Pertemuan 12

# ENCAPSULATION (PEMBUNGKUS)

# Pengenalan

- Enkapsulasi adalah pembungkus, pembungkus disini dimaksudkan untuk menjaga suatu proses program agar tidak dapat diakses secara sembarangan atau di intervensi oleh program lain.
- Konsep enkapsulasi sangat penting dilakukan untuk menjaga kebutuhan program agar dapat diakses sewaktu-waktu, sekaligus menjaga program tersebut.
- Dalam kehidupan sehari hari enkapsulasi dapat dimisalkan sebagai arus listrik pada generator, dan sistem perputaran generator untuk menghasilkan arus listrik.
- Kerja arus listrik tidak mempengaruhi kerja dari sistem perputaran generator, begitu pula sebaliknya. Karena didalam arus listrik tersebut, kita tidak perlu mengetahui bagaimana kinerja sistem perputaran generator, apakah generator berputar kebelakang atau ke depan atau bahkan serong.
- Begitu pula dalam sistem perputaran generator, kita tidak perlu tahu bagaimana arus listrik, apakah menyala atau tidak.
- Begitulah konsep kerja dari enkapsulasi, dia akan melindungi sebuah program dari akses ataupun intervensi dari program lain yang mempengaruhinya.
- Hal ini sangat menjaga keutuhan program yang telah dibuat dengan konsep dan rencana yang sudah ditentukan dari awal.

# Access Control

| Modifier | class yang sama | package yang sama | subclass package lain | class manapun |
|---|---|---|---|---|
| private | √ | | | |
| default | √ | √ | | |
| protected | √ | √ | √ | |
| public | √ | √ | √ | √ |

# Hak akses

- Private : hanya diakses class itu sendiri
- Public : dapat diakses dari manapun
- Protected : hanya dapat diakses dari package (satu folder) dan subclass
- Default : tanpa modifier, hanya bisa diakses dari package dan class itu sendiri

# private

- Variabel dan method yang dideklarasikan private hanya bisa diakses oleh class yang mendeklarasikan variabel dan method tersebut.
- Contoh mengakses private variable dari subclass

```
1. class Complex {
2.       private double real, imaginary;
3. }
4.
5.
6. class SubComplex extends Complex {
7.       SubComplex(double r, double i) {
8.          real = r; // Trouble!
9.       }
10.}
```

- Contoh mengakses private variable dari class lain

```
1.  class Complex {
2.      private double real, imaginary;
3.
4.      public Complex(double r, double i) {
5.              real = r; imaginary = i;
6.      }
7.      public Complex add(Complex c) {
8.              return new Complex(real + c.real,
9.              imaginary + c.imaginary);
10.     }
11. }
12.
13. class Client {
14.     void useThem() {
15.             Complex c1 = new Complex(1, 2);
16.             Complex c2 = new Complex(3, 4);
17.             Complex c3 = c1.add(c2);
18.             double d = c3.real; // Illegal!
19.     }
20. }
```

# default

- ***default*** bukan merupakan Java keyword
- Merupakan jenis akses kontrol jika kita tidak menuliskan akses kontrol secara eksplisit
- Semua feature class-class yang ada dalam satu package bisa diakses oleh semua yang ada dalam package tersebut
- Class diluar package boleh melakukan subclass, tetapi subclass tersebut tidak bisa mengakses feature superclass.

# Contoh default

```
1. package sportinggoods;
2. class Ski {
3.     void applyWax() { . . . }   → default access
4. }

1. package sportinggoods;
2. class DownhillSki extends Ski {
3.     void tuneup() {
4.             applyWax();   → OK
5.             // other tuneup functionality here
6.     }
7. }
```

# protected

- Protected mempunyai kemampuan akses yang lebih besar daripada private dan default
- Protected feature dari suatu class bisa diakses oleh semua class dalam satu package
- Class diluar package boleh melakukan melakukan subclass, dan subclass tersebut bisa mengakses feature superclass.

# Contoh protected

```
1. package adifferentpackage; // Class Ski now in
// a different package
2. class Ski {
3.     protected void applyWax() { . . . }
4. }

1. package sportinggoods;
2. class DownhillSki extends Ski {
3.     void tuneup() {
4.             applyWax(); → OK
5.             // other tuneup functionality here
6.     }
7. }
```