GO-04 02: Структуры и интерфейсы. Методы структур

Описание:

В предыдущем задании мы рассмотрели базовое понятие структур в Go. Теперь поговорим о методах: в отличие от обычных функций метод имеет в своей сигнатуре получателя:

```
func (имя_получателя тип_получателя) имя_метода (параметры)
```

То есть, мы привязываем некоторое поведение к нашей структуре на этапе компиляции (в отличие от ситуации, когда внешний код инжектит функцию к соотвествующему полю структуры в рантайме, как в предыдущем задании с полем CalcDiscount).
Теперь мы можем вызывать от имени экземляра нашей структуры соотвествующий метод (ниже будет пример).
Давайте возьмем наш пример (структура Customer) и предположим, что теперь нам нужно каким-то образом списывать сумму задолженности с баланса пользователя. Давайте сформируем требования:

- необходимо обеспечить атомарность операции, то есть, если сумма списалась с поля Debt, должна списаться и с поля Balance;
- необходимо проверить, присуствует ли на балансе сумма, необходимая для списания долга.

Оптимальным вариантом будет реализовать какую-то функцию с соответствующей логикой, но при этом эта функция - это чистой воды реализация внутренней логики структуры Customer. Добавим соответствующий метод.
internal/customer.go

```
package internal

import (
	"errors"
)

type Customer struct {
	Name         string
	Age          int
	balance      int
	debt         int
	Discount     bool
	CalcDiscount func() (int, error)
```

```
}

func (c Customer) WrOffDebt() (Customer, error) {
	if c.debt >= c.balance {
		return c, errors.New("Not possible write off")
	}

	c.balance -= c.debt
	c.debt = 0

	return c, nil
}

func NewCustomer(name string, age int, balance int, debt int, discount
bool) Customer {
	return Customer{
		Name:     name,
		Age:      age,
		balance:  balance,
		debt:     debt,
		Discount: discount,
	}
}
```

Давайте попробуем вызвать WrOffDebt:
cmd/myapp/main.go

```
package main

import (
	"fmt"
	"myapp/internal"
)

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

	cust, _ = cust.WrOffDebt()

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

```
}

go run main.go

Output:
{Name:Dmitry Age:23 balance:9000 debt:0 Discount:true
CalcDiscount:<nil>}
```

Как мы видим, в результате выполнения метода мы получили новый объект Customer с измененным полем balance. Также вы можете заметить, что поля balance и debt у нас теперь видны только внутри пакета (напомним, что любые объекты, начинающиеся с маленькой буквы, видны только внутри пакета), тем самым мы запрещаем работать с ними напрямую извне, единственный способ повлиять на них - метод WrOffDebt.
В нашем примере мы выполняем метод, который возвращает копию нашей структуры, из-за чего нам необходимо переприсвоить возвращаемое значение. Но давайте посмотрим, как еще мы можем работать с методами, которые занимаются мутацией основного объекта. Объявим наш метод чуть-чуть по-другому: internal/customer.go

```
package internal

import (
	"errors"
)

type Customer struct {
	Name         string
	Age          int
	balance      int
	debt         int
	Discount     bool
	CalcDiscount func() (int, error)
}

func (c *Customer) WrOffDebt() error {
	if c.debt >= c.balance {
		return errors.New("Not possible write off")
	}

	c.balance -= c.debt
	c.debt = 0

	return nil
```

```
}

func NewCustomer(name string, age int, balance int, debt int, discount
bool) *Customer {
	return &Customer{
		Name:     name,
		Age:      age,
		balance:  balance,
		debt:     debt,
		Discount: discount,
	}
}
```

cmd/myapp/main.go

```
package main

import (
	"fmt"
	"myapp/internal"
)

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

	cust.WrOffDebt()

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

go run main.go

Output:
&{Name:Dmitry Age:23 balance:9000 debt:0 Discount:true
CalcDiscount:<nil>}
```

Как мы видим, мы присвоили метод указателю на структуру, тем самым все действия внутри метода влияют на основную структуру, что избавляет нас от лишних присваиваний. Обратите внимание:

Для того чтобы вызвать метод, привязанный к значению, а не указателю, нам не обязательно создавать объект по значению:

```
package main

import (
	"fmt"
)

type A struct {}

func (a A) SayHello() {
	fmt.Println("Hello, world")
}

func main() {
	instance1 := &A{}
	instance1.SayHello()

	instance2 := A{}
	instance2.SayHello()
}

go run main.go

Output:
Hello, world
Hello, world
```

Обратное утверждение не совсем верно и будет рассмотрено в задании GO-04 03: Структуры и интерфейсы. Интерфейсы и утиная типизация

Полезные ссылки:

- Method sets

Задание:

1. Создайте в своем проекте module04 из ветки module04_01 - ветку module04_02.
2. Предположим, что теперь нам необходимо не позволить внешнему коду предоставлять нам реализацию свойства CalcDiscount, а жестко задать

реализацию в привязке к структуре Customer, для этого сделайте CalcDiscount методом, а не свойством структуры Customer:
    - Логику оставить, как была в функции CalcDiscount.
    - Константу DEFAULT_DISCOUNT перенести в пакет internal.
    - Свойство Discount сделать нередактируемым вне пакета internal.
3. В ответе пришлите ссылку на MP ветки module04_02 с нужными правками в ветку master своего проекта.