Lembar Kerja Peserta Didik  
Modul 4 - OOP Lanjutan

short line

# PETUNJUK PENGERJAAN

1. LKPD ini dikerjakan secara berkelompok.
2. Kelompok menggunakan susunan anggota kelompok yang sudah ditetapkan di awal.
3. Sebelum kalian mengerjakan LKPD ini secara berkelompok, pastikan anda sudah bergabung di GitHub
4. LKPD diisi sesuai dengan perintah yang telah dijelaskan pada setiap soalnya.
5. Jika kalian belum memahami instruksi yang diberikan di dalam LKPD, mintalah penjelasan dari bapak/ibu guru.
6. Setelah selesai mengerjakan soal, persiapkan diri kalian untuk melakukan presentasi penjelasan kode.
7. **DIPERBOLEHKAN MENGGUNAKAN CHATGPT SELAMA PROSES PENGERJAAN.**

# POIN TAMBAHAN

1. Apabila kalian mengerjakan tutorial *visibility modifiers* yang diberikan pada **MODUL 4.**
2. Apabila kalian mengerjakan dan mengumpulkan tepat waktu **(WAKTU PENGERJAAN 30 MENIT)**.

# PETUNJUK PENGUMPULAN

1. Pengumpulan tugas dilakukan perwakilan oleh ketua kelompok.
2. File yang dikumpulkan cukup dokumen ini saja dengan format .pdf.
   1. Cara mendownload dokumen dengan format pdf:
      1. Buka opsi file yang ada di pojok kiri atas layar.
      2. Kemudian pilih opsi download.
      3. Selanjutnya pilih tipe file PDF Documents.
3. Lalu kumpulkan file pada GitHub yang telah disediakan.

# REFERENSI TAMBAHAN UNTUK BELAJAR

1. Kalian bisa melihat video dari YouTube berikut apabila kalian masih belum paham sepenuhnya mengenai materi yang dipelajari sebelumnya:
2. Penjelasan *visibility modifiers* → [Belajar Kotlin OOP - 27 Visibility Modifier](https://youtu.be/AgQkSWoH7mA?si=vr1SGys40K-r6UZ2)
3. Tutorial dasar *visibility modifiers*  → [Tutorial Kotlin Dasar - 18. Visibility Modifiers](https://youtu.be/sQg40ppMhzc?si=xpWMjBn8fLrt6vl2)

# SOAL

Nama Kelompok : **Bebas**

Nama/No.Absen Anggota Kelompok :

1. Adhe Bagus W.S / 04 (AERONSUKI)
2. Billie Agung R. / 19 (IWAK CAKALANG)
3. Fakhri Tajusa F. / 32 (FSXA)
4. …
5. dst..

**TUTORIAL 1 - TUTORIAL PENGGUNAAN VISIBILITY MODIFIERS PUBLIC PADA KOTLIN**

1. Lampirkan gambar *screenshot* kode kalian dan sertakan notepad yang berisi nama anggota kelompok kalian!

| **Notes:** Gambar hanya contoh, jadi bisa kalian hapus ya. |
| --- |

1. Lampirkan kode yang kalian buat.

| class Animal(  var name: String, // Mutable property (can be changed)  var age: Int, // Mutable property (can be changed)  var weight: Double // Mutable property (can be changed)  ) {  var isMammal: Boolean = true  }  fun main() {  // Membuat objek Animal  val elephant = Animal("Elephant", 10, 500.0)  // Mengakses dan memodifikasi properti publik  println("Animal Name: ${elephant.name}") // Output: Elephant  elephant.name = "Big Elephant" // Modifying the name  println("Updated Name: ${elephant.name}") // Output: Big Elephant  }  }    }  **Notes:** Kode hanya contoh, jadi bisa kalian hapus ya. |
| --- |

1. Selanjutnya jelaskan secara singkat cara kerja kode kalian!

| 1. \*\*Kelas `Animal`:\*\*  - Kelas ini memiliki tiga properti utama:  - `name` (String) untuk menyimpan nama hewan.  - `age` (Int) untuk menyimpan umur hewan.  - `weight` (Double) untuk menyimpan berat hewan.  - Ketiga properti ini dideklarasikan sebagai \*\*mutable\*\* (`var`), yang berarti nilainya bisa diubah setelah objek diciptakan.  - Ada juga properti tambahan `isMammal`, yang diset ke nilai default `true`. Ini berarti hewan tersebut diasumsikan sebagai mamalia.  2. \*\*Fungsi `main`:\*\*  - Sebuah objek `Animal` bernama `elephant` diciptakan dengan nama "Elephant", umur 10, dan berat 500.0.  - Nilai properti `name` diakses dan ditampilkan menggunakan `println`, yang akan mencetak "Animal Name: Elephant".  - Nilai properti `name` kemudian diubah menjadi "Big Elephant", dan perubahan ini ditampilkan kembali dengan `println`, menghasilkan output "Updated Name: Big Elephant".  Secara keseluruhan, kode ini mendemonstrasikan cara membuat kelas dengan properti yang dapat diubah, menciptakan objek, dan mengakses serta memodifikasi propertinya.  **Notes:** Penjelasan hanya contoh, jadi bisa kalian hapus ya. |
| --- |

**TUTORIAL 2 - TUTORIAL PENGGUNAAN VISIBILITY MODIFIERS PRIVATE PADA KOTLIN**

1. Lampirkan gambar screenshot kode kalian dan sertakan notepad yang berisi nama anggota kelompok kalian!

|  |
| --- |

1. Lampirkan kode yang kalian buat.

| class Animal(private var name: String, private var age: Int) {  // Getter untuk name  fun getName(): String {  return name  }  // Setter untuk name  fun setName(newName: String) {  name = newName  }  // Getter untuk age  fun getAge(): Int {  return age  }  // Setter untuk age  fun setAge(newAge: Int) {  age = newAge  }  }  fun main() {  val myPet = Animal("Kucing", 2)  // Menggunakan getter untuk mendapatkan nilai 'name'  println("Nama hewan: ${myPet.getName()}") // Output: Kucing  // Menggunakan setter untuk mengubah nilai 'name'  myPet.setName("Banteng")  println("Nama hewan setelah diubah: ${myPet.getName()}") // Output: Banteng  // Menggunakan getter untuk mendapatkan nilai 'age'  println("Umur hewan: ${myPet.getAge()}") // Output: 2  // Menggunakan setter untuk mengubah nilai 'age'  myPet.setAge(3)  println("Umur hewan setelah diubah: ${myPet.getAge()}") // Output: 3  } |
| --- |

1. Selanjutnya jelaskan secara singkat cara kerja kode kalian!=

| Kode ini mendefinisikan kelas `Animal` dengan dua properti privat, `name` dan `age`. Karena properti tersebut bersifat privat, kita tidak bisa mengaksesnya langsung dari luar kelas, sehingga digunakan \*\*getter\*\* dan \*\*setter\*\* untuk mengakses dan memodifikasi nilainya.  Penjelasan singkat:  1. \*\*Kelas `Animal`:\*\*  - \*\*Getter\*\* dan \*\*setter\*\* dibuat untuk properti `name` dan `age`. Getter berfungsi untuk mengembalikan nilai properti, sedangkan setter digunakan untuk mengubah nilainya.  2. \*\*Fungsi `main`:\*\*  - Objek `myPet` dibuat dengan nama "Kucing" dan umur 2.  - Menggunakan getter untuk menampilkan nilai `name` dan `age`.  - Setter digunakan untuk mengubah `name` menjadi "Banteng" dan `age` menjadi 3, lalu nilai yang diubah ditampilkan kembali menggunakan getter.  Jadi, kode ini mendemonstrasikan cara menggunakan getter dan setter untuk mengelola akses ke properti privat dalam sebuah kelas di Kotlin. |
| --- |

**Studi kasus berapa kamu? - Nama studi kasusnya apa?**

1. Lampirkan gambar screenshot kode kalian dan sertakan notepad yang berisi nama anggota kelompok kalian!

|  |
| --- |

1. Lampirkan kode yang kalian buat.

| class Car(val brand: String, private var speed: Int) {  private var engineOn: Boolean = false  // Fungsi untuk mendapatkan nilai speed  fun getSpeed(): Int {  return speed  }  // Fungsi untuk menambah kecepatan  fun increaseSpeed(increment: Int) {  if (!engineOn) {  println("Mesin masih mati. Nyalakan mesin terlebih dahulu.")  return  }  val maxSpeed = 200  if (speed + increment > maxSpeed) {  speed = maxSpeed  println("Kecepatan mencapai batas maksimum: $speed km/h")  } else {  speed += increment  println("Kecepatan meningkat menjadi: $speed km/h")  }  showSpeedBar()  }  // Fungsi untuk mengurangi kecepatan  fun decreaseSpeed(decrement: Int) {  if (!engineOn) {  println("Mesin masih mati. Nyalakan mesin terlebih dahulu.")  return  }  if (speed - decrement < 0) {  speed = 0  println("Mobil berhenti total: $speed km/h")  } else {  speed -= decrement  println("Kecepatan berkurang menjadi: $speed km/h")  }  showSpeedBar()  }  // Fungsi untuk menyalakan mesin  fun startEngine() {  if (!engineOn) {  engineOn = true  println("Mesin dinyalakan. Siap untuk jalan!")  } else {  println("Mesin sudah menyala.")  }  }  // Fungsi untuk mematikan mesin  fun stopEngine() {  if (engineOn) {  engineOn = false  println("Mesin dimatikan.")  } else {  println("Mesin sudah mati.")  }  }  // Fungsi untuk menampilkan simulasi bar kecepatan  private fun showSpeedBar() {  val barLength = (speed / 10)  val bar = "#".repeat(barLength)  println("Speed Bar: [$bar] $speed km/h")  }  }  fun main() {  val car = Car("Toyota", 50)  // Jack memulai uji coba mobil balap dengan menampilkan kecepatan awal  println("Jack memulai uji coba mobil ${car.brand}")    // Nyalakan mesin  car.startEngine()  // Tampilkan kecepatan awal  println("Kecepatan awal: ${car.getSpeed()} km/h")    // Jack menambah kecepatan mobil sebanyak 100 km/h  car.increaseSpeed(100)  // Jack mencoba mengerem sebanyak 60 km/h  car.decreaseSpeed(60)  // Jack menambah kecepatan lagi sebanyak 80 km/h  car.increaseSpeed(80)  // Jack berhenti dan mematikan mesin  car.decreaseSpeed(70)  car.stopEngine()  } |
| --- |

1. Selanjutnya jelaskan secara singkat cara kerja kode kalian!

| Kode yang kamu berikan adalah sebuah simulasi sederhana untuk menguji kecepatan mobil dengan berbagai fitur tambahan seperti menambah dan mengurangi kecepatan, menyalakan/mematikan mesin, serta animasi visual kecepatan. Berikut penjelasan mengenai cara kerja kode tersebut:  ### 1. \*\*Kelas `Car`\*\*  Kelas ini mewakili sebuah objek mobil dengan properti dan fungsi-fungsi yang mengatur perilakunya:  - \*\*Properti:\*\*  - `brand` (public): Merek mobil. Properti ini bersifat terbuka untuk diakses dari luar kelas.  - `speed` (private): Kecepatan mobil, hanya bisa diakses dan dimodifikasi dari dalam kelas.  - `engineOn` (private): Status mesin mobil (hidup atau mati).  - \*\*Fungsi-fungsi:\*\*  1. \*\*`getSpeed()`\*\*: Mengembalikan nilai kecepatan mobil saat ini.  2. \*\*`increaseSpeed(increment: Int)`\*\*: Menambah kecepatan mobil berdasarkan nilai `increment`, tapi hanya jika mesin sudah menyala (`engineOn == true`). Fungsi ini juga memastikan bahwa kecepatan tidak boleh melebihi batas maksimum 200 km/h.  3. \*\*`decreaseSpeed(decrement: Int)`\*\*: Mengurangi kecepatan mobil. Jika mesin belum menyala, maka fungsi tidak akan bekerja. Kecepatan tidak boleh turun di bawah 0 km/h, dan jika penurunan kecepatan melebihi angka ini, akan otomatis di-set ke 0.  4. \*\*`startEngine()`\*\*: Menyalakan mesin mobil. Jika mesin sudah menyala, akan ditampilkan pesan bahwa mesin sudah menyala.  5. \*\*`stopEngine()`\*\*: Mematikan mesin mobil. Jika mesin sudah mati, maka pesan akan menampilkan bahwa mesin sudah mati.  6. \*\*`showSpeedBar()`\*\*: Fungsi untuk menampilkan animasi bar sederhana yang menggambarkan kecepatan saat ini. Semakin cepat mobil, semakin panjang bar yang ditampilkan.  ### 2. \*\*Fungsi `main()`\*\*  Fungsi utama (`main()`) yang dijalankan untuk melakukan simulasi:  - \*\*Membuat Objek `Car`\*\*: Mobil baru dengan merek "Toyota" dan kecepatan awal 50 km/h dibuat.  - \*\*Menyalakan Mesin\*\*: Sebelum Jack bisa menguji mobil, mesin harus dinyalakan menggunakan `startEngine()`. Jika mesin belum menyala, maka kecepatan tidak bisa diubah.  - \*\*Menampilkan Kecepatan Awal\*\*: Kecepatan awal mobil ditampilkan.  - \*\*Menambah Kecepatan\*\*: Jack menambah kecepatan mobil sebanyak 100 km/h dengan fungsi `increaseSpeed()`. Fungsi ini akan menampilkan pesan dan memastikan bahwa kecepatan tidak lebih dari 200 km/h.  - \*\*Mengerem (decreaseSpeed)\*\*: Jack mencoba menurunkan kecepatan sebanyak 60 km/h. Fungsi `decreaseSpeed()` akan mengurangi kecepatan, tetapi memastikan bahwa kecepatan tidak turun di bawah 0 km/h.  - \*\*Menambah Kecepatan Lagi\*\*: Jack kembali menambah kecepatan sebanyak 80 km/h. Sama seperti sebelumnya, fungsi ini memeriksa apakah kecepatan tetap dalam batas yang aman.  - \*\*Berhenti dan Mematikan Mesin\*\*: Jack akhirnya mengurangi kecepatan mobil hingga berhenti, lalu mematikan mesin menggunakan `stopEngine()`.  ### 3. \*\*Logika Keseluruhan\*\*  - Mobil \*\*tidak bisa bergerak\*\* atau mengubah kecepatan jika mesinnya belum dinyalakan. Semua fungsi untuk mengatur kecepatan hanya akan bekerja jika status mesin (`engineOn`) sudah `true`.  - Fungsi `increaseSpeed()` dan `decreaseSpeed()` mengelola perubahan kecepatan, namun memastikan \*\*batas atas kecepatan maksimal\*\* 200 km/h dan \*\*batas bawah\*\* 0 km/h.  - \*\*Animasi kecepatan\*\*: Setiap kali kecepatan berubah, akan ada bar visual (`showSpeedBar()`) yang menunjukkan seberapa cepat mobil melaju berdasarkan panjang bar. Semakin tinggi kecepatannya, semakin panjang simbol `#` yang muncul.  ### Contoh Alur Eksekusi:  1. Mobil dimulai dengan kecepatan 50 km/h, tetapi mesinnya mati.  2. Jack menyalakan mesin menggunakan `startEngine()`.  3. Jack menambah kecepatan sebesar 100 km/h, sehingga mobil mencapai 150 km/h. Bar kecepatan ditampilkan untuk merepresentasikan kecepatan ini.  4. Jack kemudian mengerem dan menurunkan kecepatan sebesar 60 km/h, mengurangi kecepatan menjadi 90 km/h.  5. Jack mencoba menambah kecepatan lagi, tapi jika melebihi batas 200 km/h, maka akan diatur ke batas maksimum tersebut.  6. Setelah selesai, Jack mengurangi kecepatan sampai mobil berhenti, dan kemudian mematikan mesin.  Kode ini menciptakan simulasi yang lebih interaktif dan memperhatikan keamanan kecepatan dengan batas maksimum dan minimum. |
| --- |