

# Hari 8 - Interface

## Interface

Interface adalah kumpulan definisi method yang tidak memiliki isi (hanya definisi saja), yang dibungkus dengan nama tertentu.

Interface merupakan tipe data. Nilai object bertipe interface zero value-nya adalah `nil`. Interface mulai bisa digunakan jika sudah ada isinya, yaitu object konkret yang memiliki definisi method minimal sama dengan yang ada di interface-nya.

### 1. Penerapan Interface

Yang pertama perlu dilakukan untuk menerapkan interface adalah menyiapkan interface beserta definisi method nya. Keyword `type` dan `interface` digunakan untuk pendefinisian interface.

```
package main

import "fmt"
import "math"

type hitung interface {
    luas() float64
    keliling() float64
}
```

Pada kode di atas, interface `hitung` memiliki 2 definisi method, `luas()` dan `keliling()`. Interface ini nantinya digunakan sebagai tipe data pada variabel, dimana variabel tersebut akan menampung object bangun datar hasil dari struct yang akan kita buat.

Dengan memanfaatkan interface `hitung`, perhitungan luas dan keliling bangun datar bisa dilakukan, tanpa perlu tahu jenis bangun datarnya sendiri itu apa.

Siapkan struct bangun datar `lingkaran`, struct ini memiliki method yang beberapa diantaranya terdefinisi di interface `hitung`.

```
type lingkaran struct {
    diameter float64
}

func (l lingkaran) jariJari() float64 {
    return l.diameter / 2
}

func (l lingkaran) luas() float64 {
    return math.Pi * math.Pow(l.jariJari(), 2)
}

func (l lingkaran) keliling() float64 {
    return math.Pi * l.diameter
}
```

Struct `lingkaran` di atas memiliki tiga method, `jariJari()`, `luas()`, dan `keliling()`.

Selanjutnya, siapkan struct bangun datar `persegi`.

```
type persegi struct {
    sisi float64
}

func (p persegi) luas() float64 {
    return math.Pow(p.sisi, 2)
}

func (p persegi) keliling() float64 {
    return p.sisi * 4
}
```

Buat implementasi perhitungan di `main`.

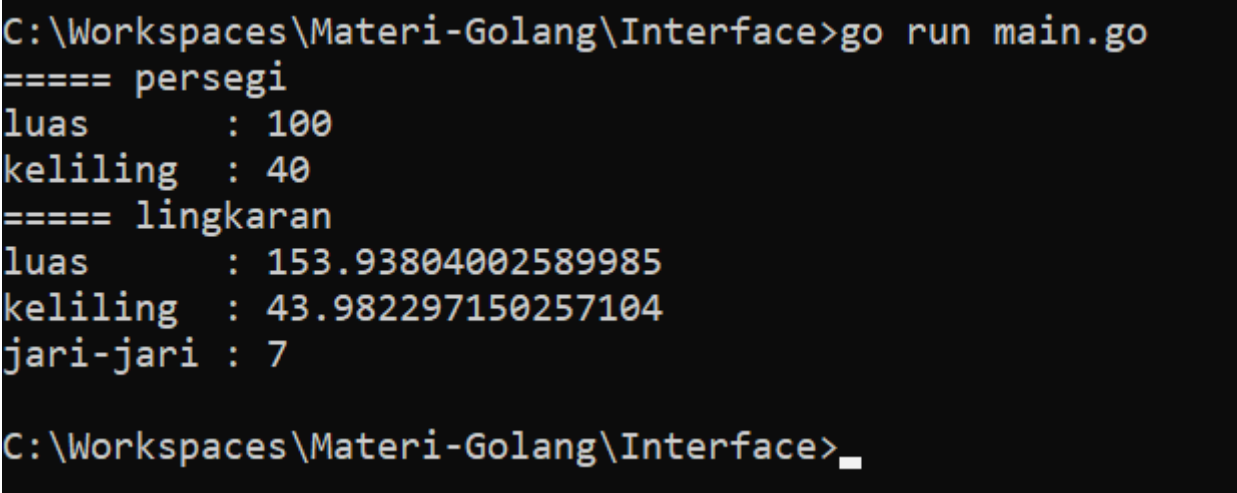
```
func main() {
    var bangunDatar hitung

    bangunDatar = persegi{10.0}
    fmt.Println("==== persegi")
    fmt.Println("luas      :", bangunDatar.luas())
    fmt.Println("keliling  :",
    bangunDatar.keliling())

    bangunDatar = lingkaran{14.0}
    fmt.Println("==== lingkaran")
    fmt.Println("luas      :", bangunDatar.luas())
    fmt.Println("keliling  :",
    bangunDatar.keliling())
    fmt.Println("jari-jari :", bangunDatar.
    (lingkaran).jariJari())
}
```

Perhatikan kode di atas. Variabel object `bangunDatar` bertipe interface `hitung`. Variabel tersebut digunakan untuk menampung object konkrit buatan struct `lingkaran` dan `persegi`.

Dari variabel tersebut, method `luas()` dan `keliling()` diakses. Secara otomatis Golang akan mengarahkan pemanggilan method pada interface ke method asli milik struct yang bersangkutan.



Method `jariJari()` pada struct `lingkaran` tidak akan bisa diakses karena tidak terdefinisi dalam interface `hitung`. Pengaksesannya dengan paksa akan menyebabkan error.

Untuk mengakses method yang tidak ter-definisi di interface, variabel-nya harus di-casting terlebih dahulu ke tipe asli variabel konkritnya (pada kasus ini tipenya `lingkaran`), setelahnya method akan bisa diakses.

Cara casting object interface sedikit unik, yaitu dengan menuliskan nama tipe tujuan dalam kurung, ditempatkan setelah nama interface dengan menggunakan notasi titik (seperti cara mengakses property, hanya saja ada tanda kurung nya). Contohnya bisa dilihat di kode berikut. Statement `bangunDatar.(lingkaran)` adalah contoh casting pada object interface.

```
var bangunDatar hitung = lingkaran{14.0}
var bangunLingkaran lingkaran = bangunDatar.
(lingkaran)

bangunLingkaran.jariJari()
```

Perlu diketahui juga, jika ada interface yang menampung object konkrit dimana struct-nya tidak memiliki salah satu method yang terdefinisi di interface, error juga akan muncul. Intinya kembali ke aturan awal, variabel interface hanya bisa menampung object yang minimal memiliki semua method yang terdefinisi di interface-nya.

## 2. Embedded Interface

Interface bisa di-embed ke interface lain, sama seperti struct. Cara penerapannya juga sama, cukup dengan menuliskan nama interface yang ingin di-embed ke dalam interface tujuan.

Pada contoh berikut, disiapkan interface bernama `hitung2d` dan `hitung3d`. Kedua interface tersebut kemudian di-embed ke interface baru bernama `hitung`.

```
import "math"

type hitung2d interface {
    luas() float64
    keliling() float64
}

type hitung3d interface {
    volume() float64
}

type hitung interface {
    hitung2d
    hitung3d
}
```

Interface **hitung2d** berisikan method untuk kalkulasi luas dan keliling, sedang **hitung3d** berisikan method untuk mencari volume bidang. Kedua interface tersebut diturunkan di interface **hitung**, menjadikannya memiliki kemampuan untuk menghitung luas, keliling, dan volume.

Next, siapkan struct baru bernama **kubus** yang memiliki method **luas()**, **keliling()**, dan **volume()**.

```
type kubus struct {
    sisi float64
}

func (k kubus) volume() float64 {
    return math.Pow(k.sisi, 3)
}

func (k kubus) luas() float64 {
    return math.Pow(k.sisi, 2) * 6
}

func (k kubus) keliling() float64 {
    return k.sisi * 12
}
```

object hasil cetakan struct **kubus** di atas, nantinya akan ditampung oleh object cetakan interface **hitung** yang isinya merupakan gabungan interface **hitung2d** dan **hitung3d**.

Terakhir, buat implementasi-nya di main.

```
func main() {
    var bangunRuang hitung = kubus{4}

    fmt.Println("==== kubus")
    fmt.Println("luas      :", bangunRuang.luas())
    fmt.Println("keliling  :",
bangunRuang.keliling())
    fmt.Println("volume    :",
bangunRuang.volume())
}
```

Bisa dilihat di kode di atas, lewat interface **hitung**, method **luas**, **keliling**, dan **volume** bisa di akses.

```
C:\Workspaces\Materi-Golang\Interface>go run main.go
==== kubus
luas      : 96
keliling  : 48
volume    : 64

C:\Workspaces\Materi-Golang\Interface>_
```

---

## Interface Kosong

---

## 1. Penggunaan `interface{}`

`interface{}` merupakan tipe data, sehingga cara penggunaannya sama seperti pada tipe data lainnya, hanya saja nilai yang diisikan bisa apa saja. Contoh:

```
package main

import "fmt"

func main() {
    var secret interface{}

    secret = "ethan hunt"
    fmt.Println(secret)

    secret = []string{"apple", "manggo", "banana"}
    fmt.Println(secret)

    secret = 12.4
    fmt.Println(secret)
}
```

Keyword `interface` seperti yang kita tau, digunakan untuk pembuatan interface. Tetapi ketika ditambahkan kurung kurawal (`{}`) di belakang-nya (menjadi `interface{}`), maka kegunaannya akan berubah, yaitu sebagai tipe data.

```
C:\Workspaces\Materi-Golang\Interface>go run main.go
ethan hunt
[apple manggo banana]
12.4

C:\Workspaces\Materi-Golang\Interface>
```

## 2. Casting Variabel Interface Kosong

Variabel bertipe `interface{}` bisa ditampilkan ke layar sebagai `string` dengan memanfaatkan fungsi print, seperti `fmt.Println()`. Tapi perlu diketahui bahwa nilai yang dimunculkan tersebut bukanlah nilai asli, melainkan bentuk string dari nilai aslinya.

Hal ini penting diketahui, karena untuk melakukan operasi yang membutuhkan nilai asli pada variabel yang bertipe `interface{}`, diperlukan casting ke tipe aslinya. Contoh seperti pada kode berikut.

```
package main

import "fmt"
import "strings"

func main() {
    var secret interface{}

    secret = 2
    var number = secret.(int) * 10
    fmt.Println(secret, "multiplied by 10 is :",
    number)

    secret = []string{"apple", "manggo", "banana"}
    var fruits = strings.Join(secret.([]string), ",
    ")
    fmt.Println(fruits, "is my favorite fruits")
}
```

Pertama, variabel `secret` menampung nilai bertipe numerik. Ada kebutuhan untuk mengalikan nilai yang ditampung variabel tersebut dengan angka `10`. Maka perlu dilakukan casting ke tipe aslinya, yaitu `int`, setelahnya barulah nilai bisa dioperasikan, yaitu `secret.(int) * 10`.

Pada contoh kedua, `secret` berisikan array string. Kita memerlukan string tersebut untuk digabungkan dengan pemisah tanda koma. Maka perlu di-casting ke `[]string` terlebih dahulu sebelum bisa digunakan di `strings.Join()`, contohnya pada `strings.Join(secret.([]string), ", ")`.

```
C:\Workspaces\Materi-Golang\Interface>go run main.go
2 multiplied by 10 is : 20
apple, manggo, banana is my favorite fruits

C:\Workspaces\Materi-Golang\Interface>
```

Variabel `interface{}` bisa menyimpan data apa saja, termasuk data object, pointer, ataupun gabungan keduanya. Di bawah ini merupakan contoh penerapan interface untuk menampung data object pointer.

```
package main

import (
    "fmt"
)

type person struct {
    name string
    age int
}

func main(){
    var secret interface{} = &person{name: "wick",
age: 27}
    var name = secret.(*person).name
    fmt.Println(name)
}
```

Variabel `secret` dideklarasikan bertipe `interface{}` menampung referensi object cetakan struct `person`. Cara casting dari `interface{}` ke struct pointer adalah dengan menuliskan nama struct-nya dan ditambahkan tanda asterisk (\*) di awal, contohnya seperti `secret.(*person)`. Setelah itu barulah nilai asli bisa diakses.

## Referensi Video:

- [Interface](#) (Programmer Zaman Now)
- [Interface Kosong](#) (Programmer Zaman Now)
- [Type Assertion](#) (Programmer Zaman Now)

## Referensi Tulisan:

- <https://dasarpemrogramangolang.novalagung.com/A-interface.html>
- <https://dasarpemrogramangolang.novalagung.com/A-interface-kosong.html>

## Rating - Feedback

Berikan Rating pada posting ini:



Berikan kritik dan saran..

Submit