Interface

Interface

Interface adalah sebuah tipe data pada bahasa Go yang merupakan kumpulan dari definisi satu atau lebih *method*. Kita dapat menggunakan tipe data *interface* jika kita telah mengimplementasikan *method-method* yang telah didefinisikan oleh interface tersebut melalui tipe data lainnya.

Cara membuat sebuah interface cukup mudah, contohnya seperti pada gambar dibawah ini.

```
type shape interface {
  area() float64
  perimeter() float64
}
```

Gambar diatas merupakan sebuah *interface* bernama *shape* yang mempunyai 2 method bernama *area*, dan *perimeter*. Method *area* merupakan sebuah method yang me-return nilai dengan tipe data *float64* dan method *perimeter* merupakan sebuah method yang me-return nilai dengan tipe data float64. Kita dapat menggunakan interface *shape* jika kita telah mengimplementasikan method-method yang telah didefinisikan oleh interface *shape* melalui tipe data lainnya. Untuk sesi kali ini, kita akan mengimplementasikan method-method dari interface *shape* melalui tipe data *struct*.

Interface

Perhatikan pada gambar disebelah kanan. Terdapat sebuah interface bernama *shape* yang sudah kita bahas pada halaman sebelumnya. Lalu terdapat 2 buah *struct* bernama *circle* dan *rectangle.* Struct *circle* mempunyai satu field bernama *radius* yang memiliki tipe data float64. Kemudian struct *rectangle* memiliki 2 field bernama *width* dan *height* yang keduanya memiliki tipe data yang sama yaitu *float64*.

Karena kita ingin menggunakan tipe data interface *shape* dan ingin kita gunakan untuk struct *circle* dan *shape*, maka kedua *struct* tersebut perlu mengimplementasikan method-method yang didefinisikan oleh interface *shape*.

- Pada line 20 - 22, struct *circle* mengimplementasikan method *area* untuk menghitung luas lingkaran.
- Pada line 24-26, struct *rectangle* mengimplementasikan method *area* untuk menghitung luas persegi panjang.
- Pada line 28-30, struct *circle* mengimplementasikan method *perimeter* untuk menghitung keliling lingkaran.
- Pada line 32-34, struct *rectangle* mengimplementasikan method *area* untuk menghitung keliling persegi panjang.

```
1  package main
2
3  import (
4    "fmt"
5    "math"
6  )
7
8  type shape interface {
9    area() float64
10   perimeter() float64
11 }
12
13 type rectangle struct {
14   width, height float64
15 }
16 type circle struct {
17   radius float64
18 }
19
20 func (c circle) area() float64 {
21   return math.Pi * math.Pow(c.radius, 2)
22 }
23
24 func (r rectangle) area() float64 {
25   return r.height * r.width
26 }
27
28 func (c circle) perimeter() float64 {
29   return 2 * math.Pi * c.radius
30 }
31
32 func (r rectangle) perimeter() float64 {
33   return 2 * (r.height + r.width)
34 }
```

Interface

Perhatikan pada gambar di sebelah kanan. Terdapat 2 buah variable bernama *c1* dan *r1*.

Variable *c1* memiliki tipe data interface *shape* , dan diberikan nilai berupa struct *circle*. Hal ini dapat terjadi karena struct *circle* telah mengimplementasikan seluruh method yang didefinisikan oleh interface *shape*.

Kemudian terdapat sebuah variable bernama *r1* yang juga memiliki tipe data interface *shape* dan diberikan nilai berupa struct *rectangle*.

Variable *r1* dapat menampung struct *rectangle* karena struct *rectangle* juga telah mengimplementasikan seluruh method yang didefinisikan oleh interface *shape*.

```
42  func main() {
43      var c1 shape = circle{radius: 5}
44      var r1 shape = rectangle{width: 3, height: 2}
45
46      fmt.Printf("Type of c1: %T", c1)
47      fmt.Printf("Type of r1: %T", r1)
48
49  }
```

```
Type of c1: main.circle
Type of r1: main.rectangle
```

Interface

Namun jika kita mencoba untuk mencari tahu tipe data dari *c1* dan *r1* dengan verb %T, maka hasilnya seperti pada gambar kedua.
Jika kita perhatikan hasilnya, variable *c1* memiliki tipe data struct *circle* dan variable *r1* memiliki tipe data struct *rectangle*.

Hasil yang kita lihat pada gambar kedua merupakan tipe data konkrit atau tipe data asli dari variable *c1* dan *r1*. Kedua variable tersebut memiliki 2 tipe data yaitu *struct* yang telah dijadikan sebagai nilai dan interface *shape*.

Inilah yang disebut dengan **polymorphism** atau **dynamic type**. Dengan mengimplementasikan *interface,* maka suatu variable dapat mempunyai 2 tipe data.

```
42  func main() {
43      var c1 shape = circle{radius: 5}
44      var r1 shape = rectangle{width: 3, height: 2}
45
46      fmt.Printf("Type of c1: %T", c1)
47      fmt.Printf("Type of r1: %T", r1)
48
49  }
```

```
Type of c1: main.circle
Type of r1: main.rectangle
```

Interface

Sekarang mari kita panggil method-method yang telah kita buat pada halaman sebelumnya. Caranya seperti pada gambar pertama di sebelah kanan.

Jika kita jalankan maka hasilnya akan seperti pada gambar kedua.

```
func main() {
  var c1 shape = circle{radius: 5}
  var r1 shape = rectangle{width: 3, height: 2}

  fmt.Println("Circle area", c1.area())
  fmt.Println("Circle perimeter", c1.perimeter())

  fmt.Println("Rectangle area", r1.area())
  fmt.Println("Rectangle perimter", r1.perimeter())
}
```

```
Circle area 78.53981633974483
Circle perimeter 31.41592653589793
Rectangle area 6
Rectangle perimter 10
```

Interface

Mari kita re-factor codingan kita dari halaman sebelumnya. Contohnya seperti pada gambar pertama di sebelah kanan. Kita membuat sebuah function bernama *print* yang menerima 2 parameter dengan tipe data *string* dan *interface*. Kemudian di dalam function *print* kita memanggil method *area* dan *perimeter*.

Karena variable *c1* dan *r1* juga merupakan tipe data *interface,* maka kita dapat menjadikannya sebagai argumen kedua dari function *print*. Lalu pada argument pertamanya kita memberikan keterang bentuknya.

Jika kita jalankan pada terminal kita, maka hasilnya akan seperti pada gambar kedua. Hasilnya masih sama seperti halaman sebelumnya namun sekarang codingan kita menjadi lebih rapi dengan menerapkan function *print*.

```go
func print(t string, s shape) {
  fmt.Printf("%s area %v\n", t, s.area())
  fmt.Printf("%s perimeter %v\n", t, s.perimeter())
}

func main() {
  var c1 shape = circle{radius: 5}
  var r1 shape = rectangle{width: 3, height: 2}

  print("Rectangle", c1)
  print("Circle", r1)

}
```

```
Rectangle area 78.53981633974483
Rectangle perimeter 31.41592653589793
Circle area 6
Circle perimeter 10
```

Interface (Type assertion)

Ketika struct *circle* menambahkan satu method selain dari method-method yang didefinisikan oleh interface *shape*, maka variable *c1* tidak dapat menggunakan method baru tersebut.

Contohnya seperti gambar pertama di sebelah kanan, terjadi error pada saat compile time ketika variable *c1* mencoba untuk memanggil method *volume* yang dimiliki oleh struct *volume*.

Hal ini terjadi karena interface *shape* tidak mendefinisikan method *volume* dan variable *c1* telah kita buat menjadi sebuah *variable* dengan tipe data interface *shape*.

```
func (c circle) volume() float64 {
  return 4 / 3 * math.Pi * math.Pow(c.radius, 3)
}

func main() {
  var c1 shape = circle{radius: 5}

  c1.volume()
}
```

```
40
41  func (c circle) volume() float64 {
42    return 4 / 3 * math.Pi * math.Pow(c.radius, 3)
43  }
44
45  func main() {
46    var c1 shape = circle{radius: 5}
47
48    c1.(circle).volume()
49  }
```

Interface (Type assertion)

Untuk menanggulangi error tersebut, diperlukan teknik type *assertion* yang dimana teknik ini berfungsi untuk mengembalikan suatu tipe data interface kepada tipe data aslinya.

Jika pada kasus kita kali ini, kita ingin mengembalikan variable *c1* kepada tipe data aslinya yaitu tipe data struct *circle*.

Cara melakukan *type assertion* bisa dilihat seperti pada gambar kedua di sebelah kanan pada line 48.

Format penulisan type assertion adalah namaVariable.(tipe data asli). Dengan menggunakan *type assertion* maka kita dapat mengembalikan suatu tipe data *interface* kembali ke tipe data konkrit atau aslinya.

```
func (c circle) volume() float64 {
  return 4 / 3 * math.Pi * math.Pow(c.radius, 3)
}

func main() {
  var c1 shape = circle{radius: 5}

  c1.volume()
}
```

```
40
41  func (c circle) volume() float64 {
42    return 4 / 3 * math.Pi * math.Pow(c.radius, 3)
43  }
44
45  func main() {
46    var c1 shape = circle{radius: 5}
47
48    c1.(circle).volume()
49  }
```

Interface (Type assertion)

Kita juga dapat memeriksa apakah sebuah *type assertion* yang kita gunakan berhasil atau tidak. Caranya dengan memberikan 2 variable opsional yang menjadi penampung dari hasil yang akan dikembalikan oleh type assertionnya.

Contohnya seperti pada line 48. Variable *value* akan menerima tipe data asli dan nilai yang sudah diberikan kepada field dari struct *circle* jika *type assertion* berhasil, dan variable *ok* akan menerima nilai true jika *type assertion* berhasil dan nilai false jika *type assertion* gagal.

Lalu pada line 50, kita melakukan pengecekan terhadap variable *ok,* jika nilai true maka kita akan menampilkan nilai keseluruhan dari struct *circle,* dan menampilkan hasil dari pemanggilan method *volume* yang dimiliki oleh struct *circle*.

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua dibawah.

```
45  func main() {
46      var c1 shape = circle{radius: 5}
47
48      value, ok := c1.(circle)
49
50      if ok == true {
51          fmt.Printf("Circle value: %+v\n", value)
52          fmt.Printf("Circle volume: %f", value.volume())
53      }
54  }
```

```
Circle value: {radius:5}
Circle volume: 392.699082
```