
v.Add(5)
v.Add(4)

// [1 2 3 5 4] 5
fmt.Println(v, v.Size())
```

Интерфейсы

```
type Payer interface {
    Pay(int) error
}
```

Интерфейсы

```
type Wallet struct {
    Cash int
}

func (w *Wallet) Pay(x int) error {
    if w.Cash < x {
        return fmt.Errorf("not enough cash")
    }
    w.Cash -= x
    return nil
}
```

Интерфейсы

```
type Card struct {
    Balance int
    Number int
    CVV int
    CardHolder string
    ValidUntil time.Time
}

func (c *Card) Pay(x int) error {
    if w.Balance < x {
        return fmt.Errorf("not enough money")
    }
    w.Balance -= x
    return nil
}
```

Интерфейсы

```
type ApplePay struct {
    Balance int
    AppleAccountId int
}

func (c *Card) Pay(x int) error {
    if w.Balance < x {
        return fmt.Errorf("not enough money")
    }
    w.Balance -= x
    return nil
}
```

Интерфейсы

```
func Buy(p Payer) {
    err := p.Pay(20)
    if err != nil {
        fmt.Printf("Failed. %v %T\n", err, p)
        return
    }
    fmt.Printf("Thank you! %T\n", p)
}

myWallet := &Wallet{100}
Buy(myWallet)

var myMoney Payer
myMoney = &Card{Balance: 100, CardHolder: "ch"}
Buy(myMoney)
myMoney = &ApplePay{Balance: 9}
Buy(myMoney)
```

Интерфейсы

```
func Greet(p Payer) {
    switch p.(type) {
    case *Wallet:
        fmt.Println("Cash?")
    case *Card:
        plasticCard, ok := p.(*Card)
        if !ok {
            panic()
        }
        name := plasticCard.CardHolder
        fmt.Printf("Hello, %v \n", name)
    default:
        fmt.Println("Something new!")
    }
}
```

Пустой интерфейс

```
func CheckAndBuy(in interface{}) error {
    var p Payer
    if p, ok := in.(Payer); !ok {
        return fmt.Errorf("It's not payer!")
    }

    Greet(p)
    Buy(p)
}
```

Композиция интерфейсов

```
type Payer interface {
    Pay(int) error
}

type Ringer interface {
    Ring(string) error
}

type Phone interface {
    Payer
    Ringer
    Reboot() error
}
```

Система пакетов

```
/bin/
/pkg/
/src/
   github.com/mrfoxygmfr/hello-world/
      cmd/
         help.go
      utils/
         models/
            user.go
         database.go
      go.mod
      main.go
```

Шаблон пакета

```
github.com/mrfoxygmfr/hello-world/
    build/
    cmd/
    configs/
    internal/
    scripts/
    pkg/
```

https://github.com/golang-standards/project-layout