type & struct

Objectifs de l’atelier

Création d’alias de type

Création de structure de type

Travailler avec des constantes

les types natifs : les grandes familles

```
Numeric
String
Boolean
Pointer
Struct
Interface
```

```
Function
Method sets
Array
Slice
Map
Channel
```

les types natifs : types numériques

```
uint8       the set of all unsigned  8-bit integers (0 to 255)
uint16      the set of all unsigned 16-bit integers (0 to 65 535)
uint32      the set of all unsigned 32-bit integers (0 to 4 294 967 295)
uint64      the set of all unsigned 64-bit integers (0 to 18 446 744 073 709 551 615)

int8        the set of all signed  8-bit integers (-128 to 127)
int16       the set of all signed 16-bit integers (-32 768 to 32 767)
int32       the set of all signed 32-bit integers (-2 147 483 648 to 2 147 483 647)
int64       the set of all signed 64-bit integers (-9 223 372 036 854 775 808 to 9 223 372 036
854 775 807)

float32     the set of all IEEE-754 32-bit floating-point numbers
float64     the set of all IEEE-754 64-bit floating-point numbers

complex64   the set of all complex numbers with float32 real and imaginary parts
complex128  the set of all complex numbers with float64 real and imaginary parts

byte        alias for uint8
rune        alias for int32
```

new type

création d'un type

```
type myInt int32
...
var mi myInt
mi = 55
fmt.Printf("%T",mi) // main.myInt
```

attacher une méthode à un type

```
func (mi myInt) Multiply(n int) myInt {
   return mi * myInt(n)
}
```

TP

Définissez un nouveau type **myInt** de type **int32**

Créer les méthodes suivantes avec la signature de la fonction suivante :
un paramètre de type int, le retour est de type **myInt**. Attention le paramètre de type int doit être casté dans le type **myInt** afin de réaliser l'opération.

Add : pour l'opération d'addition
Sub : pour l'opération de soustraction
Multiply : pour la multiplication
Divide : pour la division

alias of type

création d’un alias de type

```
type newInt = myInt
...
var me newInt
me = 55
fmt.Printf(“%T”,mi) // main.myInt
```

utiliser une méthode du type sous-jacent

```
me.Multiply(3)
```

Exercice pratique

Depuis le TP précédent utilisez le type **myInt** afin de créer un alias de type **newInt**.

Dans la fonction main créer une variable **N** de type **newInt** avec une valeur initialisé à 2 utilisez les méthodes créé sur **myInt** depuis cette variable.

Affichez avec le package fmt la valeur du type **N**.

type of struct

Créer une structure

```
1  type User struct {
2      id int
3      FirstName, LastName string
4      Human
5  }
```

Utiliser une structure

```
1  var u User
2  u.id = 9265284
3  u.FirstName = “Bob”
4  u.LastName = ”Wilson”
5  u.Human.BirthDate = time.Now()
```

Utiliser une structure

```
1  var u User
2  u.id = 9265284
3  u.FirstName = “Bob”
4  u.LastName = ”Wilson”
5  u.BirthDate = time.Now()
```

Attacher une méthode à une structure

```
1  func (u *User) SayHi() string {
2       return “Hello ” + u.FirstName
3  }
4  …
5  u.SayHi()
```

const

Déclarer une constante

```
1
2
3
4
5
6
7
8
9
10
11
```

```go
const PI = 3,14
...
fmt.Println(PI) // output : 3,14
```

Déclarer une constante

```
1   const (
2       Ag = "Argent"
3       La = "Lanthane"
4       Ce = "Cérium"
5       Gd = "Gadolinium"
6       Pt = "Platine"
7       Au = "Or"
8   )
9
10
11
```

Déclarer une constante

```
1  type Periodic uint8
2  const (
3      Ag Periodic = iota
4      La
5      Ce
6      Gd
7      Pt
8      Au
9  )
10 …
11 fmt.Printf("%T : %v", Ag, Ag) // main.Periodic : 0
```

autre....

Précision d’une constante
https://play.golang.org/p/TqoMlF5VcO