GO-04 04: Структуры и интерфейсы. Пустой интерфейс

Описание:

Теперь рассмотрим такое понятие, как пустой интерфейс.

```
var emI interface{}
```

Пустой интерфейс - это интерфейс, не требущий реализации, то есть, по сути, в переменную такого типа можно присвоить любое значение: структуру, примитив, сложный тип.

Для чего это нужно:

В случае обычного интерфейса на этапе компиляции определяется, можем ли мы присвоить ему объект, например, в момент вызова функции.

```
func f(w io.Writer) error {
    _, err := w.Write([]byte("Hello, world!")
    return err
}
```

Вызов такой функции с неподходящим аргументом (объект не реализует io.Writer) приведет к compile-time error

Но если мы напишем так:

```
func f(w interface{}) error {
    writer, ok := w.(io.Writer)
    if !ok {
        return errors.New("Incorrect type")
    }
    _, err := writer.Write([]byte("Hello, world!"))
    return err
}
```

Пояснение:

w.(io.Writer) - такая конструкция позволяет привести любой интерфейс к любому другому типу или интерфейсу и проверить корректность такого приведения.

То есть, программа успешно запустится и в случае несоответствия типов мы сами можем в рантайме обработать такую ситуацию (например, вернуть ошибку). Такая функция примет все, что угодно, а дальнейшие проверки ложатся на плечи разработчика.

Где это нужно:

- Допустим, наша функция умеет работать с любыми объектами - ей не важно, слайс у нас на входе или какая-то структура. В таком случае мы просто описываем логики для разичных типов внутри. Пример тому - fmt.Printf.
- Мы по каким-то причинам не хотим, чтобы несоответствие типов вызывало ошибку на этапе компиляции, а также хотим обработать исключительную ситуацию в рантайме (обычно так происходит, когда мы не знаем точно, что может прийти в функцию).

Но с пустым интерфейсом стоит быть осторожными и применять тогда, когды вы понимаете, зачем это делается, так как в противном случае он вносит определенную долю неоднозначности в код.

Обратите внимание:

В Go также присутствует пустая структура:

```
s := struct{}{}
```

Ее особенность в том, что объект такого типа занимает 0 байт:

```
var s struct{}
fmt.Println(unsafe.Sizeof(s))

// go run main.go
// Outrput:
// 0
```

Ее удобно использовать, когда вы, например, хотите найти элемент по ключу и создаете для этого мапу (что даст сложность алгоритма O(1)). А значения элементов этой мапы вам инициализировать нечем, тогда вы можете присваивать им пустую структуру для экономии памяти.

Такое же утверждение верно в случае каналов - если вы хотите просто сообщить о наступлении события, но само сообщение вы хотите оставить пустым.

Полезные ссылки:

- Type switch
- Interface conversions and type assertions

Задание:

1. Создайте в своем проекте module04 из ветки module04_03 - ветку module04_04.
2. В пакете main, используя логику из функции func startTransaction(debtor internal.Debtor) error (функция из описания предыдущего задания), разместите функцию startTransactionDynamic, которая не выдаст ошибку на этапе компиляции,

если ей передать неверный аргумент, а вернет ее в рантайме из функции вызывающей стороне (errors.New("Incorrect type")). Логику не меняйте.

3. В ответе пришлите ссылку на MP ветки module04_04 с нужными правками в ветку master своего проекта.