Golang Generic

- Pengenalan Generic
  - Manfaat Generic
- Type Parameter
  - Type Constraint
  - Type Data any
  - Menggunakan Type Parameter
- Multiple Type Parameter
- Comparable
- Type Parameter Inheritance
- Type Sets
  - Type Approximation
  - Type Inference
- Generic Type
- Generic Struct
- Generic Method
- Generic Interface
- In Line Type Constraint
- Generic di Type Paramter
- Experimental package
  - Constraints Package
  - Maps & Slices Packages

Pengenalan Generic

Generic adalah kemampuan menambahkan parameter type saat membuat function.

Berbeda dengan tipe data yang biasa kita gunakan di function, generic memungkinkan kita bisa mengubah-ubah bentuk tipe data sesuai dengan yang kita mau.

Fitur generic baru ada sejak Golang versi 1.18

Manfaat Generic

- Pengecekan ketika proses kompilasi
- Tidak perlu manual menggunakan pengecekan tipe data dan konversi tipe data
- Memudahkan programmer membuat kode program yang generic sehingga bisa digunakan oleh berbagai tipe data.

Contoh kode bukan generic

```go
func SumInt(values []int) int {
        var sum = 0
        for _, value := range values {
                sum += value
        }
        return sum
}
```

```go
func SumFloat(values []float64) float64 {
        for sum float64 = 0
        for _, value := range value {
                sum += value
        }
        return sum
}
```

Type Parameter

Untuk menandai sebuat function merupakan tipe generic, kita perlu manambahkan Type parameter pada function tersebut.

Pembuatakn type parameter menggunakan tanda [] (kurung kotak), dimaan di dalam kurung kota tersebut, kita tentukan nama Type Permeternya

Hampir sama dengan di bahasa pemrograman lain seperti Java, C# dan lain-lain, biasanaya nama type parameter hanya menggunakan satu huruf misal, T, K, V dan lain-lain. Walaupun bisa saja lebih dari satu huruf

Kode type parameter

```go
func Lenght[T] () {}
```

Type Constraint

Dibahasa pemrograman seperti Java, C# dan lain-lain, Type parameter biasanya tidak perlu kita tentukan tipe datanya, berbeda dengan di Golang.

Dari pengalaman yang dilakukan para pengembang golang, akhirnya di golang type parameter wajib memiliki constraint.

Type Constraint merupakan aturan yang digunakan untuk menentukan tipe data yang diperbolehkan pada Type Parameter.

Contoh, jika kita ingin type parameter bisa digunakan untuk semua tipe data, kita bisa gunakan interface{} (kosong) sebagai contraint nya.

Type Constraint yang lebih detail akan kita bahas di materi Type Sets.

Kode Type Constraint

```language-go
func Length[T interface{}]() {}
```

Type Data any

Digolang 1.18 diperkenalkan alias baru bernama any untuk interface{} kosong, ini bisa mempermudah kita membuat Type Parameter dengan constraint interface{}, jadi kita cukup gunakan constraint any.

```
// any is an alias for interface{} and is equivalent to interface{} in all ways.
type any = interface{}
```

Kode Type Data any

```language-go
func Length[T any]() {}
```

Menggunakan Type Parameter

Setelah kita buat Type Parameter di function, selanjutnya kita bisa menggunakan Type Parameter tersebut sebagai tipe data didalam function tersebut.

Misal digunakan untuk return type atau function parameter, kita cukup gunakan nama type parameternya saja

Type Parameter hanya bisa digunakan di functionnya saja, tidak bisa digunakan di luar function.

```language-go
func Length[T any](param T) T {
    fmt.Println(param)
    return param
}

func TestLength(T *testing.T) {
    var result string = Length[string]("Danang")
    assert.Equal(T, "Danang", result)

    resultNumber := Length[int](100)
    assert.Equal(T, 100, resultNumber)
}
```

Multiple Type Parameter

Penggunaan Type Parameter bisa lebih dari satu, jika kita ingin menambahkan multiple Type Parameter, kita cukup menggunakan tanda , (koma) sebagai pemisah antara Type

Parameter.

Namun Type Parameter harus berbeda, tidak boleh sama jika kita menambahkan Type Parameter lebih dari satu.

```language-go
func MultipleParameter[T any, V any](param1 T, param2 V) {
    fmt.Println(param1)
    fmt.Println(param2)
}

func TestMultipleParameter(T *testing.T) {
    MultipleParameter[string, int]("Danang", 100)
    MultipleParameter[int, string](100, "Danang")
}
```

Comparable

Selain any, juga terdapat tipe data bernama comparable.

Comparable merupakan interface yang diimplementasikan oleh tipe data yang bisa dibandingkan (menggunakan operator != dan ==) serperti boolean, numbers, strings, pointers, channels, interface, array yang isinya ada comparable type, atau structs yang fieldsnya adalah comparable type

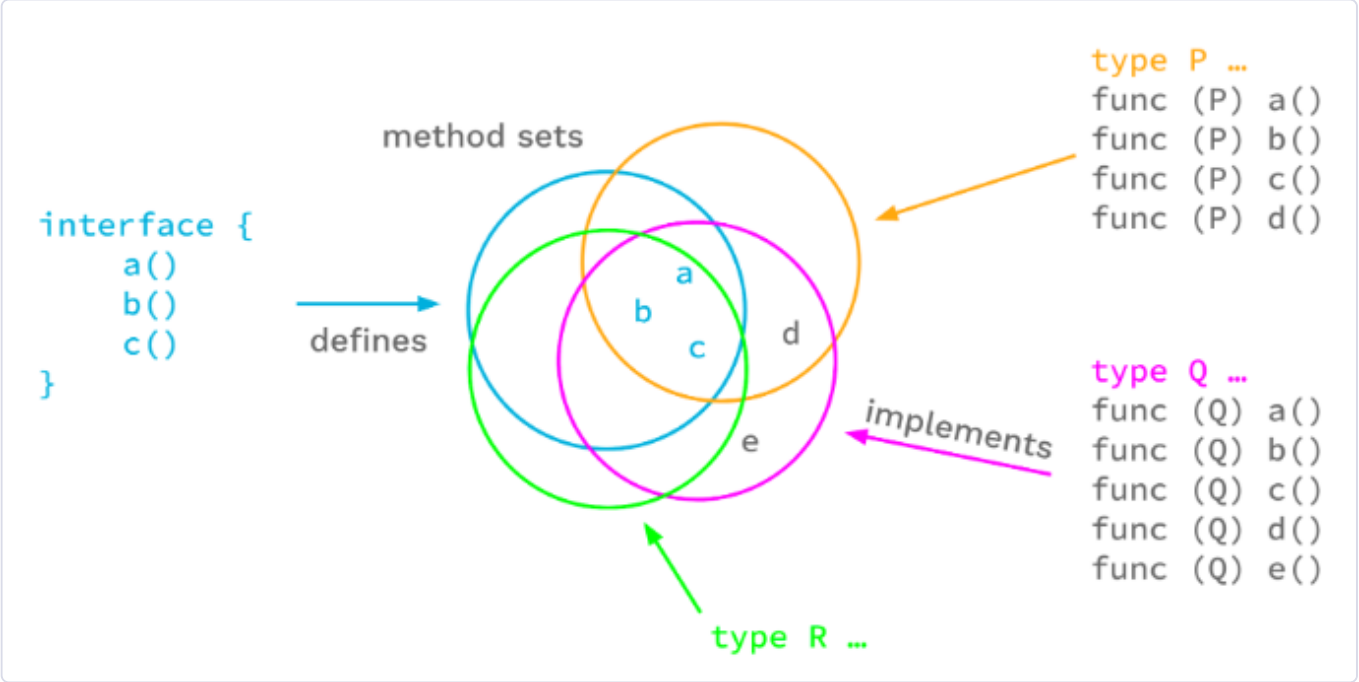
```language-go
func IsSame[T comparable](param1, param2 T) bool {
    if param1 == param2 {
       return true
    } else {
       return false
    }
}

func TestIsSame(T *testing.T) {
    assert.True(T, IsSame[string]("Danang", "Danang"))
    assert.False(T, IsSame[int](100, 200))
}
```

Type Parameter Inheritance

Golang sendiri sebenarnya tidak memiliki pewarisan, namun seperti kita ketahui, jika kita membuat sebuath tipe yang sesuai dengan kontrak interface, maka dianggap sebagai implementasi interface tersebut.

Type Parameter juga mendukung hal serupa, kita bisa gunakan constraint dengan menggunakan interface, maka secara otomatis semua interface yang compatible dengan type constraint tersebut bisa kita gunakan.




```language-go
type Employee interface {
    GetName() string
}

func GetName[T Employee](param T) string {
    return param.GetName()
}

type Manager interface {
    GetName() string
    GetManagerName() string
}

type MyManager struct {
    Name string
}

func (m *MyManager) GetName() string {
    return m.Name
}

func (m *MyManager) GetManagerName() string {
    return m.Name
}

type VicePresident interface {
    GetName() string
    GetVicePresidentName() string
}

type MyVicePresident struct {
```

```
    Name string
}

func (m *MyVicePresident) GetName() string {
    return m.Name
}

func (m *MyVicePresident) GetVicePresidentName() string {
    return m.Name
}

func TestGetName(T *testing.T) {
    assert.Equal(T, "Danang", GetName[Manager](&MyManager{Name: "Danang"}))
    assert.Equal(T, "Danang", GetName[VicePresident](&MyVicePresident{Name:
"Danang"}))
}
```

Type Sets

Dengan Type Sets kita bisa menentukan lebih dari satu tipe constraint yang diperbolehkan pada type parameter.

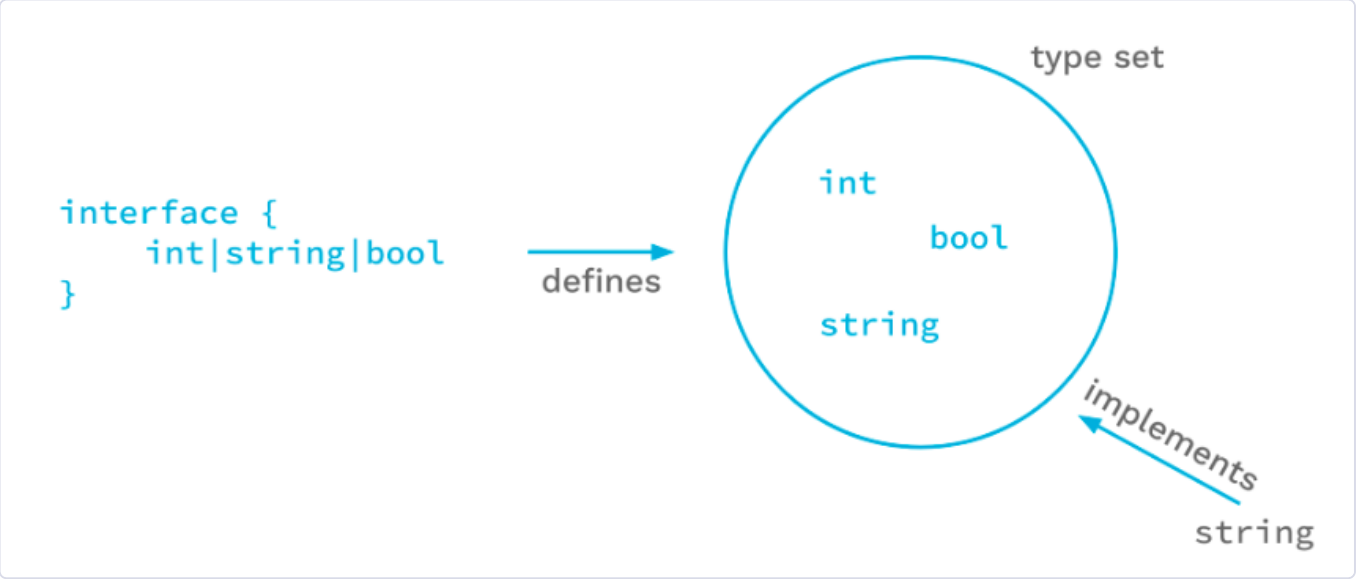
Membuat Type Set

Type sets adalah sebuah ijnterface, membuat Type Set:

```language-go
type NameTypeSet interface {
        P | Q | R
}
```

Type Set hanya bisa digunakan pada type parameter, tidak bisa digunakan sebagai tipe data field atau variable,

Jika operator bisa digunakan disemua tipe data di dalam type set, maka perator tersebut bisa digunakan dalam kode generic.




```language-go
type Number interface {
    int | int8 | int16 | int32 | int64 |
       float32 | float64
}

func Min[T Number](first, second T) T {
    if first < second {
       return first
    } else {
       return second
    }
}

func TestMin(T *testing.T) {
    assert.Equal(T, int(100), Min[int](100, 200))
    assert.Equal(T, int64(100), Min[int64](int64(100), int64(200)))
    assert.Equal(T, float64(100.0), Min[float64](100.0, 200.0))
}
```

Type Approximation

Kadang kita sering membuat Type Declaration di Golang untuk tipe data lain, misal kita membuat type data Age untuk tipe data int

Secara default, jika kita gunakan Age sebagai type declaration untuk int, lalu kita membuat Type Set yang berisi constraint int, maka type data Age dianggap tidak compatible dengan Type Set yang kita buat.

```language-go
type Number interface {
    int | int8 | int16 | int32 | int64 |
       float32 | float64
}
```

```
type Age int
```

```
func TestMin(T *testing.T) {  new *
    assert.Equal(T, int(100), Min[int]( first: 100,  second: 200))
    assert.Equal(T, int64(100), Min[int64](int64(100), int64(200)))
    assert.Equal(T, float64(100.0), Min[float64]( first: 100.0,  second: 200.0))

    assert.Equal(T, Age(20), Min[Age](Age(20), Age(30)))
}
```

Untungnya golang memiliki feature bernama Type Aproximataion, dimana kita bisa menyebutkan bahwa semua constraint dengan tipe tersebut dan juga yang memiliki tipe dasarnya adalah tipe tersebut, maka bisa digunakan.

Untuk menggunakan Type Approximataion, kita bisa mengguanakan tanda ~ (tilde)

```language-go
type Number interface {
    ~int | int8 | int16 | int32 | int64 |
       float32 | float64
}

type Age int
```

Type Inference

Type Inference merupakan fitur dimana kita tidak perlu menyebutkan Type Parameter ketika memanggil kode generic.

Tipe data Type Parameter bisa dibaca secara otomatis misal dari parameter yang kita kirm.

Namum perlu diingat, pada beberapa kasus, jika terjadi error karena Type Inference, kita bisa dengan mudah memperbaikinya dengan cara meneyebutkan Type Paramter nya saja.

```language-go
func TestMin(T *testing.T) {
    assert.Equal(T, int(100), Min[int](100, 200))
    assert.Equal(T, int64(100), Min[int64](int64(100), int64(200)))
    assert.Equal(T, float64(100.0), Min[float64](100.0, 200.0))

    assert.Equal(T, Age(20), Min[Age](Age(20), Age(30)))
}

func TestMinInference(T *testing.T) {
    assert.Equal(T, int(100), Min(100, 200))
    assert.Equal(T, int64(100), Min(int64(100), int64(200)))
    assert.Equal(T, float64(100.0), Min(100.0, 200.0))
    assert.Equal(T, Age(10), Min(Age(10), 10))
}
```

Generic Type

Sebelumnya kita sudah bahasa tentang generic di function. Generic juga bisa digunakan ketika membuat type

```language-go
type Bag[T any] []T

func PrintBag[T any](bag Bag[T]) {
    for _, value := range bag {
       fmt.Println(value)
    }
}

func TestBag(T *testing.T) {
    numbers := Bag[int]{1, 2, 3, 4, 5}
    PrintBag(numbers)

    names := Bag[string]{"Danang", "Haris", "Setiawan"}
    fmt.Println(names)
    PrintBag(names)
}
```

Generic Struct

Struct juga mendukung generic. Dengan menggunakan generic, kita bisa membuat Field dengan tipe data yang sesuai dengan Type Parameter.

```language-go
type Data[T any] struct {
    First  T
    Second T
}

func TestData(T *testing.T) {
    data := Data[string]{
       "Danang",
       "Haris",
    }

    fmt.Println(data)
}
```

Generic Method

Selain di function, kita juga bisa tambahkan generic di method (function di struct). Namun, generic di method merupakan generic yang terdapat di structnya.

Kita wajib menyebutkan semua type parameter yang terdapat di Struct, walaupun tidak kita gunakan misalnya, atau jika tidak ingin kita gunakan, kita bisa gunakan _ (garis bawah)

sebagai pengganti type paramternya.

Method tidak bisa memiliki type paramter yang mirip dengan di function.

```language-go
func (d *Data[_]) SayHello(name string) (message string) {
    message = "Hello " + name
    return
}

func (d *Data[T]) ChangeFirst(first T) T {
    d.First = first
    return first
}

func TestGenericMethod(T *testing.T) {
    data := Data[string]{
       First:  "Danang",
       Second: "Haris",
    }

    assert.Equal(T, "Setiawan", data.ChangeFirst("Setiawan"))
    assert.Equal(T, "Hello Danang", data.SayHello("Danang"))
}
```

Generic Interface

Generic juga bisa digunakan di interface. Secara otomatis, semua struct yang ingin mengikuti kontrak interface tersebut harus menggunakan generic juga.

```language-go
// Generic Interface
type GetterSetter[T any] interface {
    GetValue() T
    SetValue(value T)
}

func ChangeValue[T any](param GetterSetter[T], value T) T {
    param.SetValue(value)
    return param.GetValue()
}

// Implementasi Struct
type MyData[T any] struct {
    Value T
}

func (m *MyData[T]) GetValue() T {
    return m.Value
}
```

```go
func (m *MyData[T]) SetValue(value T) {
    m.Value = value
}

// Test Generic Interface
func TestInterface(t *testing.T) {
    myData := MyData[string]{}
    result := ChangeValue(&myData, "Danang")

    assert.Equal(t, "Danang", result)
}
```

In Line Type Constraint

Sebelumnya, kita selalu menggunakan type declaration atau type set ketika membuat type constraint di type parameter.

Sebenarnya tidak ada kewajiban kita harus membuat type declaration atau type set jika kita ingin membuat type parameter, kita bisa gunakansecara langsung (in line) pada ype constraint, misalnya di awal kita sudah bahas tentang interface {} (kosong), tapi kita selalu gunakan type declaration any.

Jika kita mau, kita juga bisa langsung gunakan interface `{int | float32 | float64}` dibanding membuat type set Number misalnya.

```go
func FindMin[T interface{ int | int64 | float64 }](first, second T) T {
    if first < second {
        return first
    } else {
        return second
    }
}

func TestFindMin(T *testing.T) {
    assert.Equal(T, 100, FindMin(100, 200))
    assert.Equal(T, int64(100), FindMin(int64(100), 200))
    assert.Equal(T, float64(100), FindMin(200, 100.0))
}
```

Generic di Type Paramter

Pada kasus tertentu, kadang ada kebutuhan kita menggunakan type parameter yang ternyata type tersebut jug generica atau memiliki type parameter.

Kita juga bisa menggunakan ***in line type constraint*** agar lebih mudah, dengan cara menambahkan type parameter selanjutnya, misal

- `[S interface{[]E}, E interface{}]`, artinya `S` harus slice elemen `E`, dimana `E` boleh tipe apapun.
- `[S[]E, E any]`, artinya `S` harus slice elemen `E`, diaman `E` boleh tipe apapun

```language-go
func GetFirst[T []E, E any](data T) E {
    first := data[0]
    return first
}

func TestGetFirst(t *testing.T) {
    names := []string{
       "Danang", "Haris", "Setiawan",
    }

    first := GetFirst[[]string, string](names)
    assert.Equal(t, "Danang", first)
}
```

Experimental package

Saat versi Golang 1.18 terdapat experimental package yang banyak menggunakan fitur Generic, namun belum resmi masuk ke golang standard library

Kedepanya, karena ini masih experimental (percobaan), bisa jadi package ini akan berubah atau bahkan mungkin di hapus.

- **https://pkg.go.dev/golang.org/x/exp**

Silahkan install sebagai dependency di Go Modules menggunakan perintah
`go get golang.org/x/exp`

Constraints Package

Constraints Package berisi type declaraton yang bisa kita gunakan untuk tipe data bawaan Golang, misal Number, Complex, Ordered, dan lain-lain.

- **https://pkg.go.dev/golang.org/x/exp/constraints**

Maps & Slices Packages

Terdapat juga package maps dan slices, yang berisi function untuk mengelola data Map dan Slice, namun sudah menggunakan fitur Generic

- https://pkg.go.dev/golang.org/x/exp/maps
- https://pkg.go.dev/golang.org/x/exp/slices