

# Hari 10 - Package & Import

## Package & Import

Pengembangan aplikasi dalam *real development* pasti membutuhkan banyak sekali file program. Tidak mungkin dalam sebuah project semua file memiliki nama package `main`, biasanya akan dipisah sebagai package berbeda sesuai bagiannya.

Project folder selain berisikan file-file `.go` juga bisa berisikan sub-folder lainnya. Di Go, setiap folder atau sub-folder adalah satu package, file-file yang ada di dalam sebuah folder package-nya harus sama. Dan package pada file-file tersebut harus berbeda dengan package pada file-file lainnya yang berada pada folder berbeda.

Jadi mudahnya, 1 folder adalah 1 package.

Dalam sebuah package, biasanya kita menulis sangat banyak komponen, entah itu fungsi, struct, variabel, atau lainnya. Komponen tersebut bisa leluasa digunakan dalam package yang sama. Contoh sederhananya seperti program yang telah kita praktekan di bab sebelum-sebelumnya, dalam package `main` ada banyak yang di-*define*: fungsi, variabel, closure, struct, dan lainnya; kesemuanya bisa langsung dimanfaatkan.

Jika dalam satu program terdapat lebih dari 1 package, atau ada package lain selain `main`, maka komponen dalam package lain tersebut tidak bisa diakses secara bebas dari file yang package-nya `main`, karena tiap komponen memiliki hak akses.

Ada 2 jenis hak akses di Go:

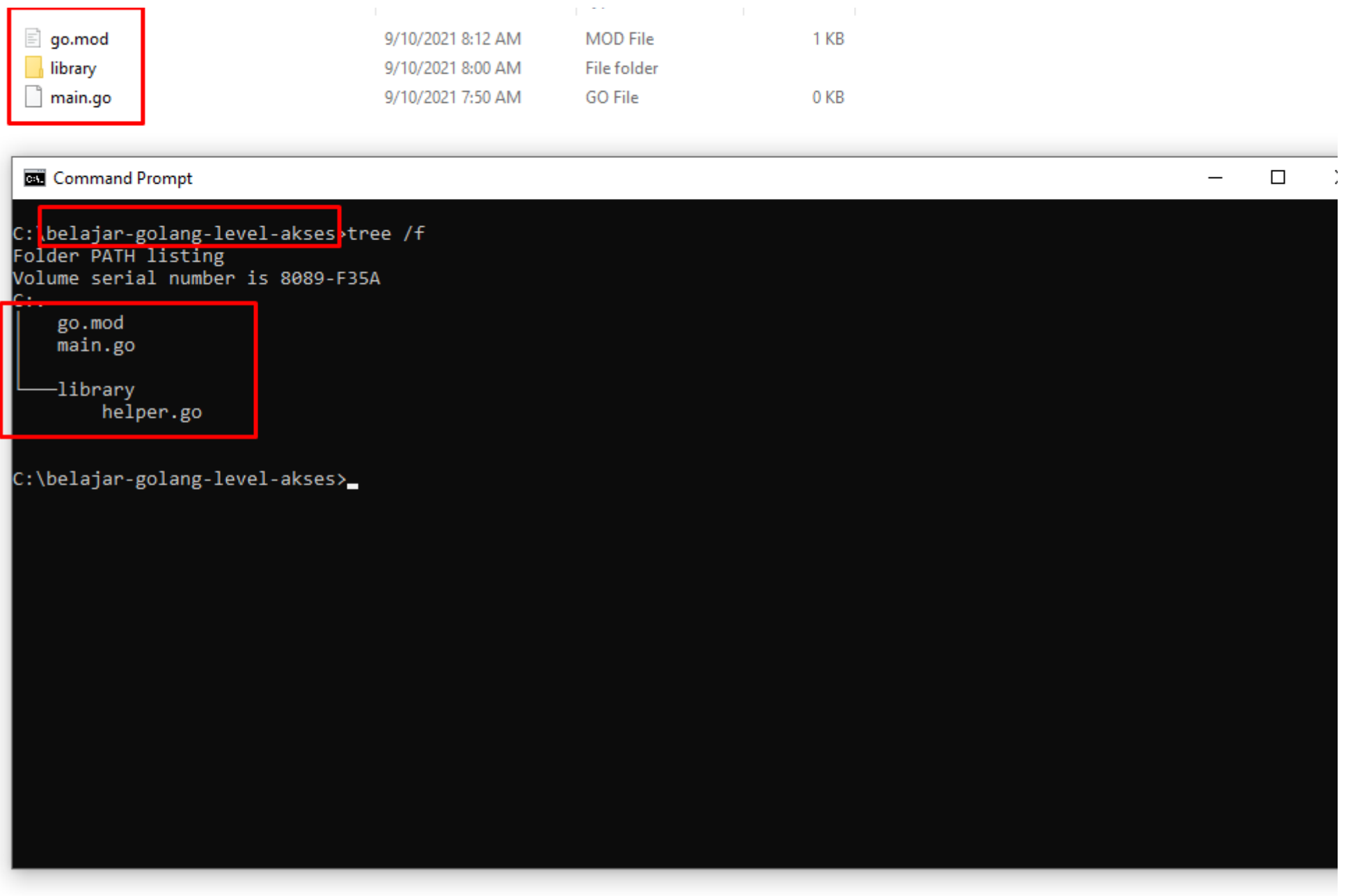
- Hak akses **Exported** atau **public**. Menandakan komponen tersebut diperbolehkan untuk diakses dari package lain yang berbeda
- Hak akses **Unexported** atau **private**. Berarti komponen hanya bisa diakses dalam package yang sama, bisa dalam satu file saja atau dalam beberapa file yang masih 1 folder.

Di Go cara menentukan level akses atau modifier sangat mudah, penandanya adalah **character case** huruf pertama nama fungsi, struct, variabel, atau lainnya. Ketika namanya diawali dengan huruf besar menandakan *exported* (atau *public*). Dan sebaliknya, jika diawali huruf kecil, berarti *unexported* (atau *private*).

## 1. Penggunaan Package & Import

Pertama buat folder proyek baru bernama `belajar-golang-level-akses`, inialisasi sebagai proyek dengan nama yang sama. Kemudian buat file baru bernama `main.go` di dalamnya, lalu set nama package file tersebut sebagai **main**.

Kemudian, buat sub-folder baru bernama `library` di dalam folder `belajar-golang-level-akses`. Didalam folder `library`, buat file baru `library.go`, set nama package-nya **library**.



Buka file `library.go` lalu isi dengan kode berikut.

```
package library

import "fmt"

func SayHello() {
    fmt.Println("hello")
}

func introduce(name string) {
    fmt.Println("nama saya", name)
}
```

File `library.go` yang telah dibuat ditentukan nama package-nya adalah `library` (sesuai dengan nama folder), berisi dua buah fungsi, `SayHello()` dan `introduce()`.

- Fungsi `SayHello()`, level aksesnya adalah publik, ditandai dengan nama fungsi diawali huruf besar.
- Fungsi `introduce()` dengan level akses private, ditandai oleh huruf kecil di awal nama fungsi.

Selanjutnya kita lakukan tes apakah memang fungsi yang ber-modifier private dalam package `library` tidak bisa diakses dari package lain.

Buka file `main.go`, lalu tulis kode berikut.

```
package main

import "belajar-golang-level-akses/library"

func main() {
    library.SayHello()
    library.introduce("john")
}
```

Bisa dilihat bahwa package `library` yang telah dibuat tadi, di-import ke dalam package `main`.

Folder utama atau root folder dalam project yang sedang digarap adalah `belajar-golang-level-akses`, sehingga untuk import package lain yang merupakan subfolder, harus dituliskan lengkap path folder nya, seperti `belajar-golang-level-akses/library`.

Penanda root folder adalah tempat dimana file `go.mod` berada.

Ok, kita lanjut. Perhatikan kode berikut.

```
library.SayHello()
library.introduce("john")
```

Cara pemanggilan fungsi yang berada dalam package lain adalah dengan menuliskan nama package target diikuti dengan nama fungsi menggunakan *dot notation* atau tanda titik, seperti `library.SayHello()` atau `library.introduce("john")`

OK, sekarang coba jalankan kode yang sudah disiapkan di atas, hasilnya error.

```
C:\belajar-golang-level-akses>go run main.go
# command-line-arguments
.\main.go:7:2: cannot refer to unexported name library.introduce

C:\belajar-golang-level-akses>
```

Error di atas disebabkan karena fungsi `introduce()` yang berada dalam package `library` memiliki level akses *unexported* (atau *private*), fungsi ini tidak bisa diakses dari package lain (pada kasus ini `main`). Agar bisa diakses, solusinya bisa dengan menjadikannya ke bentuk *exported* (atau *public*), atau diubah cara pemanggilannya. Disini kita menggunakan cara ke-2.

Tambahkan parameter `name` pada fungsi `SayHello()`, lalu panggil fungsi `introduce()` dengan menyisipkan parameter `name` dari dalam fungsi `SayHello()`.

```
func SayHello(name string) {
    fmt.Println("hello")
    introduce(name)
}
```

Di `main`, cukup panggil fungsi `SayHello()` saja, sisipkan sebuah string sebagai parameter.

```
func main() {
    library.SayHello("john")
}
```

Coba jalankan lagi.

```
C:\belajar-golang-level-akses>go run main.go
hello
nama saya john

C:\belajar-golang-level-akses>_
```

## 2. Import Dengan Prefix Tanda Titik

Seperti yang kita tahu, untuk mengakses fungsi/struct/variabel yg berada di package lain, nama package nya perlu ditulis, contohnya seperti pada penggunaan penggunaan `library.Student` dan `fmt.Println()`.

Di Go, komponen yang berada di package lain yang di-import bisa dijadikan se-level dengan komponen package peng-import, caranya dengan menambahkan tanda titik `.` setelah penulisan keyword `import`. Maksud dari se-level disini adalah, semua properti di package lain yg di-import bisa diakses tanpa perlu menuliskan nama package, seperti ketika mengakses sesuatu dari file yang sama.

```
)

func main() {
    SayHello("john")
}
```

Pada kode di atas package `library` di-import menggunakan tanda titik. Dengan itu, pemanggilan struct `Student` tidak perlu dengan menuliskan nama package nya.

### 3. Pemanfaatan Alias Ketika Import Package

Fungsi yang berada di package lain bisa diakses dengan cara menuliskan nama-package diikuti nama fungsi-nya, contohnya seperti `fmt.Println()`. Package yang sudah di-import tersebut bisa diubah namanya dengan cara menggunakan alias pada saat import. Contohnya bisa dilihat pada kode berikut.

```
import (
    f "fmt"
)




func main() {
    f.Println("Hello World!")
}
```

Pada kode di-atas, package `fmt` di tentukan aliasnya adalah `f`, untuk mengakses `Println()` cukup dengan `f.Println()`.

### 4. Mengakses Properti Dalam File Yang Package-nya Sama

Jika properti yang ingin di akses masih dalam satu package tapi berbeda file, cara mengaksesnya bisa langsung dengan memanggil namanya. Hanya saja ketika eksekusi, file-file lain yang yang nama package-nya sama juga ikut dipanggil.

Langsung saja kita praktekan, buat file baru dalam `belajar-golang-level-akses` dengan nama `partial.go`.

C: > Local Disk (C:) > belajar-golang-level-akses					
Name	Date	Type	Size	Length	
 go.mod	9/10/2021 8:12 AM	MOD File	1 KB		
 main.go	9/10/2021 7:50 AM	GO File	1 KB		
 partial.go	9/10/2021 9:10 AM	GO File	1 KB		

Tulis kode berikut pada file `partial.go`. File ini kita set package-nya `main` (sama dengan nama package file `main.go`).

```
package main

import "fmt"

func sayHello(name string) {
    fmt.Println("halo", name)
}
```

Hapus semua isi file `main.go`, lalu silakan tulis kode berikut.

```
package main

func main() {
    sayHello("john")
}
```

Sekarang terdapat 2 file berbeda (`main.go` dan `partial.go`) dengan package adalah sama, `main`. Pada saat `go build` atau `go run`, semua file dengan nama package `main` harus dituliskan sebagai argumen command.

**go run main.go partial.go**

Fungsi `sayHello` pada file `partial.go` bisa dikenali meski level aksesnya adalah `unexported`. Hal ini karena kedua file tersebut (`main.go` dan `partial.go`) memiliki package yang sama.

```
C:\belajar-golang-level-akses>go run main.go partial.go
halo john

C:\belajar-golang-level-akses>_
```

### 3. Package Initialization

Saat kita membuat package, kita bisa membuat sebuah function yang akan diakses ketika package kita diakses Ini sangat cocok ketika contohnya, jika package kita berisi function-function untuk berkomunikasi dengan database, kita membuat function inisialisasi untuk membuka koneksi ke database Untuk membuat function yang diakses secara otomatis ketika package diakses, kita cukup membuat function dengan nama init.

berikut contohnya terdapat kode seperti di bawah ini pada helper.go:

```
var DatabaseConnection string

func init() {
    DatabaseConnection = "MySQL"

    fmt.Println("Melakukan init")
}
```

dan pada main.go:

```
import (
    "fmt"
    "belajar-golang-level-akses/library"
)

func main() {
    fmt.Println(library.DatabaseConnection)
}
```

Dapat terlihat diatas bahwa terdapat sebuah variabel global DatabaseConnection yang di assign dengan value "MySQL" dan pada function main di print

Kadang kita hanya ingin menjalankan init function di package tanpa harus mengeksekusi salah satu function yang ada di package Secara default, Go akan komplain ketika ada package yang di import namun tidak digunakan Untuk menangani hal tersebut, kita bisa menggunakan blank identifier ( \_ ) sebelum nama package ketika melakukan import

```
import (
    "fmt"
    _"belajar-golang-level-akses/library"
)

func main() {
    //
    fmt.Println(library.DatabaseConnection)
}
```

---

### Referensi Video:

- [Package & Import](#) (Programmer Zaman Now)
- [Access Modifier](#) (Programmer Zaman Now)
- [Package Initialization](#) (Programmer Zaman Now)

### Referensi Tulisan:

- <https://dasarpemrogramangolang.novalagung.com/A-property-public-dan-private.html>

## Rating - Feedback



Berikan kritik dan saran..

Submit