GO-04 01: Структуры и интерфейсы. Структуры в GO

Описание:

В этом модуле мы с вами разберем работу со структурами, интерфейсами и методами, рассмотрим особенности наследования и типизации в языке Go.
Итак, начнем со структур. Структура - это сгруппированный в один объект набор полей/свойств. Кто знаком с понятием структур в C, думаю, сразу поймет, о чем речь. Предположим, что у нас есть сущность Покупатель и что у него есть фиксированный набор свойств (имя, возраст, баланс и т.д.). Таких покупателей много, и в контексте программы мы как-то с ними работаем - для этого и существуют структуры, мы объявляем, какими свойствами будут обладать объекты и создаем по мере необходимости конкретные экземпляры этих объектов.
Рассмотрим на примере:
Для начала подготовим наш layout (GO-03 03: Модули и пакеты. layout проекта)

```
|   go.mod
|
+---cmd
|   \---myapp
|           main.go
|
\---internal
        customer.go
```

В пакете internal в файле internal/customer.go объявим структуру Customer:

```
package internal

type Customer struct {
    Name string
    Age int
    Balance int
    Debt int
    Discount bool
    CalcDiscount func() (int, error)
}
```

Получается, что теперь мы можем создавать экземпляры объекта Customer с фиксированным набором свойств.

Поподробнее остановимся на поле CalcDiscount - значением данного поля может являться любая функция, удовлетворяющая набору аргументов + возвращаемое значение.
В данном конкретном примере будем считать, что для различных кастомеров могут быть различные функции расчета скидки, то есть вызывающая сторона может инжектить различные реализации функции. Чуть ниже рассмотрим на примере.
Способы создания объекта, содержащего данную структуру:

```
var cust internal.Customer

cust := new(internal.Customer)

cust := internal.Customer{}
```

Давайте в нашем файле main.go создадим экземпляр данной структуры:

```
package main

import (
    "fmt"
    "myapp/internal"
)

func main() {
    var cust internal.Customer

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

Output:

```
~$ go run ./main.go
{Name: Age:0 Balance:0 Debt:0 Discount:false CalcDiscount:<nil>}
```

Мы можем видеть, что поля структуры проинициализировались значениями по умолчанию, что говорит о том, что память под нее выделилась сразу при создании экземпляра.
Такой же результат мы бы получили, если бы объявили структуру 3-м способом:

```
cust := internal.Customer{}
```

Но в случае такого объявления мы сразу можем проинициализировать значения:

```
package main

import (
    "fmt"
```

```
    "myapp/internal"
)

func main() {
    cust := internal.Customer{
        Age:     23,
        Balance: 10000,
        Debt:    1000,
        Name:    "Dmitry",
    }

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

Output:

```
~$ go run ./main.go
{Name:Dmitry Age:23 Balance:10000 Debt:1000 Discount:false
CalcDiscount:<nil>}
```

Итак, мы видим, что поля структуры заполнены (функцию CalcDiscount мы еще не определяли, поэтому пока nil).
В процессе работы мы можем переопределять значения полей экземпляра структуры:

```
package main

import (
    "fmt"
    "myapp/internal"
)

func main() {
    cust := internal.Customer{
        Age:     23,
        Balance: 10000,
        Debt:    1000,
        Name:    "Dmitry",
    }

    cust.Name = "Petya"

    fmt.Printf(cust.Name)
```

```
}
```

Output:

```
~$ go run ./main.go
Petya
```

Теперь давайте попробуем встроить в наш экземпляр анонимную функцию расчета скидки:

```
package main

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

const DEFAULT_DISCOUNT = 500

func main() {
    cust := internal.Customer{
        Age:     23,
        Balance: 10000,
        Debt:    1000,
        Name:    "Dmitry",
    }

    cust.CalcDiscount = func() (int, error) {
        if !cust.Discount {
            return 0, errors.New("Discount not available")
        }
        result := DEFAULT_DISCOUNT - cust.Debt
        if result < 0 {
            return 0, nil
        }
        return result, nil
    }

    discount, _ := cust.CalcDiscount()

    fmt.Printf("%d", discount)
```

```
}
```

Итак, предположим, что скидка зависит от какой-то константы DEFAULT_DISCOUNT и остатка долга кастомера (если долг большой, то скидки не будет).
Output:

```
~$ go run ./main.go
0
```

Также вы можете заметить, как объекты родительской области видимости (DEFAULT_DISCOUNT и cust) замкнулись на нашу функцию (см. GO-02 05: Основы языка. Конструкции языка и функции).
Что касается инициализации начальных значений, есть еще один способ - функция конструктор:
Файл internal/customer.go:

```
package internal

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

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,
    }
}
```

Файл main.go:

```
package main

import (
    "errors"
```

```
    "fmt"
    "myapp/internal"
)

const DEFAULT_DISCOUNT = 500

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

    cust.CalcDiscount = func() (int, error) {
        if !cust.Discount {
            return 0, errors.New("Discount not available")
        }
        result := DEFAULT_DISCOUNT - cust.Debt
        if result < 0 {
            return 0, nil
        }
        return result, nil
    }

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

Output:

```
~$ go run ./main.go
&{Name:Dmitry Age:23 Balance:10000 Debt:1000 Discount:true
CalcDiscount:0x4bf360}
```

Обратите внимание, что наш конструктор возвращает не копию экземпляра, а указатель на него. Так тоже можно - в данной ситуации нет смысла создавать структуру, а потом копировать ее как возвращаемое значение (лишняя аллокация памяти), поэтому тут можно просто вернуть указатель. В этом примере это не особо важно, но на больших структурах возможен выигрыш в производительности.
Где это нужно:
Область видимости полей структуры ничем не отличается от обычных переменных, поля, начинающиеся с маленькой буквы, будут не видны во внешних пакетах. Эта особенность дает нам возможность запрета работы с полями напрямую (привет, ООП) и позволяет делать над ними обертки. Подробнее об этом поговорим в разделе про методы структур.

Полезные ссылки:

- Constructors and composite literals

Задание:

1. Форкните репозиторий module04 с кодом данного задания - в группу с вашими репозиториями - golang_users_repos/<your_gitlab_id>.
2. Создайте у себя в проекте module04 из ветки master ветку module04_01.
3. Реализуйте функцию CalcPrice, которая:
    - принимает на вход структуру Customer и цену какой-то абстрактной покупки (int);
    - возвращает в качестве первого аргумента итоговую цену с учетом скидки, а в качестве второго аргумента - ошибку, в случае если ее вернет реализация CalcDiscount;
    - в случае возврата из CalcDiscount ошибки первый аргумент установить в 0.

```
func CalcPrice(Customer, int) (int, error)
```

4. Функция должна быть реализована в пакете internal.
5. В ответе пришлите ссылку на MP в ветку master своего проекта ветки module04_01.