

# GO-04 03: Структуры и интерфейсы.

## Интерфейсы и утиная типизация

### Описание:

Теперь поговорим об интерфейсах. Интерфейсы позволяют вызывающей стороне привязываться не к реализации, а только к определенному поведению (набору методов). Особенность Go в том, что в отличие от ООП языков в нем нет явного указания, какой интерфейс реализует наша структура. Достаточно просто удовлетворять набору и сигнатуре методов интерфейса - это называется утиной типизацией.

Возьмем уже известный нам проект с сущностью Customer и объявим в пакете internal интерфейс Debtor, описывающий сигнатуру метода WroffDebt (метод списания долга из GO-04 02: Структуры и интерфейсы. Методы структур):

```
type Debtor interface {  
    WroffDebt() error  
}
```

То есть, теперь все структуры, имеющие данный набор методов (в нашем случае - WroffDebt), удовлетворяют данному интерфейсу.

Теперь давайте объявим функцию startTransaction в нашем пакете main:

cmd/myapp/main.go

```
package main
```

```
import (  
    "fmt"  
    "myapp/internal"  
)
```

```
func main() {  
    cust := internal.NewCustomer("Dmitry", 23, 10000, 1000, true)  
  
    startTransaction(cust)
```

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

```
func startTransaction(debtor internal.Debtor) error {  
    return debtor.WroffDebt()
```

```
}
```

Функция `startTransaction` может принимать любую структуру, удовлетворяющую интерфейсу `Debtor`, то есть завязана только на определенное ожидаемое поведение. Создадим структуру `Partner` в пакете `internal` со своей реализацией метода `WroffDebt` (допустим, что партнерам мы просто обнуляем долг):

```
internal/partner.go
```

```
package internal
```

```
type Partner struct {
    Name    string
    Age     int
    balance int
    debt    int
}
```

```
func (c *Partner) WroffDebt() error {
    c.debt = 0

    return nil
}
```

```
func NewPartner(name string, age int, balance int, debt int) *Partner
{
    return &Partner{
        Name:    name,
        Age:     age,
        balance: balance,
        debt:    debt,
    }
}
```

Наша новая структура также удовлетворяет интерфейсу `Debtor` (его, кстати, лучше теперь вынести в отдельный файл - допустим, `interfaces.go`), что позволяет нам передать экземпляр объекта `Partner` в нашу функцию `startTransaction`:

```
cmd/myapp/main.go
```

```
package main
```

```
import (
    "fmt"
```

```

    "myapp/internal"
)

func main() {
    partner := internal.NewPartner("Dmitry", 23, 10000, 1000)

    startTransaction(partner)

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

func startTransaction(debtor internal.Debtor) error {
    return debtor.WrOffDebt()
}

go run main.go

```

Output:

```
&{Name:Dmitry Age:23 balance:10000 debt:0}
```

Обратите внимание:

В задании GO-04 02: Структуры и интерфейсы. Методы структур мы говорили о том, что, чтобы вызвать метод, привязанный к значению, а не указателю, нам необязательно создавать объект по значению, а вот обратное утверждение не совсем верно. И вот почему:

```

package main

import (
    "fmt"
)

type Printer interface {
    print()
}

type A struct {}

func (a *A) print() {
    fmt.Println("Hello, playground")
}

```

```
func main() {  
    instance := A{  
  
    var p Printer  
    p = instance  
    p.print()  
}
```

```
go run main.go
```

```
./prog.go:21:4: cannot use instance (type A) as type Printer in  
assignment:
```

```
    A does not implement Printer (print method has pointer receiver)
```

Как мы видим - в такой ситуации мы ловим панику. Подробнее об этом поведении - [Method sets](#) и [Pointers vs values](#)

## Полезные ссылки:

- [Interfaces](#)
- [Duck typing](#)

## Задание:

1. Создайте в своем проекте module04 из ветки module04\_02 - ветку module04\_03.
2. Добавьте в пакет internal интерфейс Discounter, который требует следующей сигнатуры метода:

```
CalcDiscount() (int, error)
```

3. Измените функцию CalcPrice так, чтобы на вход она теперь принимала объект, реализующий интерфейс Discounter.
4. В ответе пришлите ссылку на MR ветки module04\_03 с нужными правками в ветку master своего проекта.