MODUL X
STRUCT DAN METHOD

Pertemuan : 10
Waktu : 8 x 60 Menit (Online)

1.1 Tujuan Modul IX

Setelah mahasiswa mempelajari materi ini, diharapkan dapat :

1. Memahami fungsi Struct
2. Memahami fungsi Method

1.2 Landasan Teori

1.2.1 Struct

Go tidak memiliki class yang ada di bahasa-bahasa strict OOP lain. Tapi Go memiliki tipe data struktur yang disebut dengan Struct.

Struct adalah kumpulan definisi variabel (atau property) dan atau fungsi (atau method), yang dibungkus sebagai tipe data baru dengan nama tertentu. Property dalam struct, tipe datanya bisa bervariasi. Mirip seperti map, hanya saja key-nya sudah didefinisikan di awal, dan tipe data tiap itemnya bisa berbeda.

Dari sebuah struct, kita bisa buat variabel baru, yang memiliki atribut sesuai skema struct tersebut. Kita sepakati dalam buku ini, variabel tersebut dipanggil dengan istilah **object** atau **object struct**.

Konsep struct di golang mirip dengan konsep **class** pada OOP, meski sebenarnya berbeda. Disini penulis menggunakan konsep OOP sebagai analogi, dengan tujuan untuk mempermudah dalam mencerna isi bab ini.

Dengan memanfaatkan struct, grouping data akan lebih mudah, selain itu dan rapi dan gampang untuk di-maintain.

1.2.1.1 Deklarasi Struct

Keyword `type` digunakan untuk deklarasi struct. Di bawah ini merupakan contoh cara penggunaannya.

```
type student struct {

    name string

    grade int

}
```

Struct `student` dideklarasikan memiliki 2 property, yaitu `name` dan `grade`. Objek yang dibuat dengan struct ini nantinya memiliki skema atau struktur yang sama.

1.2.1.2 Penerapan Struct

Struct `student` yang sudah disiapkan di atas akan kita manfaatkan untuk membuat variabel objek. Property variabel tersebut di-isi kemudian ditampilkan.

```
func main() {
    var s1 student
    s1.name = "john wick"
    s1.grade = 2

    fmt.Println("name  :", s1.name)
    fmt.Println("grade :", s1.grade)
}
```

Cara membuat variabel objek sama seperti pembuatan variabel biasa. Tinggal tulis saja nama variabel diikuti nama struct, contoh: `var s1 student`.

Semua property variabel objek pada awalnya memiliki zero value sesuai tipe datanya.

Property variabel objek bisa diakses nilainya menggunakan notasi titik, contohnya `s1.name`. Nilai property-nya juga bisa diubah, contohnya `s1.grade = 2`.

1.2.1.3 Inisialisasi Object Struct

Cara inisialisasi variabel objek adalah dengan menambahkan kurung kurawal setelah nama struct. Nilai masing-masing property bisa diisi pada saat inisialisasi.

Pada contoh berikut, terdapat 3 buah variabel objek yang dideklarasikan dengan cara berbeda.

```
var s1 = student{}

s1.name = "wick"

s1.grade = 2


var s2 = student{"ethan", 2}


var s3 = student{name: "jason"}


fmt.Println("student 1 :", s1.name)

fmt.Println("student 2 :", s2.name)

fmt.Println("student 3 :", s3.name)
```

Pada kode di atas, variabel `s1` menampung objek cetakan `student`. Variabel tersebut kemudian di-set nilai property-nya.

Variabel objek `s2` dideklarasikan dengan metode yang sama dengan `s1`, pembedanya di `s2` nilai propertinya di isi langsung ketika deklarasi. Nilai pertama akan menjadi nilai property pertama (yaitu `name`), dan selanjutnya berurutan.

Pada deklarasi `s3`, dilakukan juga pengisian property ketika pencetakan objek. Hanya saja, yang diisi hanya `name` saja. Cara ini cukup efektif jika digunakan untuk membuat objek baru yang nilai property-nya tidak semua harus disiapkan di awal. Keistimewaan lain menggunakan cara ini adalah penentuan nilai property bisa dilakukan dengan tidak berurutan. Contohnya:

```
var s4 = student{name: "wayne", grade: 2}

var s5 = student{grade: 2, name: "bruce"}
```

1.2.1.4 Variabel Objek Pointer

Objek yang dibuat dari tipe struct bisa diambil nilai pointer-nya, dan bisa disimpan pada variabel objek yang bertipe struct pointer. Contoh penerapannya:

```
var s1 = student{name: "wick", grade: 2}

var s2 *student = &s1
fmt.Println("student 1, name :", s1.name)
fmt.Println("student 4, name :", s2.name)

s2.name = "ethan"
fmt.Println("student 1, name :", s1.name)
fmt.Println("student 4, name :", s2.name)
```

`s2` adalah variabel pointer hasil cetakan struct `student`. `s2` menampung nilai referensi `s1`, menjadikan setiap perubahan pada property variabel tersebut, akan juga berpengaruh pada variabel objek `s1`.

Meskipun `s2` bukan variabel asli, property nya tetap bisa diakses seperti biasa. Inilah keistimewaan property dalam objek pointer, tanpa perlu di-dereferensi nilai asli property tetap bisa diakses. Pengisian nilai pada property tersebut juga bisa langsung menggunakan nilai asli, contohnya seperti `s2.name = "ethan"`.

1.2.1.5 Embedded Struct

Embedded struct adalah mekanisme untuk menempelkan sebuah struct sebagai properti struct lain. Agar lebih mudah dipahami, mari kita bahas kode berikut.

```
package main

import "fmt"

type person struct {
    name string
    age  int
}

type student struct {
    grade int
    person
}

func main() {
    var s1 = student{}
    s1.name = "wick"
    s1.age = 21
    s1.grade = 2

    fmt.Println("name  :", s1.name)
```

```
    fmt.Println("age    :", s1.age)
    fmt.Println("age    :", s1.person.age)
    fmt.Println("grade  :", s1.grade)
}
```

Pada kode di atas, disiapkan struct `person` dengan properti yang tersedia adalah `name` dan `age`. Disiapkan juga struct `student` dengan property `grade`. Struct `person` di-embed kedalam struct `student`. Caranya cukup mudah, yaitu dengan menuliskan nama struct yang ingin di-embed ke dalam body `struct` target.

Embedded struct adalah **mutable**, nilai property-nya nya bisa diubah.

Khusus untuk properti yang bukan properti asli (properti turunan dari struct lain), bisa diakses dengan cara mengakses struct *parent*-nya terlebih dahulu, contohnya `s1.person.age`. Nilai yang dikembalikan memiliki referensi yang sama dengan `s1.age`.

1.2.1.6 Embedded Struct Dengan Nama Property Yang Sama

Jika salah satu nama properti sebuah struct memiliki kesamaan dengan properti milik struct lain yang di-embed, maka pengaksesan property-nya harus dilakukan secara eksplisit atau jelas. Contoh bisa dilihat di kode berikut.

```
package main

import "fmt"

type person struct {
    name string
    age  int
}

type student struct {
    person
    age   int
    grade int
}

func main() {
    var s1 = student{}
    s1.name = "wick"
    s1.age = 21         // age of student
    s1.person.age = 22 // age of person

    fmt.Println(s1.name)
```

```
    fmt.Println(s1.age)
    fmt.Println(s1.person.age)
}
```

Struct `person` di-embed ke dalam struct `student`, dan kedua struct tersebut kebetulan salah satu nama property-nya ada yg sama, yaitu `age`. Cara mengakses property `age` milik struct `person` lewat objek struct `student`, adalah dengan menuliskan nama struct yg di-embed kemudian nama property-nya, contohnya: `s1.person.age = 22`.

1.2.1.7 Pengisian Nilai Sub-Struct

Pengisian nilai property sub-struct bisa dilakukan dengan langsung memasukkan variabel objek yang tercetak dari struct yang sama.

```
var p1 = person{name: "wick", age: 21}
var s1 = student{person: p1, grade: 2}

fmt.Println("name  :", s1.name)
fmt.Println("age   :", s1.age)
fmt.Println("grade :", s1.grade)
```

Pada deklarasi `s1`, property `person` diisi variabel objek `p1`.

1.2.1.8 Anonymous Struct

Anonymous struct adalah struct yang tidak dideklarasikan di awal sebagai tipe data baru, melainkan langsung ketika pembuatan objek. Teknik ini cukup efisien untuk pembuatan variabel objek yang struct-nya hanya dipakai sekali.

```
package main

import "fmt"

type person struct {
    name string
    age  int
}

func main() {
    var s1 = struct {
        person
        grade int
    }{}
    s1.person = person{"wick", 21}
```

```
    s1.grade = 2

    fmt.Println("name  :", s1.person.name)
    fmt.Println("age   :", s1.person.age)
    fmt.Println("grade :", s1.grade)
}
```

Pada kode di atas, variabel `s1` langsung diisi objek anonymous struct yang memiliki property `grade`, dan property `person` yang merupakan embedded struct.

Salah satu aturan yang perlu diingat dalam pembuatan anonymous struct adalah, deklarasi harus diikuti dengan inisialisasi. Bisa dilihat pada `s1` setelah deklarasi struktur struct, terdapat kurung kurawal untuk inisialisasi objek. Meskipun nilai tidak diisikan di awal, kurung kurawal tetap harus ditulis.

```
// anonymous struct tanpa pengisian property
var s1 = struct {
    person
    grade int
}{}

// anonymous struct dengan pengisian property
var s2 = struct {
    person
    grade int
}{
    person: person{"wick", 21},
    grade:  2,
}
```

1.2.1.9 Kombinasi Slice & Struct

Slice dan `struct` bisa dikombinasikan seperti pada slice dan `map`, caranya penggunaannya-pun mirip, cukup tambahkan tanda `[]` sebelum tipe data pada saat deklarasi.

```
type person struct {
    name string
    age  int
}

var allStudents = []person{
    {name: "Wick", age: 23},
    {name: "Ethan", age: 23},
    {name: "Bourne", age: 22},
```

```
}

for _, student := range allStudents {
    fmt.Println(student.name, "age is", student.age)
}
```

1.2.1.10 Inisialisasi Slice Anonymous Struct

Anonymous struct bisa dijadikan sebagai tipe sebuah slice. Dan nilai awalnya juga bisa diinisialisasi langsung pada saat deklarasi. Berikut adalah contohnya:

```
var allStudents = []struct {
    person
    grade int
}{
    {person: person{"wick", 21}, grade: 2},
    {person: person{"ethan", 22}, grade: 3},
    {person: person{"bond", 21}, grade: 3},
}

for _, student := range allStudents {
    fmt.Println(student)
}
```

1.2.1.11 Deklarasi Anonymous Struct Menggunakan Keyword var

Cara lain untuk deklarasi anonymous struct adalah dengan menggunakan keyword `var`.

```
var student struct {
    person
    grade int
}

student.person = person{"wick", 21}
student.grade = 2
```

Statement `type student struct` adalah contoh cara deklarasi struct. Maknanya akan berbeda ketika keyword `type` diganti `var`, seperti pada contoh di atas `var student struct`, yang artinya dicetak sebuah objek dari anonymous struct kemudian disimpan pada variabel bernama `student`.

Deklarasi anonymous struct menggunakan metode ini juga bisa dilakukan beserta inisialisasi-nya.

```
// hanya deklarasi
var student struct {
    grade int
}

// deklarasi sekaligus inisialisasi
var student = struct {
    grade int
} {
    12,
}
```

1.2.1.12 Nested struct

Nested struct adalah anonymous struct yang di-embed ke sebuah struct. Deklarasinya langsung didalam struct peng-embed. Contoh:

```
type student struct {
    person struct {
        name string
        age  int
    }
    grade   int
    hobbies []string
}
```

Teknik ini biasa digunakan ketika decoding data **json** yang struktur datanya cukup kompleks dengan proses decode hanya sekali.

1.2.1.13 Deklarasi Dan Inisialisasi Struct Secara Horizontal

Deklarasi struct bisa dituliskan secara horizontal, caranya bisa dilihat pada kode berikut:

```
type person struct { name string; age int; hobbies []string }
```

Tanda semi-colon (`;`) digunakan sebagai pembatas deklarasi poperty yang dituliskan secara horizontal. Inisialisasi nilai juga bisa dituliskan dengan metode ini. Contohnya:

```
var p1 = struct { name string; age int } { age: 22, name: "wick" }
var p2 = struct { name string; age int } { "ethan", 23 }
```

Bagi pengguna editor Sublime yang terinstal plugin GoSublime didalamnya, cara ini tidak akan bisa dilakukan, karena setiap kali file di-save, kode program dirapikan. Jadi untuk mengetesnya bisa dengan menggunakan editor lain.

1.2.1.14 Tag property dalam struct

Tag merupakan informasi opsional yang bisa ditambahkan pada masing-masing property struct.

```
type person struct {
    name string `tag1`
    age  int    `tag2`
}
```

Tag biasa dimanfaatkan untuk keperluan encode/decode data json. Informasi tag juga bisa diakses lewat reflect. Nantinya akan ada pembahasan yang lebih detail mengenai pemanfaatan tag dalam struct, terutama ketika sudah masuk bab JSON.

1.2.1.15 Type Alias

Sebuah tipe data, seperti struct, bisa dibuatkan alias baru, caranya dengan `type NamaAlias = TargetStruct`. Contoh:

```
type Person struct {
    name string
    age  int
}
type People = Person

var p1 = Person{"wick", 21}
fmt.Println(p1)

var p2 = People{"wick", 21}
fmt.Println(p2)
```

Pada kode di atas, sebuah alias bernama `People` dibuat untuk struct `Person`.

Casting dari objek (yang dicetak lewat struct tertentu) ke tipe yang merupakan alias dari struct pencetak, hasilnya selalu valid. Berlaku juga sebaliknya.

```
people := People{"wick", 21}
fmt.Println(Person(people))

person := Person{"wick", 21}
fmt.Println(People(person))
```

Pembuatan struct baru juga bisa dilakukan lewat teknik type alias. Silakan perhatikan kode berikut.

```
type People1 struct {
    name string
    age  int
}
type People2 = struct {
    name string
    age  int
}
```

Struct `People1` dideklarasikan. Struct alias `People2` juga dideklarasikan, struct ini merupakan alias dari anonymous struct. Penggunaan teknik type alias untuk anonymous struct menghasilkan output yang ekuivalen dengan pendeklarasian struct.

Perlu diketahui juga, dan di atas sudah sempat disinggung, bahwa teknik type alias ini tidak didesain hanya untuk pembuatan alias pada tipe struct saja, semua jenis tipe data bisa dibuatkan alias. Di contoh berikut, dipersiapkan tipe `Number` yang merupakan alias dari tipe data `int`.

```
type Number = int
var num Number = 12
```

1.2.2 Method

Method adalah fungsi yang menempel pada `type` (bisa `struct` atau tipe data lainnya). Method bisa diakses lewat variabel objek.

Keunggulan method dibanding fungsi biasa adalah memiliki akses ke property struct hingga level *private* (level akses nantinya akan dibahas lebih detail pada bab selanjutnya). Dan juga, dengan menggunakan method sebuah proses bisa di-enkapsulasi dengan baik.

1.2.2.1 Penerapan Method

Cara menerapkan method sedikit berbeda dibanding penggunaan fungsi. Ketika deklarasi, ditentukan juga siapa pemilik method tersebut. Contohnya bisa dilihat pada kode berikut:

```
package main

import "fmt"
import "strings"

type student struct {
    name  string
    grade int
}

func (s student) sayHello() {
    fmt.Println("halo", s.name)
}

func (s student) getNameAt(i int) string {
    return strings.Split(s.name, " ")[i-1]
}
```

Cara deklarasi method sama seperti fungsi, hanya saja perlu ditambahkan deklarasi variabel objek di sela-sela keyword `func` dan nama fungsi. Struct yang digunakan akan menjadi pemilik method.

`func (s student) sayHello()` maksudnya adalah fungsi `sayHello` dideklarasikan sebagai method milik struct `student`. Pada contoh di atas struct `student` memiliki dua buah method, yaitu `sayHello()` dan `getNameAt()`.

Contoh pemanfaatan method bisa dilihat pada kode berikut.

```
func main() {
    var s1 = student{"john wick", 21}
    s1.sayHello()

    var name = s1.getNameAt(2)
    fmt.Println("nama panggilan :", name)
}
```

Cara mengakses method sama seperti pengaksesan properti berupa variabel. Tinggal panggil saja methodnya.

```
s1.sayHello()

var name = s1.getNameAt(2)
```

Method memiliki sifat yang sama persis dengan fungsi biasa. Seperti bisa berparameter, memiliki nilai balik, dan lainnya. Dari segi sintaks, pembedanya hanya ketika pengaksesan dan deklarasi. Bisa dilihat di kode berikut, sekilas perbandingan penulisan fungsi dan method.

```
func sayHello() {

func (s student) sayHello() {



func getNameAt(i int) string {

func (s student) getNameAt(i int) string {
```

1.2.2.2 Method Pointer

Method pointer adalah method yang variabel objek pemilik method tersebut berupa pointer.

Kelebihan method jenis ini adalah, ketika kita melakukan manipulasi nilai pada property lain yang masih satu struct, nilai pada property tersebut akan di rubah pada reference nya. Lebih jelasnya perhatikan kode berikut.

```go
package main

import "fmt"

type student struct {
    name  string
    grade int
}

func (s student) changeName1(name string) {
    fmt.Println("---> on changeName1, name changed to", name)
    s.name = name
}

func (s *student) changeName2(name string) {
    fmt.Println("---> on changeName2, name changed to", name)
    s.name = name
}
func main() {
    var s1 = student{"john wick", 21}
    fmt.Println("s1 before", s1.name)
    // john wick
    s1.changeName1("jason bourne")
    fmt.Println("s1 after changeName1", s1.name)
    // john wick
    s1.changeName2("ethan hunt")
    fmt.Println("s1 after changeName2", s1.name)
    // ethan hunt
}
```

Setelah eksekusi statement `s1.changeName1("jason bourne")`, nilai `s1.name` tidak berubah. Sebenarnya nilainya berubah tapi hanya dalam method `changeName1()` saja, nilai pada reference di objek-nya tidak berubah. Karena itulah ketika objek di print value dari `s1.name` tidak berubah.

Keistimewaan lain method pointer adalah, method itu sendiri bisa dipanggil dari objek pointer maupun objek biasa.

```
// pengaksesan method dari variabel objek biasa

var s1 = student{"john wick", 21}

s1.sayHello()



// pengaksesan method dari variabel objek pointer

var s2 = &student{"ethan hunt", 22}

s2.sayHello()
```

Berikut adalah penjelasan tambahan mengenai beberapa hal pada sub bab ini.

• Penggunaan Fungsi `strings.Split()`

Di bab ini ada fungsi baru yang kita gunakan: `strings.Split()`. Fungsi ini berguna untuk memisah string menggunakan pemisah yang ditentukan sendiri. Hasilnya adalah array berisikan kumpulan substring.

```
strings.Split("ethan hunt", " ")

// ["ethan", "hunt"]
```

Pada contoh di atas, string `"ethan hunt"` dipisah menggunakan separator spasi `" "`. Maka hasilnya terbentuk array berisikan 2 data, `"ethan"` dan `"hunt"`.

1.2.2.3 Apakah fmt.Println() & strings.Split() Juga Merupakan Method?

Setelah tahu apa itu method dan bagaimana penggunaannya, mungkin akan muncul di benak kita bahwa kode seperti `fmt.Println()`, `strings.Split()` dan lainnya-yang-berada-pada-package-lain adalah merupakan method. Tapi sayangnya **bukan**. `fmt` disitu bukanlah variabel objek, dan `Println()` bukan merupakan method-nya.

`fmt` adalah nama **package** yang di-import (bisa dilihat pada kode `import "fmt"`). Sedangkan `Println()` adalah **nama fungsi**. Untuk mengakses fungsi yang berada pada package lain, harus dituliskan nama package-nya. Hal ini berlaku juga di dalam package `main`. Jika ada fungsi dalam package main yang diakses dari package lain yang berbeda, maka penulisannya `main.NamaFungsi()`.

1.3 Praktikum

1.3.1 Latihan Praktikum

```
package main

import "fmt"

type person struct {
    name []string
    age  int
}

func newPerson(name string) *person {

    p := person{name: name}
    p.age = 42
    return &p
}

func main() {

    fmt.Printl(person{"Bob", 20})

    fmt.Println(person{name: "Alice", age: 30})

    fmt.Println(person{name: "Fred"})

    fmt.Println(person{name: "Ann", age: 40})

    fmt.Println(newPerson("Jon"))

    s := person{name: "Sean", age: 50}
    fmt.Println(s.name)

    sp := &s
    fmt.Println(sp.age)

    sp.age = 51
    fmt.Println(sp.age)
}
```

Untuk latihan praktikum hari ini, silahkan untuk membenarkan program diatas sehingga berjalan seperti screenshot dibawah ini.

```
{Bob 20}
{Alice 30}
{Fred 0}
&{Ann 40}
&{Jon 42}
Sean
50
51
```

1.3.2 Tugas Praktikum

Coming Soon