PPLG (Pengembangan Perangkat Lunak dan Gim)

PEMROGRAMAN APLIKASI BERGERAK

Modul 4 - OOP Lanjutan (Visibility Modifiers)

Tujuan

1. Peserta didik mampu memahami (C2) jenis dan peran Visibility Modifiers dengan benar.
2. Peserta didik mampu menerapkan (C3) Visibility Modifiers untuk mengatur aksesibilitas properti dan fungsi dalam class dengan tepat.

Penyusun

Faishal Fariz Hidayatullah

Modul Pemrograman Aplikasi Bergerak
PPLG (Pengembangan Perangkat Lunak dan Gim)
SMKN 5 Malang

“Pemrograman mengajarkanmu bahwa tidak ada masalah yang tak terpecahkan—semua ada solusinya dengan usaha dan kreativitas”.

Daftar Isi

Bab I - Pendahuluan

Apa itu *Visibility Modifiers* (Enkapsulasi)?

Encapsulation adalah salah satu prinsip dasar dalam *Object-Oriented Programming* (OOP) yang bertujuan untuk membungkus data dalam satu unit.

Tujuan utama *encapsulation* adalah untuk menyembunyikan detail internal implementasi objek dari dunia luar dan menyediakan interface yang dapat berinteraksi dengan objek tersebut.

Enkapsulasi dapat dilakukan dengan mendeklarasikan semua variabel di suatu unit (kelas) sebagai privat dan menulis metode publik di dalam kelas tersebut untuk mengatur dan mendapatkan nilai variabel.

Manfaat Enkapsulasi

Melansir dari ***geeksforgeeks.org***, berikut ini adalah manfaat *encapsulation*:

1. **Menyembunyikan data**
   Encapsulation dapat menyembunyikan detail implementasi internal suatu kelas. Hal ini berarti bahwa pengguna kelas tidak perlu mengetahui bagaimana data disimpan dalam variabelnya.

2. **Meningkatkan fleksibilitas**
   Dengan enkapsulasi, *programmer* dapat mengontrol akses ke elemen pada kelas dan menyesuaikan tingkat visibilitas sesuai dengan kebutuhan.

3. **Dapat digunakan kembali**
   Encapsulation mendukung penggunaan kembali karena detail implementasi internal suatu kelas disembunyikan dari entitas eksternal.

4. **Mudah menguji kode**

   Kode yang dienkapsulasi mudah untuk diuji, terutama selama pengujian unit. Karena kelas memiliki *interface* yang jelas, fungsionalitas setiap metode dapat diuji secara terpisah dengan lebih mudah.

5. **Memberi kebebasan bagi programmer**

   Enkapsulasi memberdayakan *programmer* dengan memberikan kebebasan untuk mengubah detail implementasi internal suatu kelas tanpa mempengaruhi *interface* eksternal yang digunakan oleh bagian lain dari program.

Macam-Macam Hak Akses

Kali ini kita akan mengenal beberapa tentang *visibility modifiers* atau hak akses pada Kotlin. Tentunya, tidak semua properti dan method pada sebuah kelas memiliki hak akses publik. Ada beberapa yang hanya dapat diakses dari dalam dan ada yang dapat diakses dari luar kelasnya. Dengan menentukan hak akses tersebut, kita dapat membatasi akses data pada sebuah kelas, inilah yang disebut dengan encapsulation pada salah satu pilar OOP.

Berikut macam-macam hak akses dan penjelasan singkatnya yang dapat digunakan pada Kotlin:

No.	Hak Akses	Fungsi
1.	***Public***	Hak akses yang cakupannya paling luas. Anggota yang diberi modifier ini dapat diakses dari manapun.
2.	***Private***	Hak akses yang cakupannya paling terbatas. Anggota yang menerapkannya hanya dapat diakses pada *scope* yang sama.
3.	***Protected***	Hak akses yang cakupannya terbatas pada hirarki kelas. Anggota hanya dapat diakses pada kelas turunannya atau kelas itu sendiri.
4.	***Internal***	Hak akses yang cakupannya terbatas pada satu modul. Anggota yang menggunakannya tidak dapat diakses diluar dari modulnya tersebut.

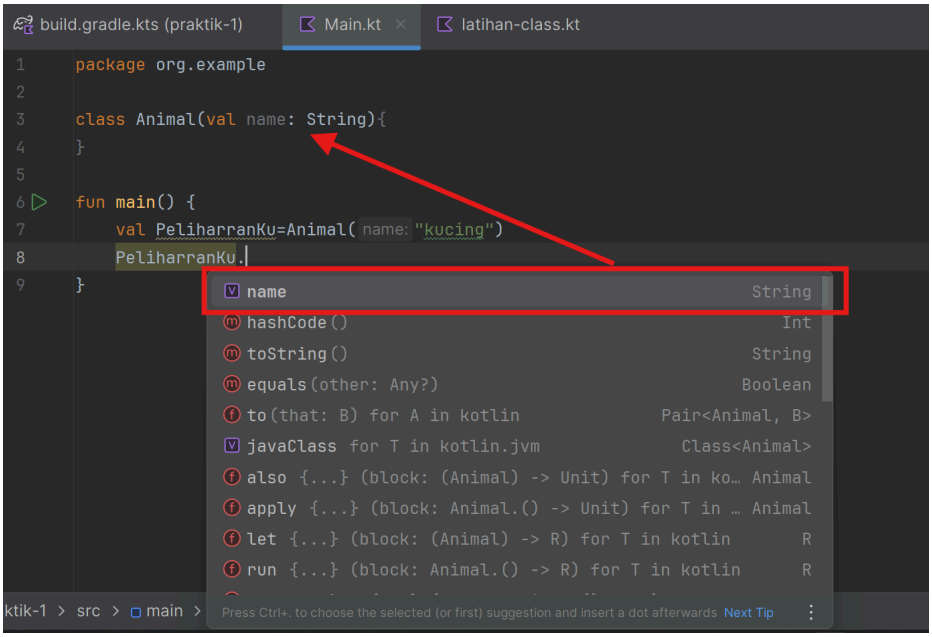
Semua *modifier* tersebut bisa digunakan untuk kelas, objek, konstruktor, fungsi, beserta properti yang ada di dalamnya. Kecuali *modifier protected* yang hanya bisa digunakan untuk **anggota di dalam sebuah kelas dan *interface.*** *Protected* tidak bisa digunakan pada *package member* seperti kelas, objek, dan yang lainnya. Setelah mengetahui pentingnya hak akses, selanjutnya kita akan membahas bagaimana kita menentukan hak akses *public, private, protected* dan internal pada Kotlin.

Bab II - Hak Akses Pada *Visibility Modifiers*

Public

Berbeda dengan bahasa pemrograman umumnya, *default modifier* pada Kotlin adalah ***public***. Ketika sebuah anggota memiliki hak akses *public* maka anggota tersebut dapat diakses dari luar kelasnya melalui sebuah objek kelas tersebut.

Pada pembahasan sebelumnya kita sudah memiliki sebuah kelas *Animal* dengan properti publik seperti *name, age, weight* dan *isMammal*. Properti tersebut dapat kita akses dari luar kelas Animal.



Dari *completion suggestion* terlihat bahwa properti tersebut dapat kita akses di luar dari kelasnya.

Private

Ketika suatu anggota memiliki hak akses *private,* maka anggota tersebut tidak dapat diakses dari luar *scope*-nya. Untuk menggunakan *modifier private* kita perlu menambahkan ***keyword private*** seperti contoh berikut:

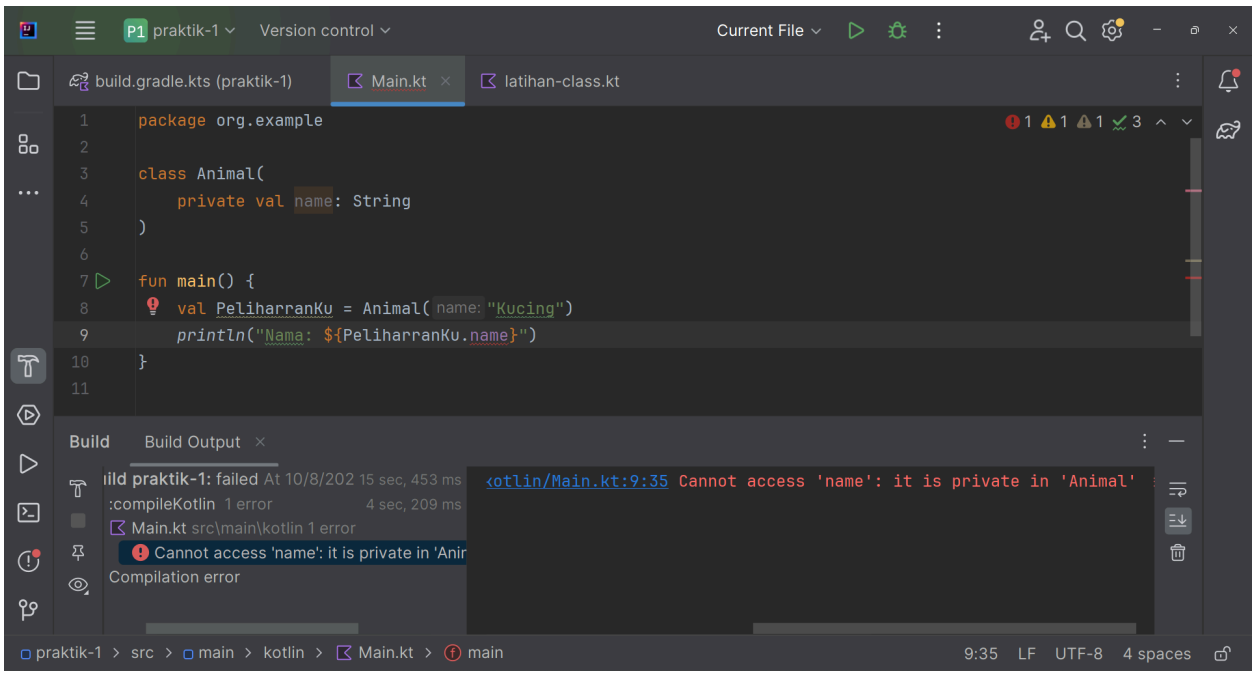
```
private val name: String,
```

Mari kita coba ubah hak akses pada seluruh properti kelas ***Animal* menjadi *private.***

```
class Animal(
    private val name: String
)

fun main() {
    val PeliharranKu = Animal("Kucing")
    println("Nama: ${PeliharranKu.name}")
}
```

Dengan menggunakan hak akses *private*, maka kita tidak diizinkan untuk mengakses properti pada kelas ***Animal*** tersebut dari luar kelasnya. Anda akan berjumpa dengan ***error Cannot access '[PROPERTY]': it is private in 'Animal'***.



Satu satunya **cara untuk mengakses properti private** dari sebuah kelas adalah dengan menambahkan **fungsi *getter* dan *setter* secara manual.** Fungsi *getter* dan *setter* sebenarnya dihasilkan secara otomatis oleh Kotlin ketika properti tersebut memiliki hak akses *public* tetapi tidak untuk *private*. Untuk penulisan *getter* dan *setter* pada hak akses *private* sama seperti fungsi pada umumnya:

```
class Animal(private var name: String){
    fun getName() : String {
        return name
    }
    fun setName(newName: String) {
        name = newName
    }
}

fun main() {
    val PeliharranKu = Animal("Kucing")
    println(PeliharranKu.getName())
    PeliharranKu.setName("Banteng")
    println(PeliharranKu.getName())
}
```

Fungsi **getName()** bertujuan untuk mengembalikan nilai **name** yang memiliki tipe data String. Kemudian fungsi **setName()** bertujuan untuk mengubah nilai properti **name** dengan nilai baru. Fungsi **setName()** membutuhkan satu parameter bertipe String yang nantinya akan dimasukkan nilainya ke dalam properti **name**.

Pada kode di atas, terlihat bahwa kita berhasil mengubah nilai properti **name** dari nilai awal yang kita inisialisasikan pada konstruktor. Ia menjadi nilai baru yang kita tentukan dengan menggunakan fungsi **setName()**.

Protected

Hak akses **protected** mirip seperti **private**, namun pembatasannya lebih luas dalam sebuah hirarki kelas. Hak akses *protected* digunakan ketika kita menginginkan sebuah anggota dari induk kelas dapat diakses hanya oleh kelas yang merupakan turunannya. Perhatikan kode di bawah ini untuk contoh penggunaan hak akses *protected*.

```
open class Animal(val name: String, protected val weight: Double)

class Cat(pName: String, pWeight: Double) : Animal(pName, pWeight)
```

Pada kode tersebut, properti **weight** pada kelas **Animal** memiliki hak akses protected. Kita tetap bisa mengaksesnya dari kelas **Cat** yang termasuk dalam hirarki kelas **Animal**. Namun kita tidak dapat mengakses properti tersebut secara langsung dari luar hirarki kelasnya. Eror akan terjadi jika kita melakukan hal tersebut.

```
open class Animal(val name: String, protected val weight: Double)

class Cat(pName: String, pWeight: Double) : Animal(pName, pWeight)


fun main() {
    val cat = Cat("Dicoding Miaw", 2.0)
    println("Nama Kucing: ${cat.name}")
    println("Berat Kucing: ${cat.weight}") // error: expecting a
top level declaration
}
```

Internal

Internal merupakan hak akses baru yang diperkenalkan pada Kotlin. Hak akses ini membatasi suatu anggota untuk dapat diakses hanya pada satu modul. Berikut ini contoh penggunaan hak akses internal:

```
internal class Animal(val name: String)
```

Pada contoh di atas, kelas Animal telah ditetapkan sebagai kelas internal, maka kelas tersebut hanya dapat diakses dari modul yang sama. Hak akses ini sangat berguna ketika kita mengembangkan sebuah aplikasi yang memiliki beberapa modul di dalamnya.

Bab III - Latihan *Visibility Modifiers*

Petunjuk pengerjaan soal:

1. Soal dikerjakan secara **BERKELOMPOK.**
2. Soal dikerjakan pada **LKPD YANG SUDAH DISEDIAKAN PADA GCR.**
3. Soal dikerjakan **SESUAI PADA WAKTU YANG TELAH DITENTUKAN (30 menit).**
4. **SETIAP NOMOR SOAL DIKERJAKAN SESUAI DENGAN KELOMPOK YANG TELAH DIBENTUK (1 KELOMPOK 1 STUDI KASUS).**
5. **BOLEH BROWSING NAMUN TIDAK BOLEH MENGGUNAKAN CHATGPT YA!**
6. Jika kalian belum memahami instruksi yang diberikan di dalam **SOAL**, mintalah penjelasan dari bapak/ibu guru.
7. Setelah selesai mengerjakan soal, persiapkan diri kalian untuk melakukan **presentasi penjelasan kode**.
8. **SOAL BISA DILIHAT DI:**
   https://github.com/faishalfhid/pplg-1-github/tree/main/Tugas%20Studi%20Kasus