VERSION 1.6

10 Maret 2025



# PEMROGRAMAN BERORIENTASI OBJEK

MODUL 3 – CONSTRUCTOR, ENCAPSULATION, INHERITANCE,

OVERRIDING, SUPER KEYWORDS

DISUSUN OLEH:

WIRA YUDHA AJI PRATAMA

KEN ARYO BIMANTORO

DIAUDIT OLEH:

Ir. Galih Wasis Wicaksono, S.Kom, M.Cs.

PRESENTED BY: TIM LAB. IT

UNIVERSITAS MUHAMMADIYAH MALANG

**PENDAHULUAN**



## TUJUAN

1. Mahasiswa memahami konsep constructor, enkapsulasi, inheritance, overriding, dan penggunaan super keywords dalam Java.
2. Mahasiswa memahami bagaimana konsep-konsep tersebut digunakan untuk meningkatkan modularitas dan efisiensi dalam pemrograman berorientasi objek.
3. Mahasiswa memahami cara mengimplementasikan konsep-konsep OOP ini dalam proyek Java yang lebih kompleks.

## TARGET MODUL

1. Mahasiswa dapat membuat program yang menerapkan constructor, enkapsulasi, inheritance, dan overriding dalam Java.
2. Mahasiswa dapat menggunakan super keywords untuk mengakses konstruktor atau method dari superclass.

## PERSIAPAN

1. Device (Laptop/Komputer)
2. IDE (IntelliJ)
3. Internet

***KEYWORDS***

Encapsulation, Inheritance, Overriding, Super Keyword

## TABLE OF CONTENTS

**PENDAHULUAN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1**

TUJUAN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_1

TARGET MODUL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1

PERSIAPAN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1

KEYWORDS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1

TABLE OF CONTENTS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1

**CONSTRUCTOR\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**

TEORI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

MATERI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3

Constructor dengan Parameter\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 5

PRAKTEK\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

TIPS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9 **ENCAPSULATION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10**

TEORI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10

MATERI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10

Public\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 11

Protected\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 12

Private\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 13

Method Setter dan Getter\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 14

PRAKTEK\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_17

TIPS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20

**INHERITANCE\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20**

TEORI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20

MATERI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 21

PRAKTEK\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_26

TIPS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_28

**OVERRIDING\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 29**

TEORI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 29

MATERI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 29

PRAKTEK\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_31

TIPS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_33

**SUPER KEYWORDS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 34**

TEORI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 34

MATERI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 34

TIPS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_37

**THIS KEYWORDS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 38**

TEORI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 38

MATERI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 38

TIPS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_39

**CODELAB & TUGAS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 40**

CODELAB\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 40

TUGAS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_41

**PENILAIAN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 42**

RUBRIK PENILAIAN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_42

SKALA PENILAIAN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_43

**SUMMARY AKHIR MODUL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_44**

**CONSTRUCTOR**



## TEORI



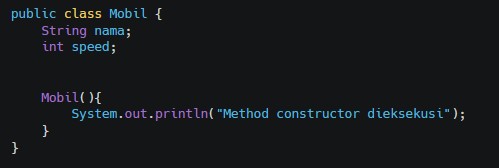
Constructor merupakan suatu method yang akan memberikan nilai awal pada saat suatu objek dibuat. Pada saat program dijalankan, constructor akan langsung memberikan nilai awal pada saat membuat suatu *instance* objek

## MATERI

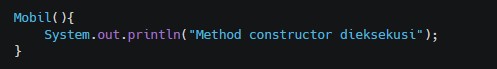
Pada saat kita bekerja dengan constructor, ada beberapa hal yang perlu diperhatikan, yaitu:

1. Nama constructor harus sama dengan nama class
2. Tidak ada return type yang diberikan ke dalam constructor signature.
3. Tidak ada return statement, di dalam tubuh constructor.

Contoh ketika kita membuat sebuah class Mobil dan menggunakan constructor:



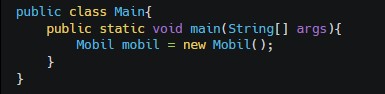
Pada contoh di atas constructor ditulis seperti ini:



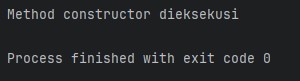
Mari kita coba untuk membuat objek baru dari class Mobil di Main class:



Sehingga sekarang kode pada Main class kita menjadi seperti ini:

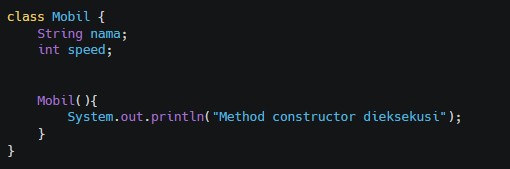


Ketika dijalankan akan menampilkan output berikut:

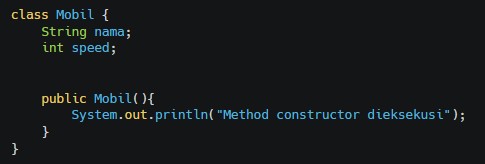


Method constructor akan dieksekusi ketika sebuah instance objek baru dibuat, bahkan tanpa memanggil method itu. Cara penulisan ketika membuat sebuah constructor class ada 2 cara, yaitu:

1. Menulis method dengan nama yang sama pada class



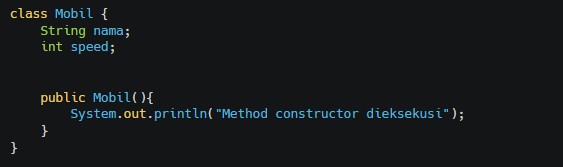
1. Menggunakan modifier public



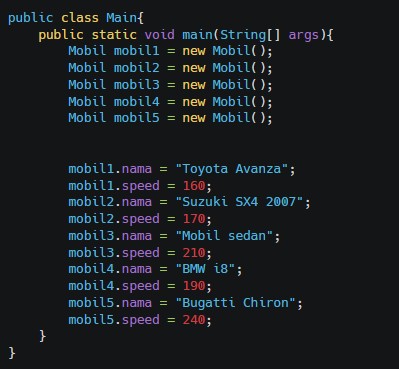
### Constructor dengan Parameter

Setelah mengetahui apa itu constructor, kita akan bertanya-tanya untuk apa fungsi dari constructor itu sendiri? Jawabannya ialah dengan sebuah contoh kasus, misalnya kita memiliki sebuah data mobil 5 buah dengan data setiap mobil memiliki nama dan speed yang berbeda-beda dan ingin membuat sebuah objek mobil satu-satu. Mungkin kode yang kita buat akan seperti ini:

○ pada class Mobil

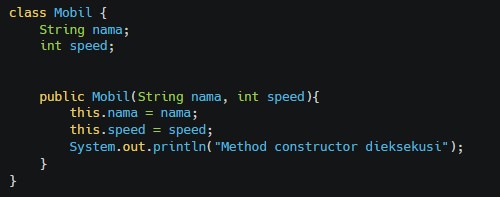


○ pada class Main

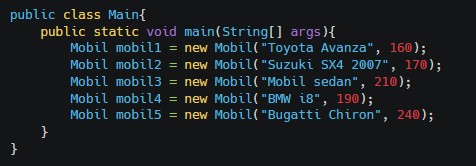


pada kode class Main terlihat bahwasannya kita harus melakukan input satu-satu dan memakan banyak tempat hanya untuk menginputkan sebuah data saja. Maka dari itu jika kita ingin menginisialisasi sebuah data secara langsung pada saat sebuah instance class dibuat, maka bisa kita manfaatkan sebuah parameter dengan cara menambahkan sebuah parameter pada constructor class Mobil dan menggunakan argumen ketika membuat instance objek class Mobil. Kodenya akan tampak seperti ini.

○ pada class Mobil



○ pada class Main



Pada kode class Mobil di atas, kita menambahkan parameter nama dan speed ke dalam constructor. Jadi ketika membuat sebuah instance objek dari class Mobil, kita harus menambahkan argumen seperti ini:



Sangat terlihat perbedaannya ketika kita tidak menggunakan constructor dan ketika menggunakan constructor, kode akan lebih simpel dan mudah dibaca ketika menggunakan

constructor.

Jika kita tidak membuat constructor pada sebuah class, maka secara otomatis dan tidak terlihat sebenarnya class itu sudah membuat constructor sendirinya tetapi body constructor yang kosong.

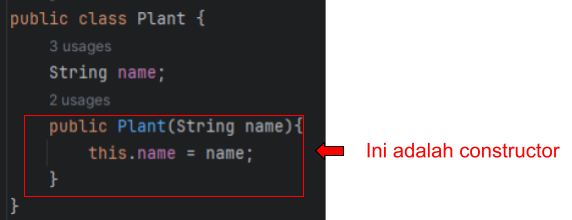
## PRAKTEK

Mari kita menerapkan teknik dari Constructor ini di project kita sebelumnya! Silahkan buka project pada praktek di modul sebelumnya. Masih ada kan? Jika tidak, silahkan buat ulang 🙂



Kita dapat melakukan sedikit perubahan pada class Main untuk membuatnya menjadi lebih singkat dan efisien. Bagaimana caranya? Silahkan pergi ke Class Farmer dan Plant untuk membuat constructor yang akan kita gunakan di Class Main.





Untuk penjelasan ʻthisʼ keywords bisa dibaca disini. Dengan begitu, Class Main dapat kita buat jadi seperti ini:



Sangat keren bukan? Kalian juga bisa mengganti parameternya menggunakan Objek. Silahkan mencoba! Jika ada kesulitan kalian bisa bertanya ke asisten.

## TIPS

Materi tambahan tentang Constructor di Java:

[Video1](https://www.youtube.com/watch?v=WPmAu26LZKo&pp=ygUUY29uc3RydWN0b3IgaW4gamF2YXM%3D)

[VIdeo2](https://www.youtube.com/watch?v=23AOrSN-wmI&pp=ygUUY29uc3RydWN0b3IgaW4gamF2YXM%3D)

[Materi](https://www.w3schools.com/java/java_constructors.asp)

**ENCAPSULATION**



## TEORI



Encapsulation adalah pembungkus, encapsulation pada object oriented maksudnya adalah membungkus class dan menjaga apa apa saja yang ada di dalam class tersebut, baik method ataupun atribut, agar tidak dapat diakses oleh class lainnya. Untuk menjaga hal tersebut dalam Encapsulation dikenal nama Hak Akses Modifier yang terdiri dari Private. Public dan Protected. Tapi perlu diingat, modifier tidak hanya bisa diberikan kepada atribut dan method saja. Tapi juga bisa diberikan kepada interface, enum, dan class itu sendiri.

## MATERI

Fungsi dari access modifier pada Java adalah untuk membatasi scope dari sebuah class, constructor, variabel, method, atau anggota data lainnya yang terdapat dalam suatu program Java.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modifier | Class | Package | Subclass | World |
| public | True | True | True | True |
| protected | True | True | True | False |
| no modifier | True | True | False | False |
| private | True | False | False | False |

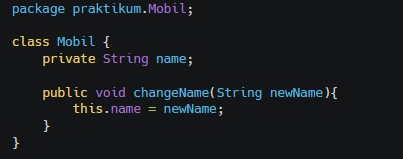
Perhatikan kode berikut:



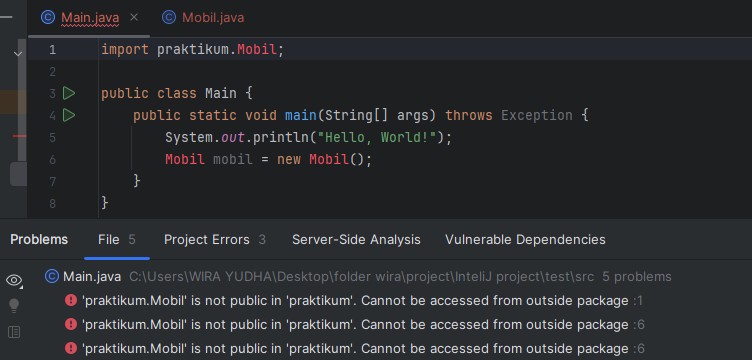
Kode yang ditandai di atas adalah modifier. Modifier akan digunakan untuk menentukan akses atribut, method, dan class.

### Public

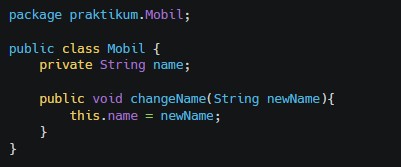
Memberikan hak akses kepada atribut atau method agar bisa diakses oleh siapapun (property atau class lain di luar class yang bersangkutan), artinya method atau atribut yang ada di class A dapat diakses oleh siapapun baik itu class A, class B dan seterusnya. Contoh:



Pada class Mobil di atas terdapat atribut name dan method changeName(). Kedua hal tersebut kita berikan sebuah modifier public, artinya keduanya akan bisa diakses dari mana saja. Namun, class Mobil tidak kita berikan sebuah modifier. Maka yang akan terjadi adalah class tersebut tidak akan bisa diimport atau diakses dari luar package.



Class Mobil berada di package com.praktikum, lalu kita coba untuk akses dari package com.main, maka akan terjadi error seperti gambar di atas. Cara untuk memperbaikinya adalah dengan menambahkan modifier public pada class Mobil. Maka kode pada class Mobil akan menjadi seperti ini:

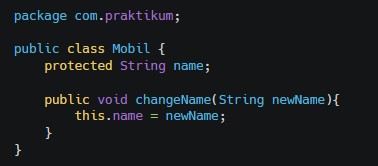


Maka error akan hilang dan kode sudah bisa dijalankan secara normal.

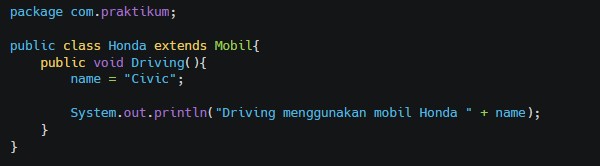
### Protected

Memberikan hak akses pada class itu sendiri dan class hasil turunannya (inheritance), artinya apa saja yang di class A hanya bisa diakses oleh class A sendiri dan class yang meng extends class A. Namun harus dipahami class lain yang berada dalam satu package dengan class A mampu mengakses tipe data protected, sedangkan yang tidak mampu mengakses adalah class - class yang berada diluar package class A. Untuk dapat mengaksesnya, class yang berada diluar package class A harus meng extends class A. Terkhususkan modifier protected hanya boleh digunakan pada atribut dan method saja, tidak bisa diberikan kepada class, enum, dan interface.

Contoh class Mobil memiliki turunan yaitu class Honda:



Pada kode class Mobil di atas kita memberikan modifier protected pada atribut name.



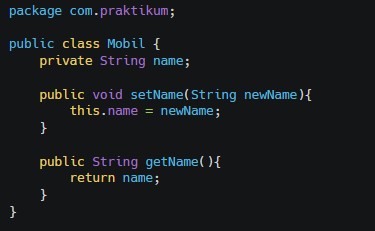
Maka ketika kita coba akses pada class Honda yang dimana merupakan class turunannya hal ini tidak akan terjadi error. Atau jika kita ingin mengakses dari class yang satu package dengan class Mobil, itu juga bisa meskipun bukan class turunan dari class Mobil.

Tetapi, apabila kita mencoba untuk mengakses atribut name dari package yang berbeda maka akan terjadi error karena atribut name sudah kita berikan modifier protected.



### Private

Memberikan hak akses hanya pada class itu sendiri, artinya apa-apa saja yang ada di dalam class A baik itu method apapun atribut hanya bisa diakses oleh class A saja, class lain tidak bisa mengaksesnya. Modifier private juga tidak bisa diberikan kepada class, enum, dan interface. Modifier private hanya diberikan kepada atribut dan method class saja. Contoh:



Pada contoh di atas, kita memberikan modifier private pada atribut name dan modifier public pada method setName() dan getName(). Apabila kita mencoba untuk langsung mengakses pada atribut name seperti ini:

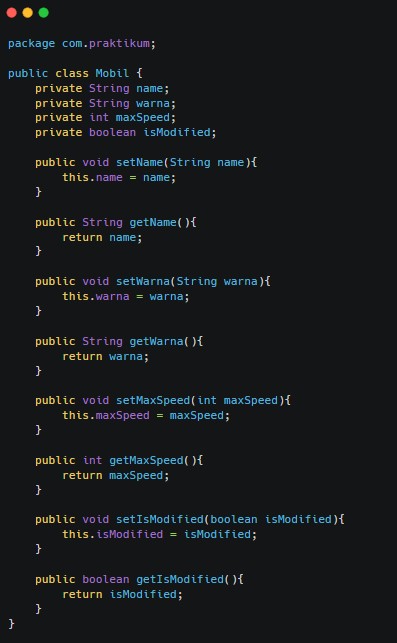


Lantas bagaimana cara untuk mengakses sebuah atribut atau method yang diberikan modifier private dari luar class? Jawabannya ialah bisa menggunakan method getter dan setter. Karena, method ini akan selalu diberikan modifier public. Method setter dan getter akan dipelajari setelah ini.

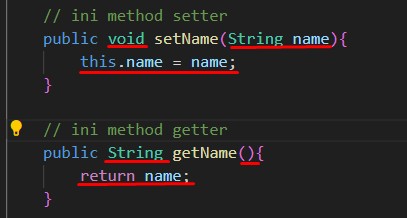
### Method Setter dan Getter

Setter adalah (chatgpt : mengubah/ menyetel nilai variabe) sebuah method saat kita memasukkan sebuah nilai/values ke dalam suatu variabel/atribut/object, sedangkan Getter adalah (chatgpt : mengambil/ mendapatkan nilai variabel) sebuah method saat kita mengambil sebuah nilai/values dari suatu variabel/atribut/object.

Cara untuk implementasi setter dan getter pada class Mobil, buat dahulu beberapa atribut dengan modifier private. Setelah itu buat sebuah method biasa, untuk setter harap diawali dengan kata set dan untuk getter harap diawali get agar mudah diketahui kode mana yang termasuk setter dan getter. Untuk modifier method setter dan getter selalu berikan modifier public, karena method ini yang akan diakses oleh class lain. Contohnya:



Perbedaan utama antara method setter dengan getter terletak pada nilai kembalian, parameter, dan isi method. Method setter tidak memiliki nilai kembalian void (kosong), karena tugasnya hanya untuk mengisi data ke dalam atribut. Sedangkan method getter memiliki nilai kembalian sesuai dengan tipe data yang akan diambil.

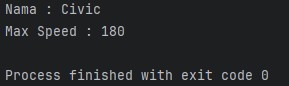


Terkadang kita memiliki pertanyaan, bolehkan untuk awalan menggunakan bahasa indonesia? misalnya isi untuk setter dan ambil untuk getter. Hal itu boleh-boleh saja, tetapi sangat tidak dianjurkan. Karena suatu saat nanti jika kita bekerja dengan tim, tim yang lain akan kesulitan untuk membaca. Apalagi tim tersebut lintas negara yang menggunakan bahasa inggris.

Setelah kita membuat method setter dan getter, kita bisa mengakses atau menggunakannya seperti method biasa. Contoh:



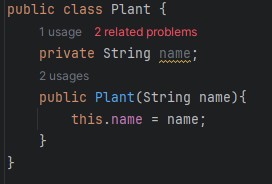
Hasil output:



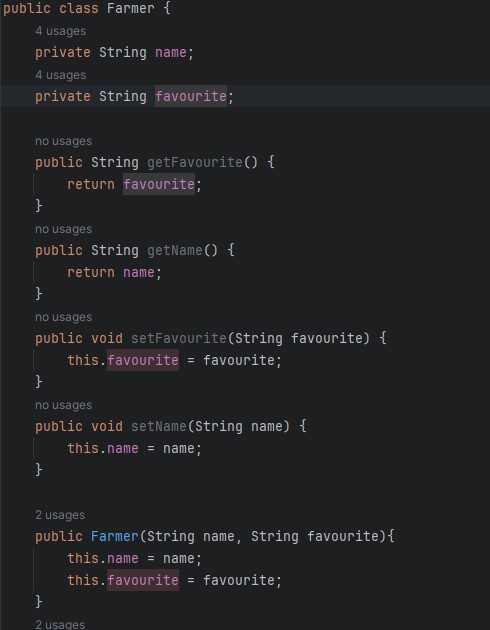
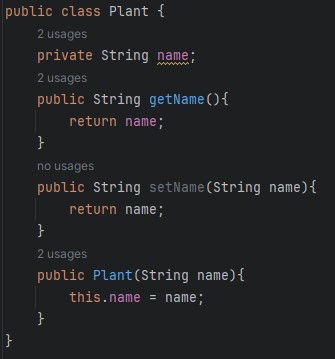
1. Beberapa alasan kenapa harus menggunakan Setter dan Getter:
2. Untuk meningkatkan keamanan data
3. Agar lebih mudah dalam mengontrol atribut dan method
4. Class bisa kita buat menjadi read-only dan write-only
5. Programmer dapat mengganti sebagian dari kode tanpa harus takut berdampak pada kode lain

## PRAKTEK

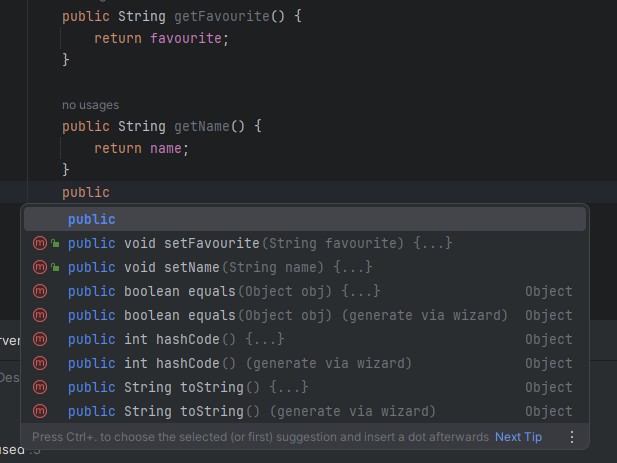
Mari kita menerapkan konsep Encapsulation ini di projek praktek kita! SIlahkan buka file projeknya. Lalu pergi ke Class Farmer dan Plant, lalu buat semua variabel menjadi private seperti ini:



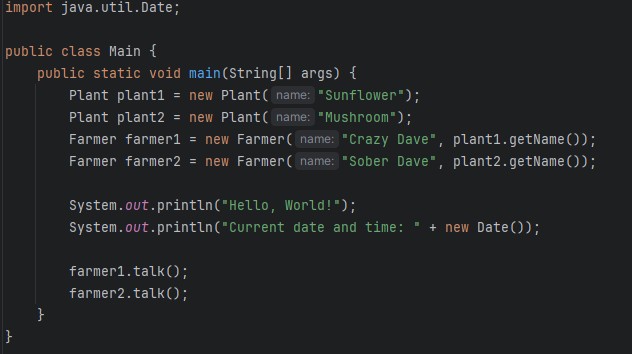
Wait, terdapat *error*. Kira-kira apa yang terjadi? Ah benar, pada Class main, kita mencoba untuk memanggil variabel ʻnameʼ di Class Plant yang dimana bersifat private. Lalu bagaimana cara mendapatkan variabel ʻnameʼ tersebut? Kita akan buat method Getter dan Setter sebagai berikut:



Kalian bisa memanfaatkan fitur auto-complete yang disediakan oleh IntelliJ kalau kalian males ngetik.



Kemudian pada Clas Main, kalian ubah constructor class Farmer jadi seperti ini:



## TIPS

Java encapsulation:

[Video](https://www.youtube.com/watch?v=eboNNUADeIc&pp=ygUbZW5jYXBzdWxhdGlvbiBqYXZhIGludGVsbGlq)

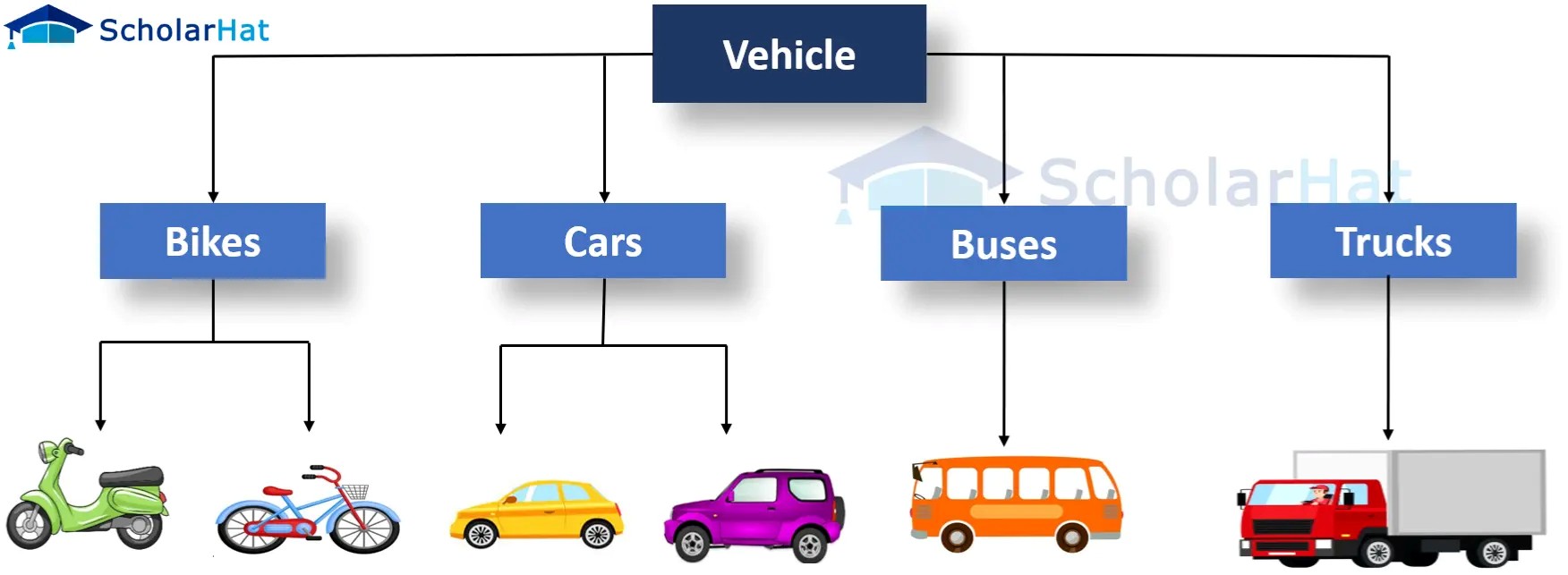
Dasar getter dan setter:

[Video](https://www.youtube.com/watch?v=xQdtVa79dXw&list=PLjRBWix725xqEddhpGsg8_VmrIGoaFhs6&index=5)

**INHERITANCE**



## TEORI



Pewarisan atau Inheritance adalah prinsip dalam pemrograman di mana sebuah class memiliki kemampuan untuk mewariskan properti dan metode yang dimilikinya kepada class lain. Konsep pewarisan digunakan untuk memanfaatkan fitur 'code reuse' guna menghindari duplikasi kode program.

Dengan adanya pewarisan, terbentuklah struktur atau hirarki class dalam kode program. Class yang memberikan warisan disebut sebagai class induk (parent class), super class, atau base class. Sementara class yang menerima warisan disebut sebagai class anak (child class), sub class, derived class, atau heir class.

Tidak semua properti dan metode dari class induk akan diwariskan. Properti dan metode dengan hak akses private tidak akan diwariskan kepada class anak. Hanya properti dan metode dengan hak akses protected dan public saja yang dapat diakses dari class anak.

Suatu class yang memiliki turunan disebut sebagai parent class atau base class. Sedangkan class turunan itu sendiri sering disebut sebagai subclass atau child class. Sebuah subclass dapat mewarisi semua yang dimiliki oleh parent class.

Karena sebuah subclass dapat mewarisi semua yang dimiliki oleh parent class-nya, maka member dari subclass terdiri dari apa yang dimilikinya sendiri dan juga apa yang diwarisi dari class parent-nya. Secara keseluruhan, dapat dikatakan bahwa suatu subclass hanya memperluas (extend) parent class-nya.

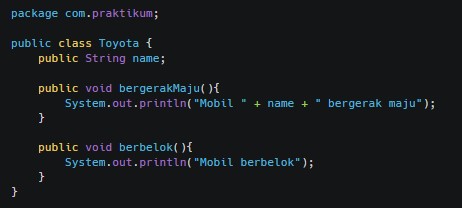
## MATERI

Alasan kenapa harus menggunakan inheritance, misalkan pada sebuah mobil, kita akan membuat class-class mobil yang lebih spesifik. Lalu kita membuat kode untuk masing-masing class seperti ini:

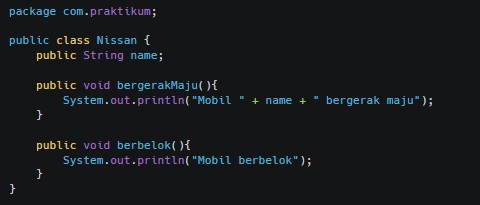
File: Honda.java



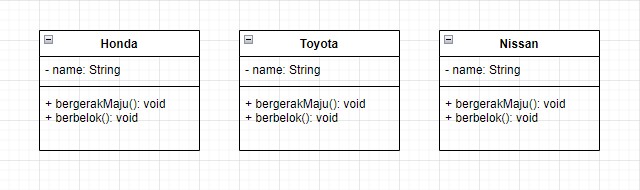
File: Toyota.java



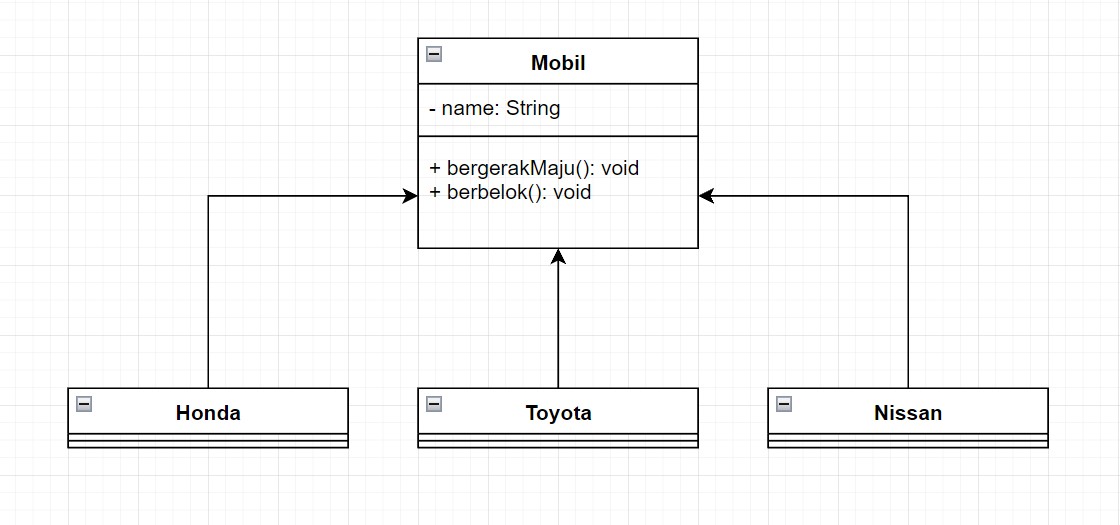
File: Nissan.java



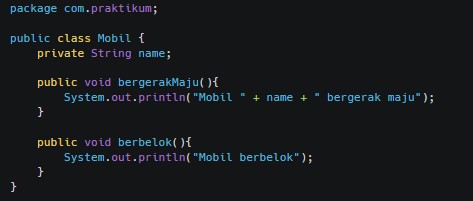
Dari ketiga class kode di atas, apakah boleh menulis kode seperti itu? Iya boleh-boleh saja. Tetapi hal itu tidak efektif, karena kita menulis kode yang sama berulang-ulang. Alih-alih kode berulang-ulang, solusi yang bisa digunakan ialah harus menggunakan inheritance. Mari kita lihat pada atribut dan method yang sama:



Pada diagram di atas terlihat bahwa atribut name, method bergerakMaju() dan berbelok() memiliki kesamaan pada class Honda, Toyota, dan Nissan. Setelah kita menggunakan inheritance dengan menyatukan atribut dan method yang memiliki cara kerja yang sama ke dalam satu class yaitu class Mobil, maka bentuk diagramnya akan menjadi seperti ini:

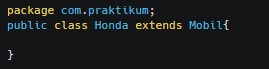


Class Mobil adalah class induk atau parent class yang memiliki class anak atau child class yaitu Honda, Toyota, dan Nissan. Apapun atribut yang ada di parent class akan dimiliki juga oleh class anak. Bentuk kode class Mobil seperti ini:

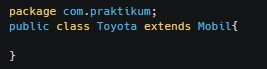


Pada child class, kita harus menggunakan kata kunci extends untuk menyatakan kalau class itu adalah child class dari class Mobil.

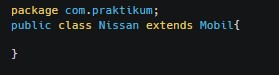
File: Honda.java



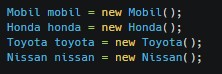
File: Toyota.java



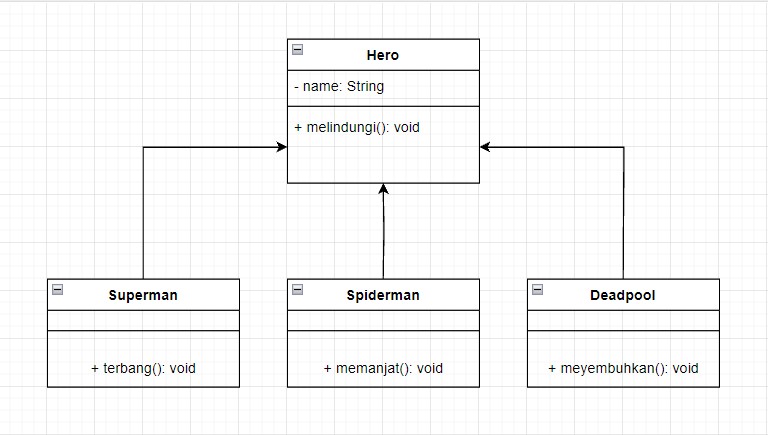
File: Nissan.java



Terlihat pada class Honda, Toyota, Nissan body class-nya kosong, pada child class kita bisa membiarkan isi body kosong atau diberi isi sesuai kebutuhan. Untuk cara membuat objek dari masing-masing class, kita bisa membuatnya seperti ini: File: Main.java

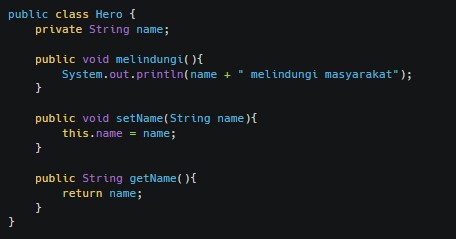


Contoh kasus lain ialah class Hero, misalnya class Hero yang memiliki child class berupa Superman, Spiderman, Deadpool. Ketiga child class memiliki kesamaan nama Hero, melindungi masyarakat. Tetapi masing-masing child class memiliki perbedaan yaitu untuk Superman bisa terbang, Spiderman bisa memanjat di tebing, Deadpool bisa penyembuhan diri. Diagram dari parent class dan child class bisa berbentuk seperti ini:

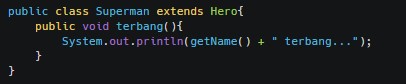


Untuk kode pada setiap class akan seperti ini:

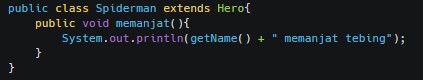
File: Hero.java



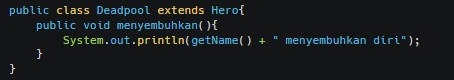
File: Superman.java



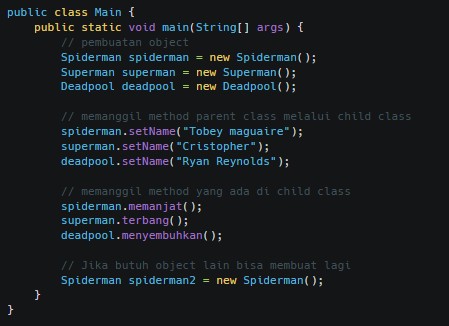
File: Spiderman.java



File: Deadpool.java



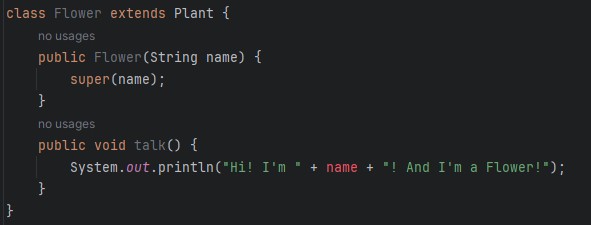
Untuk cara menggunakan pada file class Main.java seperti ini:

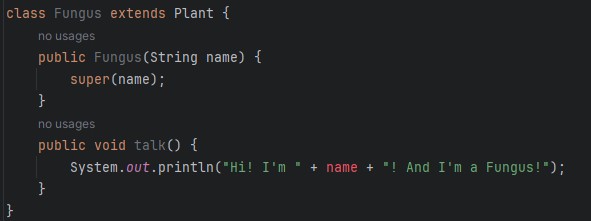


Pemanggilan method setName adalah milih parentclass yang dimiliki oleh class Hero, hal itu bisa dipanggil melalui child class karena masing-masing class Spiderman, Superman, Deadpool adalah turunan atau anak dari class Hero. Sedangkan method memanjat() hanya dimiliki oleh class Spiderman, maka class Superman dan Deadpool tidak bisa memanggil class memanjat() karena tidak memiliki class tersebut, berlaku juga untuk method terbang() di class Superman dan method menyembuhkan() di class Deadpool.

## PRAKTEK

Mari kita mempraktekan teknik Inheritance yang barusan kita pelajari pada program kita! Silahkan buat dua kelas baru dengan spesifikasi seperti berikut:





Wait, terdapat *error*. Kira-kira dimana yang salah?

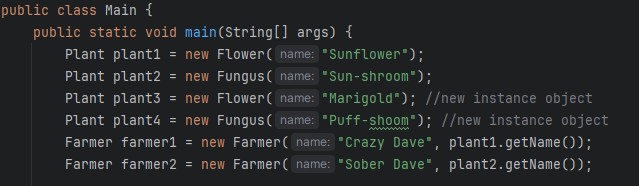
Tips: anda bisa melihat error description dengan menekan  di pojok kanan atas. Maka

akan terlihat dimana letak errornya:

Ah benar! Kita sudah merubah Attribute **“name”** di kelas **Plant** kemarin menjadi private. SIlahkan buat jadi public terlebih dahulu!

Apa itu keywords **“super”** pada constructor diatas? Kalian dapat mempelajarinya disini.

Setelah itu, kita dapat menambahkan 2 objek baru di kelas main. Saya buat seperti pada gambar dibawah. (Silahkan buat sesuai kekreativitasan kalian.)

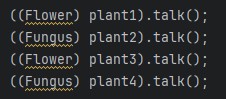


Pada gambar diatas, saya tambahkan 2 objek baru:

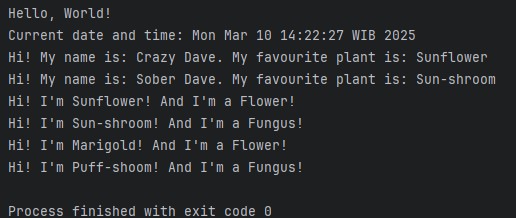
1. Objek dengan nama **plant3** dari kelas plant dengan child **Flower**.
2. Objek dengan nama **plant4** dari kelas plant dengan child **Fungus**.
3. Saya ganti nama pada constructor **plant2** menjadi **“Sun-shroom”,** yang awalnya

**“Mushroom”**

Kemudian di paling bawah kode, saya tambahkan kode berikut untuk memanggil fungsi **talk()** pada kelas **Plant** dengan child **Flower** dan **Fungus**.



Baris kode **((Flower) plant1).bloom();** adalah contoh downcasting dalam Java, yaitu mengonversi referensi dari kelas induk **(Plant)** ke subclass spesifiknya **(Flower)**. Jika sudah silahkan di run:



Pusing gak? Jika pusing silahkan lakukan eksplorasi sendiri sampai paham. Manfaatkan internet untuk sumber alternatif belajar kalian.

## TIPS

Java inheritance:

[Video](https://www.youtube.com/watch?v=Zs342ePFvRI&pp=ygUZaW5oZXJpdGFuY2UgamF2YSBpbnRlbGxpag%3D%3D)

[Material](https://www.w3schools.com/java/java_inheritance.asp)

**OVERRIDING**



## TEORI

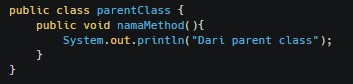


Method overriding adalah sebuah kemampuan yang membolehkan untuk mereplikasi body method utama yang ada di parent class, hal yang membuat berbeda parent method dengan override method terletak pada isi method-nya.

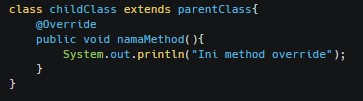
Secara sederhana method overriding dilakukan saat kita ingin membuat ulang sebuah method pada child class.

## MATERI

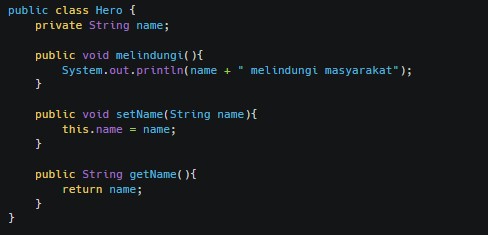
Method overriding dapat dibuat dengan menambahkan anotasi @Override di atas nama method atau sebelum pembuatan method. Contoh: Parent class



Child class



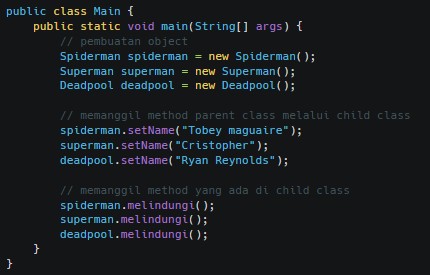
Contoh kasus penggunaan inheritance ialah, pada class Hero di atas terdapat method melindungi() dan kita ingin mengubah isi method-nya pada class child Superman menjadi “melindungi masyarakat bumi dari serangan monster”. Method melindungi() di class Hero tidak perlu diubah apa-apa dan tetap seperti ini:



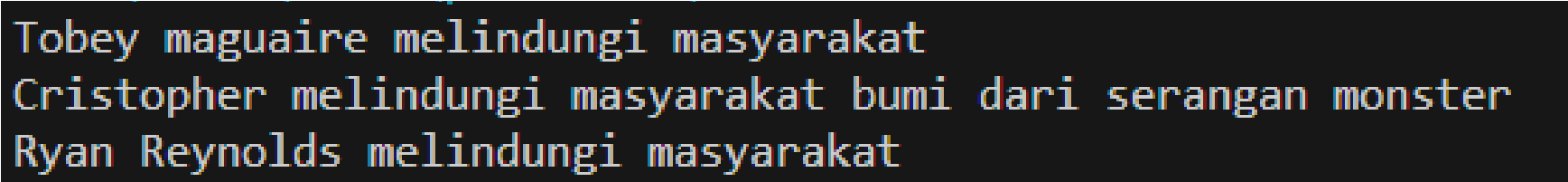
Tetapi pada class Superman kita ubah menjadi seperti ini:



Untuk mengetahui output dari perubahannya bisa kita panggil di Main class dengan cara yang sama ketika memanggil method biasa.



Terlihat pada akhir kode kita memanggil method yang sama pada 3 object berbeda, tetapi outputnya akan muncul seperti ini:

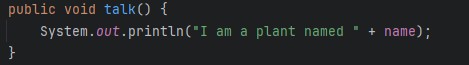


Hal ini bisa terjadi karena class Superman sudah melakukan override pada method melindungi(), jadi isi dari method sudah tidak lagi sama dengan isi method pada parent class. Berbeda dengan class Spiderman dan Deadpool yang tidak melakukan override maka isinya akan sama dengan method melindungi() pada parent class.

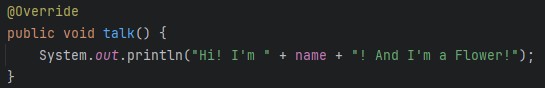
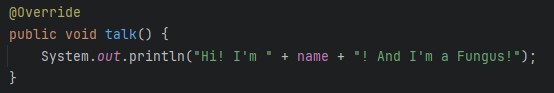
## PRAKTEK

Mari kita terapkan konsep Overriding ini dalam projek kita! Sebelum mulai, silahkan jawab pertanyaan ini: Kira-kira, teknik overriding ini dapat kita terapkan di bagian mana pada projek kita?

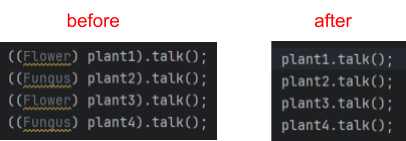
Benar! kita dapat menerapkannya pada method **talk()** pada kelas **Flower** dan **Fungus,** yang dimana itu merupakan child class dari parent class **Plant.** Silahkan tambahkan method baru dengan kata kunci **@Override** pada class **Plant** dengan spesifikasi sebagai berikut:



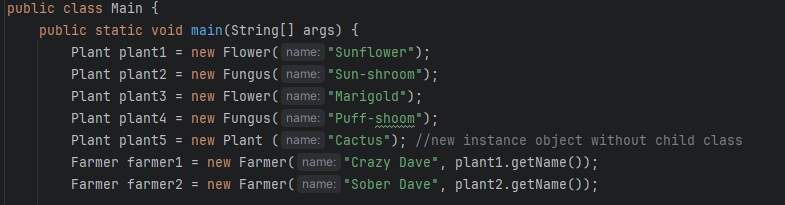
Kemudian kita tambahkan anotasi @Override diatas method talk() pada kelas Flower dan Fungus seperti berikut:



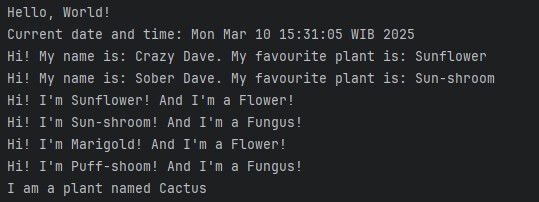
Jika sudah, maka kita berhasil melakukan overriding pada method talk(). Setelah itu kita sudah tidak usah melakukan downcasting pada pemanggilan method talk() seperti pada praktek sebelumnya. Kita bisa langsung saja menuliskan seperti ini:



Jika di run hasilnya akan sama saja seperti praktek sebelumnya. Jika kalian menambahkan objek baru tanpa menggunakan child clas. Maka dia akan mengeksekusi method **talk()** di kelas **Plant**. Contoh:



Output:



## TIPS

Video singkat penjelasan teknik overriding:

[Video](https://www.youtube.com/watch?v=CLzgS08equQ&pp=ygUXb3ZlcmlkaW5nIGphdmEgaW50ZWxsaWo%3D)

[Material](https://javarush.com/en/groups/posts/en.1975.mechanism-of-overriding-methods-or-override-in-java)

**SUPER**

**KEYWORDS**



## TEORI



Keyword super merupakan keyword yang digunakan untuk mengakses atribut ataupun method parent class dari child class. Di java, keyword ini dapat digunakan untuk melakukan beberapa hal, seperti berikut:

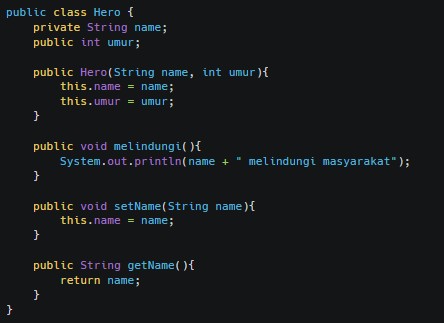
○ Mengakses method ataupun constructor parent class oleh child class

○ Menunjuk anggota parent class oleh child class

○ Mengakses method parent class oleh child class

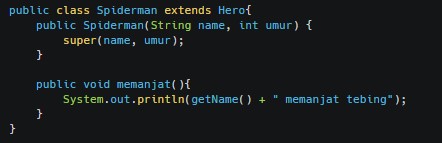
## MATERI

Untuk contoh penggunaan keywords super, kita coba membuat class Hero memiliki constructor yang digunakan untuk menginisialisasi nama hero dan umur. Berikut adalah kode pada setiap class: File: Hero.java

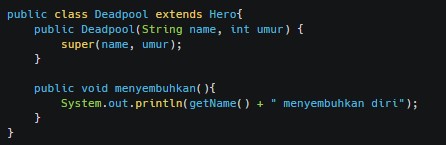


Parent class sudah kita ubah dengan bentuk constructor Hero(String name, int umur) maka pada child class akan menjadi seperti ini:

File: Spiderman.java



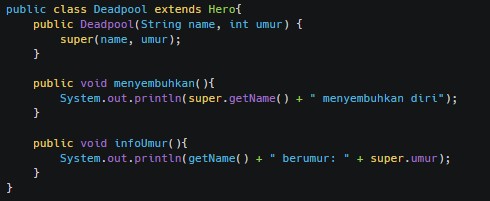
File: Deadpool.java



File: Superman.java



Terlihat pada child class Superman, Spiderman, dan Deadpool juga diwajibkan untuk memiliki constructor hal ini disebabkan parent class memiliki constructor. Tetapi di dalam constructor child class terpadu super(name, umur); yang dipanggil, maksud dari super() adalah memanggil method pada parent class yang di mana method itu adalah constructor dari class Hero. Apakah super() hanya bisa digunakan pada method saja? Jawabannya tidak, bisa juga digunakan untuk pemanggilan variabel yang ada di parent class melalui child class. Contohnya pada class Deadpool.java kita ubah menjadi seperti ini:

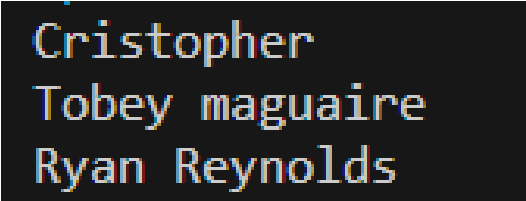


Pada class Deadpool di atas kita menggunakan keywords super untuk memanggil atribut umur dan juga method getName(), yang di mana atribut umur dan method getName() tidak ada di class Deadpool tapi ada di class Hero yang merupakan parent class. Perlu diperhatikan untuk penggunaan keywords super pada constructor, pemanggilan keywords super() harus berada di awal setelah pembuatan constructor child class agar tidak terjadi error.

Untuk cara pemanggilan pada Main class, adalah seperti berikut:



Adapun outputnya akan seperti ini:



## TIPS

Materi tentang super keywords:

[Materi](https://www.w3schools.com/java/ref_keyword_super.asp)

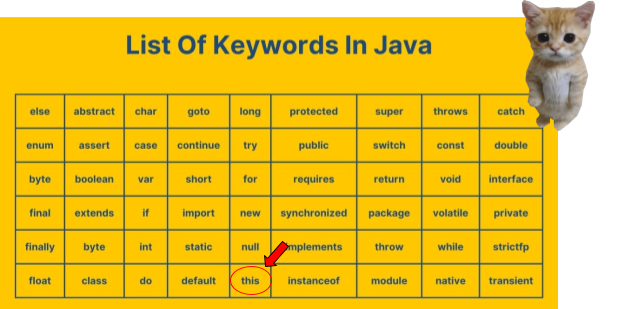
[Video](https://www.youtube.com/watch?v=Qb_NUn0TSAU&pp=ygUTc3VwZXIga2V5d29yZHMgamF2YQ%3D%3D)

**THIS**

**KEYWORDS**



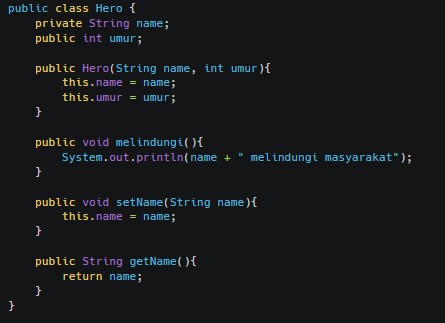
## TEORI



Sejauh materi ini kita sudah banyak belajar materi tentang constructor, encapsulation, inheritance, overriding, super. Mungkin di pertengahan atau di awal materi muncul sebuah pertanyaan apa yang dimaksud dengan keywords this? Jawabannya keywords this biasanya digunakan untuk mengisi variabel class atau mengakses variabel yang ada di class tersebut.

## MATERI

Mari kita lihat contoh dibawah ini:



Pada constructor Hero dan method setName() terdapat kode yang menggunakan keywords this. Dilihat baik-baik pada class Hero memiliki atribut String name dan int umur, pada constructor Hero(String name, int umur) terdapat atribut juga tapi sebagai parameter. Kedua hal ini berbeda untuk private String name dimiliki oleh class sedangkan String name yang ada di parameter constructor itu adalah parameter yang dimiliki oleh constructor. Pada constructor this.name = name memiliki sebuah arti bahwa atribut name yang awalnya kosong diisi oleh parameter name yang diisi oleh user.

Hal ini juga berlaku pada method setName(), karena method ini adalah setter atau method yang digunakan untuk inisialisasi sebuah nilai private, maka this.name = name juga memiliki arti atribut name pada class sekarang diisi oleh parameter name yang dimana pengisiannya dilakukan ketika pemanggilan method setter.

## TIPS

Materi tentang this keywords:

[Materi](https://www.w3schools.com/java/ref_keyword_this.asp)

[Video](https://www.youtube.com/watch?v=ETLHbHenW44&pp=ygUSdGhpcyBrZXl3b3JkcyBqYXZh)

**CODELAB**

**&**

**TUGAS**



## CODELAB

Buatlah program dalam bahasa Java yang mensimulasikan pertarungan antara Pahlawan dan Musuh menggunakan konsep materi yang ada di Modul 2 ini.

Program harus memiliki kelas-kelas berikut:

1. Kelas KarakterGame (Superclass)

* Memiliki atribut privat **nama** dan **kesehatan**.
* Memiliki **getter** dan **setter** untuk atribut **nama** dan **kesehatan**.
* Memiliki method **serang(KarakterGame target)** yang akan di-**override** oleh **subclass**.
* **Constructor** harus menerima **nama** dan **kesehatan** sebagai parameter.

2. Kelas Pahlawan (Subclass dari KarakterGame)

* Mewarisi **KarakterGame** dan menggunakan **super()** dalam constructor.
* **Meng-override** method **serang(KarakterGame target)**, dengan efek:

**○** Menampilkan pesan: "<nama Pahlawan> menyerang <nama target> menggunakan pedang!"

○ Mengurangi 20 poin kesehatan dari target.

○ Menampilkan kesehatan terbaru target.

3. Kelas Musuh (Subclass dari KarakterGame)

* Mewarisi **KarakterGame** dan menggunakan **super()** dalam constructor.
* **Meng-override** method **serang(KarakterGame target)**, dengan efek:
  1. Menampilkan pesan: "<nama Musuh> menyerang <nama target> menggunakan sihir!"

○ Mengurangi 15 poin kesehatan dari target.

○ Menampilkan kesehatan terbaru target. 4. Kelas Main (Kelas Utama) ● Membuat tiga objek:

**○ KarakterGame** bernama "**Karakter** **Umum**" dengan **100 kesehatan**.

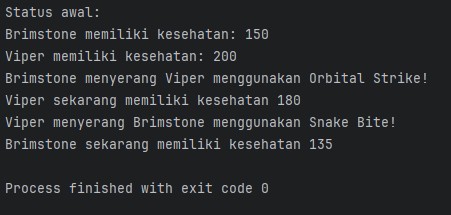
**○ Pahlawan** bernama "**Brimstone**" dengan **150 kesehatan**.

**○ Musuh** bernama "**Viper**" dengan **200 kesehatan**.

* Menampilkan status awal kesehatan **Pahlawan** dan **Musuh**.
* Memanggil method **serang()** untuk mensimulasikan pertarungan:
  1. **Brimstone** menyerang **Viper** menggunakan Orbital Strike.

○ **Viper** menyerang **Brimstone** menggunakan Snake Bite.

5. Contoh **output** yang diharapkan:



## TUGAS

Lanjutkan program login sederhana pada modul sebelumnya menggunakan konsep CONSTRUCTOR, ENCAPSULATION, INHERITANCE, OVERRIDING, SUPER KEYWORDS dalam bahasa **Java**. Program ini harus memiliki **tiga kelas utama**, yaitu:

1. **Kelas User** (sebagai superclass)
   1. Memiliki atribut nama dan nim yang dienkapsulasi (private).

○ Memiliki constructor untuk menginisialisasi nama dan nim.

○ Menyediakan getter dan setter untuk atribut nama dan nim.

○ Memiliki method login() yang akan di-*override* oleh subclass.

○ Memiliki method displayInfo() untuk menampilkan informasi pengguna.

1. **Kelas Admin** (sebagai subclass dari User)
   1. Memiliki atribut tambahan username dan password.

○ Constructor Admin menggunakan super untuk menginisialisasi nama dan nim.

○ Meng-*override* method login() untuk memeriksa kecocokan username dan password.

○ Meng-*override* method displayInfo() untuk menampilkan pesan login sukses.

1. **Kelas Mahasiswa** (sebagai subclass dari User)
   1. Constructor Mahasiswa menggunakan super untuk menginisialisasi nama dan nim.

○ Meng-*override* method login() untuk mencocokkan input nama dan nim.

○ Meng-*override* method displayInfo() untuk menampilkan pesan login sukses.

1. **Kelas LoginSystem** (sebagai program utama)
   1. Menyediakan pilihan login sebagai **Admin** atau **Mahasiswa**.

○ Jika pengguna memilih **Admin**, program meminta **username** dan **password** untuk login.

○ Jika pengguna memilih **Mahasiswa**, program meminta **nama** dan **nim** untuk login.

○ Jika login berhasil, akan ditampilkan informasi pengguna, jika gagal, muncul pesan error.

**Petunjuk Implementasi:**

* Gunakan **inheritance** untuk membuat Admin dan Mahasiswa sebagai subclass dari

User.

* Gunakan **overriding** pada method login() dan displayInfo().
* Gunakan **encapsulation** dengan menerapkan modifier private pada atribut dan menyediakan getter & setter.

**PENILAIAN**



## RUBRIK PENILAIAN

|  |  |
| --- | --- |
| **Aspek Penilaian** | **Poin** |
| **CODELAB** | **Total 30%** |
| Kerapian kode | 5% |
| Ketepatan kode & output | 15% |
| Kekreativitasan kode | 5% |
| Kode orisinal (tidak nyontek) | 5% |
| **TUGAS** | **Total 70%** |
| Kerapian kode | 5% |
| Ketepatan kode & output | 20% |
| Kode orisinal (tidak nyontek) | 5% |
| Kemampuan menjelaskan | 20% |
| Menjawab pertanyaan | 20% |
| **TOTAL** | **100%** |

## SKALA PENILAIAN



1. **= (81 - 100) → Sepuh**

**B+ = (75 - 80) → Sangat baik**

1. **= (70 - 74) → Baik**

**C+ = (60 - 69) → Cukup baik**

1. **= (55 - 59) → Cukup**
2. **= (41 - 54) → Kurang**
3. **= (0 - 40) → Bro really...**

**SUMMARY**

**AKHIR**

**MODUL**



Di modul kali ini, sudah disinggung sedikit tentang Polymorphism. Apa itu polymorphism? Kalian akan mempelajarinya lebih mendalam di modul selanjutnya. Intinya, polymorphism memungkinkan kalian untuk melakukan pemrograman dengan cara yang lebih abstrak namun efisien.

Pemahaman atau paradigma dalam pemrograman akan lebih mudah dipahami jika kita langsung mempraktekkannya dan bereksperimen daripada hanya membaca materi saja. Oleh karena itu, kemampuan berpikir abstrak dan imajinasi yang baik sangat diperlukan untuk menguasai konsep-konsep seperti ini.

Jangan lupa, belajar itu perjalanan. Salah itu wajar, malah dari kesalahan kalian akan lebih paham. Semisal nih nilai kalian di modul sebelumnya JELEK, kalian masih bisa maksimalkan di modul-modul berikutnya, atau di kelas teori. Jadi, tetap semangat dan terus eksplorasi. Siapa tahu, dari apa yang kalian pelajari sekarang, bisa membuat kalian menjadi programmer handal.