PPLG (Pengembangan Perangkat Lunak dan Gim)

PEMROGRAMAN APLIKASI BERGERAK

Modul 2 - 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

“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.

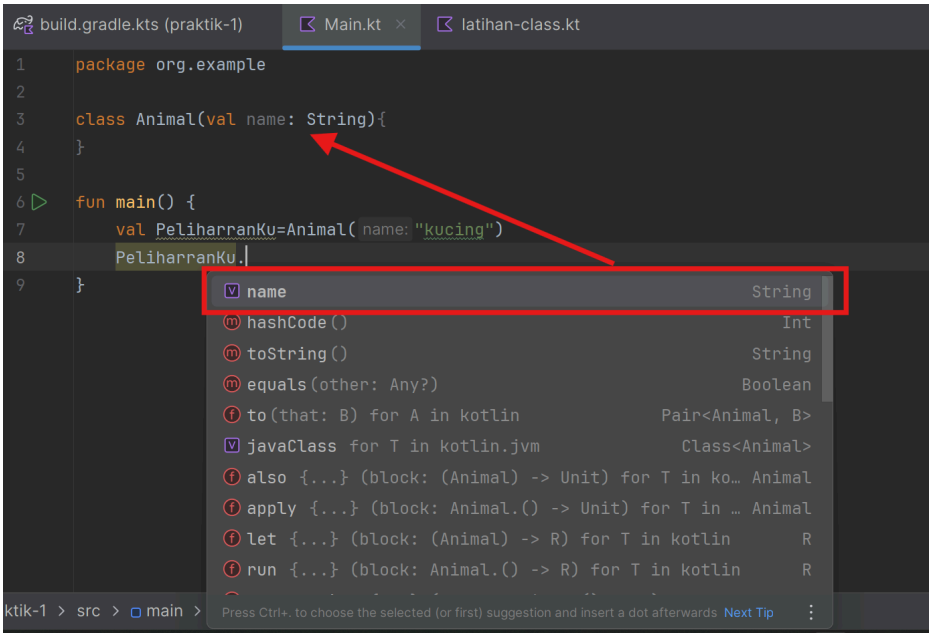
Semua modifier tersebut bisa digunakan untuk kelas, objek, konstruktor, fungsi, beserta properti yang ada di dalamnya. Kecuali modifier protected yang hanya bisa digunakan untuk **anggota di dalam sebuah kelas dan interface.** Protected tidak bisa digunakan pada package member seperti kelas, objek, dan 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:

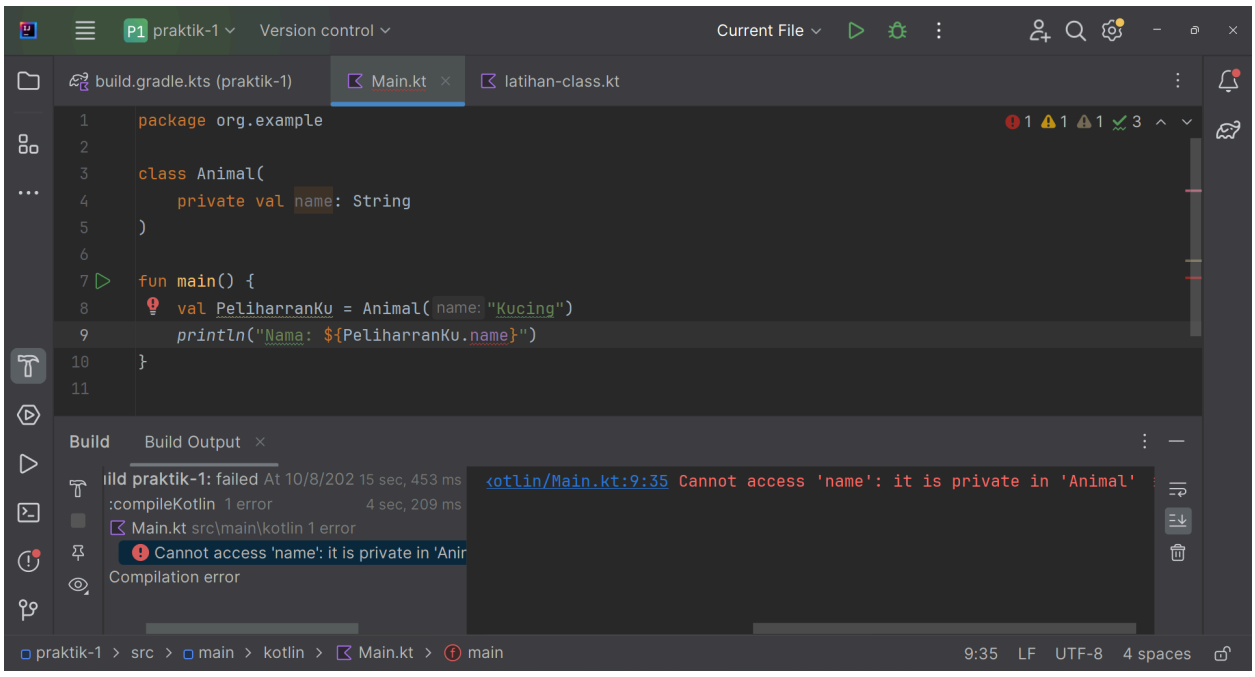
```
private val name: String,
```

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.

CONTOH LATIHAN:

STUDI KASUS - PENGATURAN SUHU AC:

Budi baru saja memasang AC di kamarnya dan ingin mengatur suhu ruangan sesuai kebutuhan. Namun, suhu AC hanya bisa diatur dalam rentang 16 hingga 30 derajat Celcius. Setiap kali Budi mengubah suhu, ia ingin memastikan bahwa suhu tersebut tidak melampaui batas yang diperbolehkan.

Instruksi: Buatlah kelas Air Conditioner dengan properti temperature (private). Buat metode untuk menaikkan dan menurunkan suhu dengan validasi agar suhu hanya berada di antara 16 hingga 30 derajat Celsius. Buat juga metode untuk melihat suhu saat ini.

Hint Kode (Jangan diubah ya!)

```
class AirConditioner(private var temperature: Int) {
    // ...
}

fun main() {
    val ac = AirConditioner(25)  // Suhu awal 25 derajat

    // Tampilkan suhu awal
    // ...

    // Naikkan suhu sebesar 5 derajat
    // ...
```

```
    // Tampilkan suhu setelah naik
    // ...

    // Turunkan suhu sebesar 10 derajat
    // ...

    // Tampilkan suhu setelah turun
    // ...

    // Coba naikkan suhu melewati batas maksimum
    // ...

    // Tampilkan suhu akhir
    // ...
}
```

JAWABAN:

```
class AirConditioner(private var temperature: Int) {

    // Fungsi untuk mendapatkan suhu saat ini
    fun getTemperature(): Int {
        return temperature
    }

    // Fungsi untuk menaikkan suhu dengan validasi
    fun increaseTemperature(amount: Int) {
        if (temperature + amount <= 30) {
            temperature += amount
        } else {
            println("Suhu tidak boleh lebih dari 30 derajat
Celsius.")
        }
    }

    // Fungsi untuk menurunkan suhu dengan validasi
    fun decreaseTemperature(amount: Int) {
        if (temperature - amount >= 16) {
            temperature -= amount
        } else {
            println("Suhu tidak boleh kurang dari 16 derajat
Celsius.")
        }
    }
}
```

```
fun main() {
    val ac = AirConditioner(25)  // Suhu awal 25 derajat

    // Tampilkan suhu awal
    println("Suhu awal: ${ac.getTemperature()} derajat Celsius")

    // Naikkan suhu sebesar 5 derajat
    ac.increaseTemperature(5)

    // Tampilkan suhu setelah naik
    println("Suhu setelah naik: ${ac.getTemperature()} derajat
Celsius")

    // Turunkan suhu sebesar 10 derajat
    ac.decreaseTemperature(10)

    // Tampilkan suhu setelah turun
    println("Suhu setelah turun: ${ac.getTemperature()} derajat
Celsius")

    // Coba naikkan suhu melewati batas maksimum
    ac.increaseTemperature(10)

    // Tampilkan suhu akhir
    println("Suhu akhir: ${ac.getTemperature()} derajat Celsius")
}
```

PENJELASAN KODE:

Kode ini merupakan implementasi sederhana dari enkapsulasi dalam pemrograman, di mana akses ke properti temperature dibatasi. *Class* Air Conditioner memungkinkan pengguna untuk mengatur suhu dengan batasan tertentu (16-30 derajat Celcius), serta memvalidasi setiap perubahan suhu untuk memastikan nilai yang wajar.

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. Jika kalian belum memahami instruksi yang diberikan di dalam **SOAL**, mintalah penjelasan dari bapak/ibu guru.
6. **SOAL BISA DILIHAT DI HALAMAN SELANJUTNYA.**

STUDI KASUS - 1:

Soal: Mobil Balap Jack

Jack adalah seorang pembalap yang ingin menguji mobil balap barunya. Dia ingin memastikan bahwa mobilnya tidak melaju lebih dari 200 km/jam demi alasan keamanan. Saat ini, mobil tersebut berjalan dengan kecepatan 50 km/jam, dan Jack berencana untuk menambah kecepatan secara bertahap selama uji coba. Bantulah Jack mengatur kecepatan mobilnya agar tidak melebihi batas yang sudah ditentukan.

Instruksi: Buat kelas Car dengan properti brand (public) dan speed (private). Buat metode untuk mendapatkan dan menambah kecepatan dengan validasi agar kecepatan tidak boleh melebihi 200 km/jam.

Hint Kode (Jangan diubah ya!)

```
class Car(val brand: String, private var speed: Int) {

    // Fungsi untuk mendapatkan nilai speed
    ???

    // Fungsi untuk menambah kecepatan
    ???
}

fun main() {
    val car = Car("Toyota", 50)

    // Jack memulai uji coba mobil balap dengan menampilkan
kecepatan awal

    // Jack menambah kecepatan mobil sebanyak 100 km/h.
    // Pastikan bahwa kecepatan mobil tidak melebihi batas maksimum
yang aman.

    // Tampilkan kecepatan mobil setelah penambahan
}
```

STUDI KASUS - 2:

Soal: Nilai GPA (IPK) Mahasiswa

Pak Dedi adalah seorang dosen di Universitas Brawijaya. Dia sedang mengelola data mahasiswa, salah satunya adalah nilai GPA (Grade Point Average). GPA mahasiswa harus berada dalam rentang 0.0 hingga 4.0, dan Pak Dedi ingin memastikan tidak ada kesalahan dalam input data. Ia meminta bantuan untuk memeriksa dan memperbarui GPA mahasiswanya dengan tepat.

Instruksi: Buat kelas Student dengan properti name (public) dan gpa (private). Buat metode untuk mengatur dan mendapatkan nilai gpa, serta validasi agar gpa hanya bisa diubah jika nilainya antara 0.0 hingga 4.0.

Hint Kode (Jangan diubah ya!)

```
class Student(val name: String, private var gpa: Double) {

    // Fungsi untuk mendapatkan nilai gpa
    ???

    // Fungsi untuk mengatur nilai gpa dengan validasi
    ???
}

fun main() {
    val student = Student("Alice", 3.5)

    // Pak Dedi ingin melihat GPA Alice saat ini

    // Pak Dedi mencoba memasukkan nilai GPA yang tidak valid
(contoh: 5.0)

    // Lihat apakah nilai GPA berubah setelah input tidak valid

    // Pak Dedi memperbarui GPA Alice ke nilai yang valid (contoh:
3.9)

    // Tampilkan GPA setelah perubahan
}
```

STUDI KASUS - 3:

Soal: Tabungan Arman

Arman adalah seorang siswa yang sedang menabung untuk membeli sepeda. Dia memiliki saldo awal sebesar Rp500.000 di celengannya. Arman hanya bisa menambah uang ke dalam celengannya, tetapi tidak bisa mengambil uang secara langsung dari sana. Setiap kali dia menambah uang, dia ingin mengetahui jumlah total uang yang ada di celengannya.

Instruksi: Buatlah kelas Savings dengan properti balance (private). Buat metode untuk menambahkan uang ke dalam celengan, dan metode untuk melihat jumlah saldo saat ini.

Hint Kode (Jangan diubah ya!)

```
class Savings(private var balance: Int) {
    // Fungsi untuk menambahkan uang
    fun addMoney(amount: Int) {
        // ...
    }
    // Fungsi untuk melihat saldo saat ini
    fun getBalance(): Int {
        // ...
    }
}
fun main() {
    val armanSavings = Savings(500000)  // Saldo awal Rp500.000
    // Arman menambah uang Rp100.000
    // ...

    // Tampilkan saldo saat ini
    // ...

    // Arman menambah uang Rp50.000
    // ...

    // Tampilkan saldo akhir
    // ...
}
```

STUDI KASUS - 4:

Soal: Volume Suara Ponsel

Tini baru saja membeli ponsel baru dan ingin mengatur volume suara dengan baik. Namun, volume suara pada ponsel tersebut hanya bisa diatur pada rentang 0 hingga 100. Tini ingin memastikan bahwa volume tidak pernah lebih tinggi dari 100 atau lebih rendah dari 0 saat dia menaikkan atau menurunkan volume.

Instruksi: Buatlah kelas Phone dengan properti volume (private). Buat metode untuk menaikkan dan menurunkan volume, dengan validasi agar volume hanya berada di antara 0 hingga 100.

Hint Kode (Jangan diubah ya!)

```
class Phone(private var volume: Int) {
    // Fungsi untuk menaikkan volume
    fun increaseVolume(amount: Int) {
        // ...
    }

    // Fungsi untuk menurunkan volume
    fun decreaseVolume(amount: Int) {
        // ...
    }

    // Fungsi untuk mendapatkan volume saat ini
    fun getVolume(): Int {
        // ...
    }
}

fun main() {
    val myPhone = Phone(50)  // Volume awal 50

    // Tampilkan volume awal
    // ...

    // Naikkan volume sebesar 30
    // ...

    // Tampilkan volume setelah naik
    // ...

    // Turunkan volume sebesar 70
    // ...
```

```
    // Tampilkan volume setelah turun
    // ...

    // Coba naikkan volume melewati batas maksimum
    // ...

    // Tampilkan volume akhir
    // ...
}
```

STUDI KASUS - 5:

Soal: Penerbitan Buku Baru

Sebuah penerbit sedang mencetak buku baru dengan judul "Pemrograman Kotlin". Mereka ingin memastikan bahwa jumlah halaman buku tersebut lebih dari 0 sebelum buku dicetak. Jika ada kesalahan dalam memasukkan jumlah halaman, penerbit harus bisa mengoreksinya sebelum proses produksi dimulai.

Instruksi: Buat kelas Book dengan properti title (public) dan pages (private). Buat metode untuk mendapatkan dan mengubah jumlah halaman dengan validasi agar nilai pages hanya bisa diubah jika lebih dari 0.

Hint Kode (Jangan diubah ya!)

```
class Book(val title: String, private var pages: Int) {

    // Fungsi untuk mendapatkan jumlah halaman
    ???

    // Fungsi untuk mengatur jumlah halaman dengan validasi
    ???
}

fun main() {
    val book = Book("Pemrograman Kotlin", 300)

    // Penerbit ingin melihat jumlah halaman awal

    // Penerbit mencoba mengubah jumlah halaman menjadi nilai yang
tidak valid (contoh: -10)

    // Tampilkan jumlah halaman setelah perubahan tidak valid

    // Penerbit memperbarui jumlah halaman menjadi nilai yang valid
(contoh: 350)

    // Tampilkan jumlah halaman setelah perubahan valid
}
```

STUDI KASUS - 6:

Soal: Produk di Toko Online

Sebuah toko online sedang menjual produk berupa laptop dengan harga Rp15.000.000. Pemilik toko ingin memastikan bahwa harga produk tidak boleh diubah menjadi nilai negatif. Setiap kali harga produk diubah, pemilik toko ingin melihat harganya secara real-time. Anda diminta untuk membantu pemilik toko menjaga integritas data harga produk.

Instruksi: Buat kelas Product dengan properti name (public) dan price (private). Buat metode untuk mendapatkan dan mengubah harga produk dengan validasi agar harga tidak boleh negatif.

Hint Kode (Jangan diubah ya!)

```
class Product(val name: String, private var price: Double) {

    // Fungsi untuk mendapatkan harga produk
    ???

    // Fungsi untuk mengatur harga dengan validasi
    ???
}

fun main() {
    val product = Product("Laptop", 15000000.0)

    // Pemilik toko ingin melihat harga produk saat ini

    // Pemilik toko mencoba mengubah harga menjadi nilai yang tidak
valid (contoh: -500000.0)

    // Tampilkan harga produk setelah perubahan tidak valid

    // Pemilik toko memperbarui harga produk menjadi nilai yang
valid (contoh: 12000000.0)

    // Tampilkan harga produk setelah perubahan valid
}
```