

GO-04 05: Структуры и интерфейсы.

Наследование в Go. Композиция

Описание:

Поговорим о наследовании в Go. Тут нет классического наследования из ООП, в Go используется композиция: то есть, один объект (структура или интерфейс) может выступать в роли части другого объекта - встраиваться (embed) в него. В итоге при создании инстанса одного объекта сразу создается и встроенный в него, при этом мы можем обращаться к его полям, как если бы они были полями родительского. Рассмотрим на примере.

Предположим, что у нас есть структура Card:

```
type Card struct {  
    Balance    string  
    ExpiredDate string  
    CVV        int  
    Num        int64  
    Owner      string  
}
```

А теперь предположим, что у нас есть сущность Кредитные карты, у которых есть еще свойство, которое хранит лимит:

```
type CreditCard struct {  
    Card  
    Limit int  
}
```

Мы можем работать с инстансом объекта CreditCard так, как если бы непосредственно у него был весь набор свойств структуры Card:

```
package main
```

```
import "fmt"
```

```
type Card struct {  
    Balance    int  
    ExpiredDate string  
    CVV        int  
    Num        int64
```

Куплено
благодаря
Skladchik.com

```

    Owner    string
}

type CreditCard struct {
    *Card
    Limit int
}

func main() {
    cc := &CreditCard{
        &Card{
            Balance:    10000,
            ExpiredDate: "01.02.2023",
            CVV:        132,
            Num:        4234536475474656,
            Owner:     "Vasily Ivanov",
        },
        100000,
    }

    fmt.Printf("%+v\n", cc)

    fmt.Printf("Limit: %d, Owner: %s", cc.Limit, cc.Owner)

    cc.Owner = "Oleg"
    cc.Limit = 0

    fmt.Printf("Limit: %d, Owner: %s", cc.Limit, cc.Owner)
}

```

394783

```
go run main.go
```

Output:

```
&{Card:0xc000116040 Limit:100000}
```

```
Limit: 100000, Owner: Vasily Ivanov
```

```
Limit: 0, Owner: Oleg
```

Но при этом мы видим, что формально внутри находится другой объект, который встроен в экземпляр CreditCard:

В Go есть также возможность встраивать интерфейсы:

```
type Buyer interface {
    buy()
}

type Seller interface {
    sell()
}

type Dealer interface {
    Buyer
    Seller
}
```

Интерфейс Dealer требует реализации методов buy() и sell().

Обратите внимание:

Начиная с Go 1.14, допускается, чтобы встроенные интерфейсы имели перекрывающиеся наборы методов. То есть, если бы наши интерфейсы Buyer и Seller имели хотя бы один схожий метод, то даже при их одновременном встраивании никаких конфликтов не возникнет (при условии, что сигнатуры дублирующихся методов не отличаются), ранее это было недопустимо.

Также в Go возможно встраивание интерфейсов в структуру, что дает возможность построить конкретную реализацию и завязаться только на поведение (при этом, соответственно, поля встроенной структуры будут недоступны, в отличие от методов), рассмотрим на примере:

```
package main

import (
    "fmt"
    "time"
)

type Checker interface {
    CheckDate() bool
}

type Card struct {
    Balance      int
    ExpiredDate  string
    CVV          int
    Num          int64
    Owner        string
}
```

```

}

func (c *Card) CheckDate() bool {
    ex, _ := time.Parse("2006.01.02", c.ExpiredDate)
    return time.Now().Before(ex)
}

type CreditCard struct {
    Checker
    Limit int
}

func main() {
    c := &Card{
        Balance:      10000,
        ExpiredDate:  "01.02.2023",
        CVV:          132,
        Num:          4234536475474656,
        Owner:        "Vasily Ivanov",
    }
    cc := &CreditCard{
        c,
        100000,
    }

    fmt.Printf("%t", cc.CheckDate())
}

```

```
go run main.go
```

Output:

```
false
```

Но если мы обратимся к полю структуры Card, то получим:

```
cc.Owner = "Dima"
```

```
go run main.go
```

Output:

```
./main.go:42:4: cc.Owner undefined (type *CreditCard has no field or
method Owner)
exit status 2
```

Process exiting with code: 1

Полезные ссылки:

- [Embedding](#)
- [Proposal: Permit embedding of interfaces with overlapping method sets](#)
- [Object composition](#)

Задание:

1. Создайте в своем проекте module04 из ветки module04_04 - ветку module04_05.
2. Создайте структуру Overduer, в которую вынесите из структуры Customer поля balance и debt.
3. Сделайте так, чтобы Customer продолжал реализовывать интерфейс Debtor, но при этом поля balance и debt из него были не доступны напрямую.

Внимание!: Логика WroffDebt() (второе задание модуля) меняться не должна.

4. В ответе пришлите ссылку на MR ветки module04_05 с нужными правками в ветку master своего проекта.