

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:

```
public class Mobil {  
    String nama;  
    int speed;  
  
    Mobil(){  
        System.out.println("Method constructor dieksekusi");  
    }  
}
```

Pada contoh di atas constructor ditulis seperti ini:

```
Mobil(){
    System.out.println("Method constructor dieksekusi");
}
```

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

```
Mobil mobil = new Mobil();
```

Sehingga sekarang kode pada Main class kita menjadi seperti ini:

```
public class Main{
    public static void main(String[] args){
        Mobil mobil = new Mobil();
    }
}
```

Ketika dijalankan akan menampilkan output berikut:

```
Method constructor dieksekusi

Process finished with exit code 0
```

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

```
class Mobil {
    String nama;
    int speed;

    Mobil(){
        System.out.println("Method constructor dieksekusi");
    }
}
```

2. Menggunakan modifier public

```
class Mobil {  
    String nama;  
    int speed;  
  
    public Mobil(){  
        System.out.println("Method constructor dieksekusi");  
    }  
}
```

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

```
class Mobil {  
    String nama;  
    int speed;  
  
    public Mobil(){  
        System.out.println("Method constructor dieksekusi");  
    }  
}
```

- pada class Main

```

public class Main{
    public static void main(String[] args){
        Mobil mobil1 = new Mobil();
        Mobil mobil2 = new Mobil();
        Mobil mobil3 = new Mobil();
        Mobil mobil4 = new Mobil();
        Mobil mobil5 = new Mobil();

        mobil1.nama = "Toyota Avanza";
        mobil1.speed = 160;
        mobil2.nama = "Suzuki SX4 2007";
        mobil2.speed = 170;
        mobil3.nama = "Mobil sedan";
        mobil3.speed = 210;
        mobil4.nama = "BMW i8";
        mobil4.speed = 190;
        mobil5.nama = "Bugatti Chiron";
        mobil5.speed = 240;
    }
}

```

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

```

class Mobil {
    String nama;
    int speed;

    public Mobil(String nama, int speed){
        this.nama = nama;
        this.speed = speed;
        System.out.println("Method constructor dieksekusi");
    }
}

```

- pada class Main

```

public class Main{
    public static void main(String[] args){
        Mobil mobil1 = new Mobil("Toyota Avanza", 160);
        Mobil mobil2 = new Mobil("Suzuki SX4 2007", 170);
        Mobil mobil3 = new Mobil("Mobil sedan", 210);
        Mobil mobil4 = new Mobil("BMW i8", 190);
        Mobil mobil5 = new Mobil("Bugatti Chiron", 240);
    }
}

```

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:

```

Mobil mobil5 = new Mobil("Porsche 911", 250);

```

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 😊


```
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        Farmer farmer1 = new Farmer();
        Farmer farmer2 = new Farmer();
        Plant plant1 = new Plant();
        Plant plant2 = new Plant();

        farmer1.name = "Crazy Dave";
        farmer2.name = "Sober Dave";

        plant1.name = "Sunflower";
        plant2.name = "Mushroom";

        farmer1.favourite = plant1.name;
        farmer2.favourite = plant2.name;

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());

        farmer1.talk();
        farmer2.talk();
    }
}
```

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.

```
public class Farmer {
    2 usages
    String name;
    2 usages
    String favourite;
    2 usages
    public Farmer(String name, String favourite){
        this.name = name;
        this.favourite = favourite;
    }
    2 usages
    void talk(){
        System.out.println("Hi! My name is: " + name + ". My favourite plant is: " + favourite);
    }
}
```

← Ini adalah constructor

```

public class Plant {
    3 usages
    String name;
    2 usages
    public Plant(String name){
        this.name = name;
    }
}

```

← Ini adalah constructor

Untuk penjelasan 'this' keywords bisa dibaca [disini](#). Dengan begitu, Class Main dapat kita buat jadi seperti ini:

```

import java.util.Date;

public class Main {
    public static void main(String[] args) {
        Plant plant1 = new Plant( name: "Sunflower");
        Plant plant2 = new Plant( name: "Mushroom");
        Farmer farmer1 = new Farmer( name: "Crazy Dave", plant1.name);
        Farmer farmer2 = new Farmer( name: "Sober Dave", plant2.name);

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());

        farmer1.talk();
        farmer2.talk();
    }
}

```

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](#)

[Video2](#)

Materi

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:

```
public class Mobil {
    private String nama;
    public int speed;
    protected boolean idDrive;

    public Mobil(){
        System.out.println("ini constructor");
    }
}
```

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:

```
package praktikum.Mobil;

class Mobil {
    private String name;

    public void changeName(String newName){
        this.name = newName;
    }
}
```

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.

```

1 import praktikum.Mobil;
2
3 public class Main {
4     public static void main(String[] args) throws Exception {
5         System.out.println("Hello, World!");
6         Mobil mobil = new Mobil();
7     }
8 }

```

Problems | File 5 | Project Errors 3 | Server-Side Analysis | Vulnerable Dependencies

Main.java C:\Users\WIRA YUDHA\Desktop\folder wira\project\InteliJ project\test\src 5 problems

- 'praktikum.Mobil' is not public in 'praktikum'. Cannot be accessed from outside package :1
- 'praktikum.Mobil' is not public in 'praktikum'. Cannot be accessed from outside package :6
- 'praktikum.Mobil' is not public in 'praktikum'. Cannot be accessed from outside package :6

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:

```

package praktikum.Mobil;

public class Mobil {
    private String name;

    public void changeName(String newName){
        this.name = newName;
    }
}

```

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 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:

```
package com.praktikum;

public class Mobil {
    protected String name;

    public void changeName(String newName){
        this.name = newName;
    }
}
```

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

```
package com.praktikum;

public class Honda extends Mobil{
    public void Driving(){
        name = "Civic";

        System.out.println("Driving menggunakan mobil Honda " + 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.

```
package com.main;
import com.praktikum.Mobil;

public class Main {
    public static void main(String[] args) throws Exception {
        Mobil mobil = new Mobil();

        // akan terjadi error di sini
        mobil.name = "Mobil civic";
    }
}
```

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:

```
package com.praktikum;

public class Mobil {
    private String name;

    public void setName(String newName){
        this.name = newName;
    }

    public String getName(){
        return name;
    }
}
```

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:

```
Mobil mobil = new Mobil();
mobil.name = "Civic"; // <= ini akan terjadi error
```

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 sebuah method saat kita memasukkan sebuah nilai/values ke dalam suatu variabel/atribut/object, sedangkan Getter adalah 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 supaya 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:

```
package com.praktikum;

public class Mobil {
    private String name;
    private String warna;
    private int maxSpeed;
    private boolean isModified;

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }

    public void setWarna(String warna){
        this.warna = warna;
    }

    public String getWarna(){
        return warna;
    }

    public void setMaxSpeed(int maxSpeed){
        this.maxSpeed = maxSpeed;
    }

    public int getMaxSpeed(){
        return maxSpeed;
    }

    public void setIsModified(boolean isModified){
        this.isModified = isModified;
    }

    public boolean getIsModified(){
        return isModified;
    }
}
```

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.


```
// ini method setter
public void setName(String name){
    this.name = name;
}

// ini method getter
public String getName(){
    return name;
}
```

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:

```
package com.main;
import com.praktikum.Mobil;

public class Main {
    public static void main(String[] args) throws Exception {

        Mobil mobil = new Mobil();
        // menggunakan method setter
        mobil.setName("Civic");
        mobil.setMaxSpeed(180);

        // menggunakan method getter
        System.out.println("Nama : " + mobil.getName());
        System.out.println("Max Speed : " + mobil.getMaxSpeed());
    }
}
```

Hasil output:

```
Nama : Civic
Max Speed : 180

Process finished with exit code 0
```

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 proyek praktek kita! Silahkan buka file projeknya. Lalu pergi ke Class Farmer dan Plant, lalu buat semua variabel menjadi private seperti ini:

```
public class Plant {
    1 usage 2 related problems
    private String name;
    2 usages
    public Plant(String name){
        this.name = name;
    }
}
```

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

```

public class Plant {
    2 usages
    private String name;
    2 usages
    public String getName(){
        return name;
    }
    no usages
    public String setName(String name){
        return name;
    }
    2 usages
    public Plant(String name){
        this.name = name;
    }
}

```

```

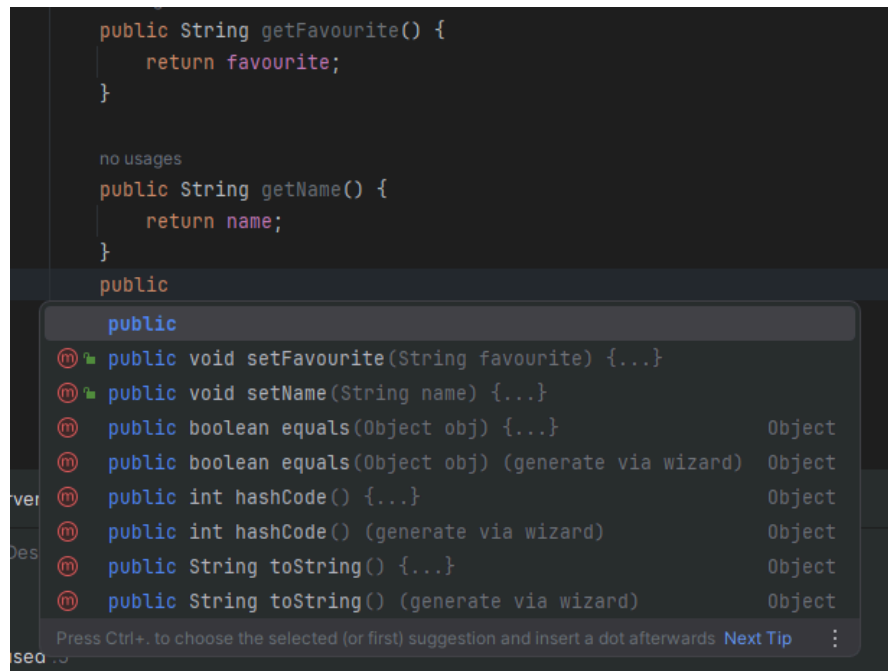
public class Farmer {
    4 usages
    private String name;
    4 usages
    private String favourite;

    no usages
    public String getFavourite() {
        return favourite;
    }
    no usages
    public String getName() {
        return name;
    }
    no usages
    public void setFavourite(String favourite) {
        this.favourite = favourite;
    }
    no usages
    public void setName(String name) {
        this.name = name;
    }

    2 usages
    public Farmer(String name, String favourite){
        this.name = name;
        this.favourite = favourite;
    }
    2 usages
}

```

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:

```
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        Plant plant1 = new Plant( name: "Sunflower");
        Plant plant2 = new Plant( name: "Mushroom");
        Farmer farmer1 = new Farmer( name: "Crazy Dave", plant1.getName());
        Farmer farmer2 = new Farmer( name: "Sober Dave", plant2.getName());

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());

        farmer1.talk();
        farmer2.talk();
    }
}
```

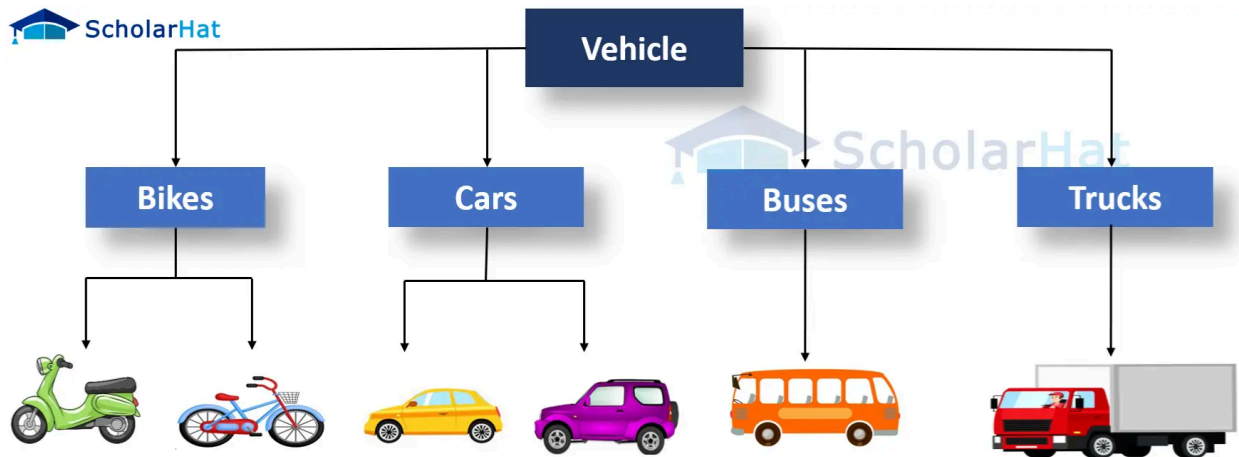
TIPS

Java encapsulation:

[Video](#)

Dasar getter dan setter:

[Video](#)

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

```
package com.praktikum;

public class Honda {
    public String name;

    public void bergerakMaju(){
        System.out.println("Mobil " + name + " bergerak maju");
    }

    public void berbelok(){
        System.out.println("Mobil berbelok");
    }
}
```

File: Toyota.java

```
package com.praktikum;

public class Toyota {
    public String name;

    public void bergerakMaju(){
        System.out.println("Mobil " + name + " bergerak maju");
    }

    public void berbelok(){
        System.out.println("Mobil berbelok");
    }
}
```

File: Nissan.java

```

package com.praktikum;

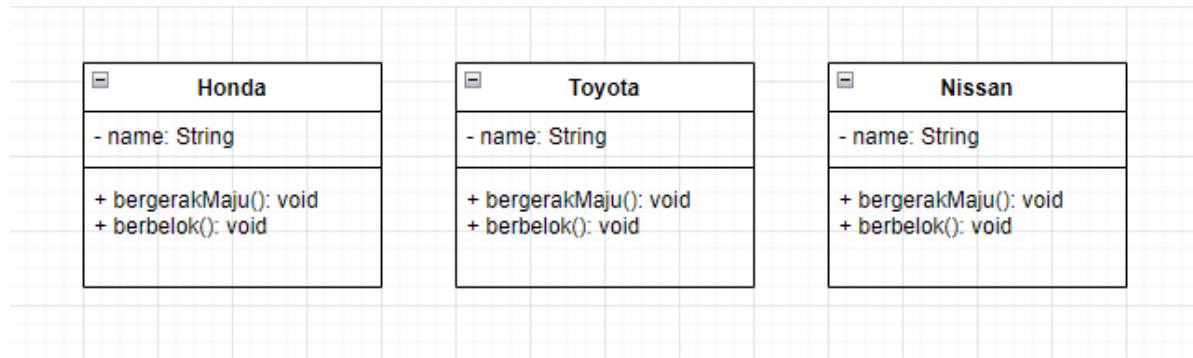
public class Nissan {
    public String name;

    public void bergerakMaju(){
        System.out.println("Mobil " + name + " bergerak maju");
    }

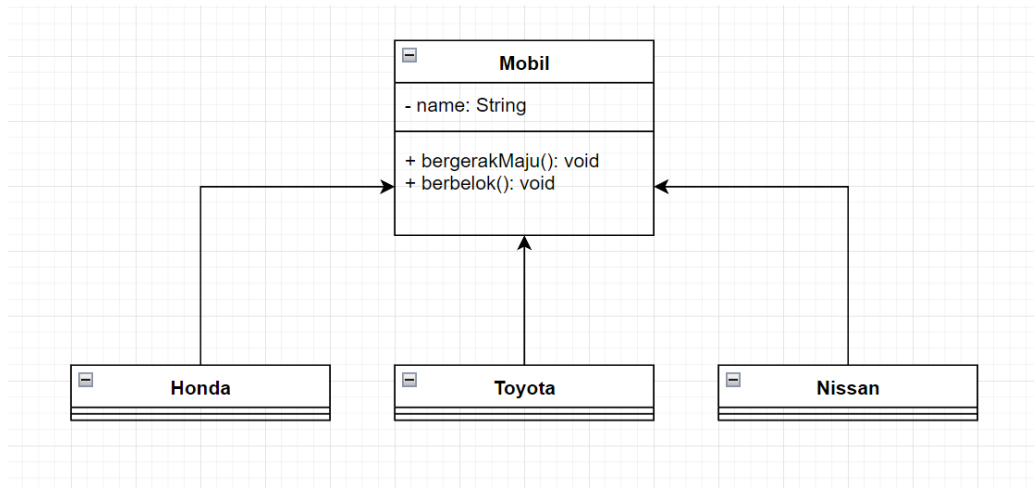
    public void berbelok(){
        System.out.println("Mobil berbelok");
    }
}

```

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:

```

package com.praktikum;

public class Mobil {
    private String name;

    public void bergerakMaju(){
        System.out.println("Mobil " + name + " bergerak maju");
    }

    public void berbelok(){
        System.out.println("Mobil berbelok");
    }
}
  
```

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

File: Honda.java

```

package com.praktikum;
public class Honda extends Mobil{

}
  
```

File: Toyota.java

```

package com.praktikum;
public class Toyota extends Mobil{

}
  
```


File: Nissan.java

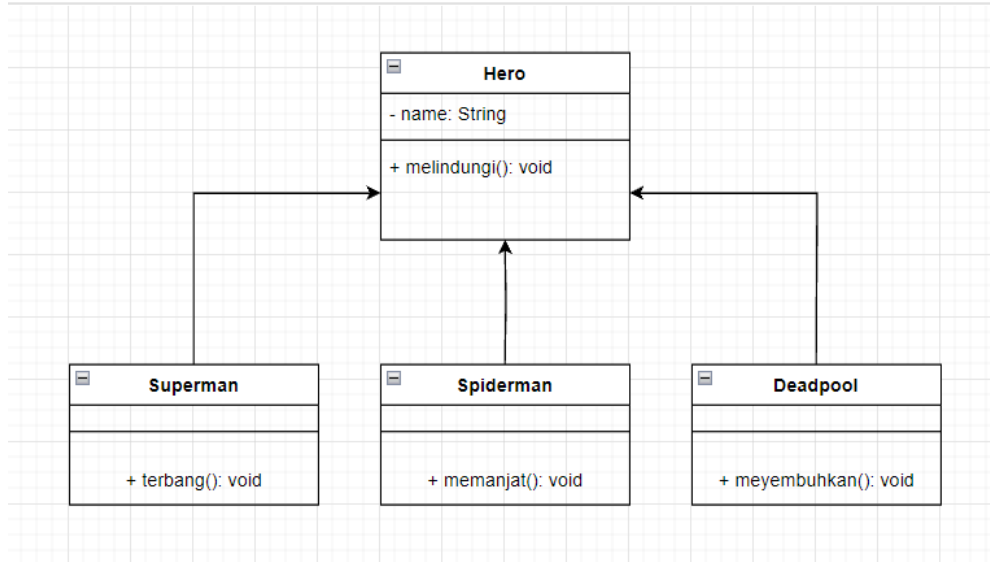
```
package com.praktikum;
public class Nissan extends Mobil{
}
}
```

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

```
Mobil mobil = new Mobil();
Honda honda = new Honda();
Toyota toyota = new Toyota();
Nissan nissan = new Nissan();
```

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

```

public class Hero {
    private String name;

    public void melindungi(){
        System.out.println(name + " melindungi masyarakat");
    }

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }
}

```

File: Superman.java

```

public class Superman extends Hero{
    public void terbang(){
        System.out.println(getName() + " terbang...");
    }
}

```

File: Spiderman.java

```

public class Spiderman extends Hero{
    public void memanjat(){
        System.out.println(getName() + " memanjat tebing");
    }
}

```

File: Deadpool.java

```

public class Deadpool extends Hero{
    public void menyembuhkan(){
        System.out.println(getName() + " menyembuhkan diri");
    }
}

```

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

```

public class Main {
    public static void main(String[] args) {
        // pembuatan object
        Spiderman spiderman = new Spiderman();
        Superman superman = new Superman();
        Deadpool deadpool = new Deadpool();

        // memanggil method parent class melalui child class
        spiderman.setName("Tobey maguaire");
        superman.setName("Cristopher");
        deadpool.setName("Ryan Reynolds");

        // memanggil method yang ada di child class
        spiderman.memanjat();
        superman.terbang();
        deadpool.menyembuhkan();

        // Jika butuh object lain bisa membuat lagi
        Spiderman spiderman2 = new Spiderman();
    }
}

```

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:

```

class Flower extends Plant {
    no usages
    public Flower(String name) {
        |    super(name);
    }
    no usages
    public void talk() {
        |    System.out.println("Hi! I'm " + name + "! And I'm a Flower!");
    }
}



```

```

class Fungus extends Plant {
    no usages
    public Fungus(String name) {
        super(name);
    }
    no usages
    public void talk() {
        System.out.println("Hi! I'm " + name + "! And I'm a Fungus!");
    }
}

```

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

Tips: anda bisa melihat error description dengan menekan  1 di pojok kanan atas. Maka akan terlihat dimana letak errornya:  'name' has private access in 'Plant' :6

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 kreativitasan kalian.)

```

public class Main {
    public static void main(String[] args) {
        Plant plant1 = new Flower( name: "Sunflower");
        Plant plant2 = new Fungus( name: "Sun-shroom");
        Plant plant3 = new Flower( name: "Marigold"); //new instance object
        Plant plant4 = new Fungus( name: "Puff-shoom"); //new instance object
        Farmer farmer1 = new Farmer( name: "Crazy Dave", plant1.getName());
        Farmer farmer2 = new Farmer( name: "Sober Dave", plant2.getName());
    }
}

```

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**.

```
((Flower) plant1).talk();
((Fungus) plant2).talk();
((Flower) plant3).talk();
((Fungus) plant4).talk();
```

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:

```
Hello, World!
Current date and time: Mon Mar 10 14:22:27 WIB 2025
Hi! My name is: Crazy Dave. My favourite plant is: Sunflower
Hi! My name is: Sober Dave. My favourite plant is: Sun-shroom
Hi! I'm Sunflower! And I'm a Flower!
Hi! I'm Sun-shroom! And I'm a Fungus!
Hi! I'm Marigold! And I'm a Flower!
Hi! I'm Puff-shoom! And I'm a Fungus!

Process finished with exit code 0
```

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

TIPS

Java inheritance:

[Video](#)

[Material](#)

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

```
public class parentClass {
    public void namaMethod(){
        System.out.println("Dari parent class");
    }
}
```

Child class

```
class childClass extends parentClass{
    @Override
    public void namaMethod(){
        System.out.println("Ini method override");
    }
}
```

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:

```
public class Hero {
    private String name;

    public void melindungi(){
        System.out.println(name + " melindungi masyarakat");
    }

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }
}
```

Tetapi pada class Superman kita ubah menjadi seperti ini:

```
public class Superman extends Hero{
    public void terbang(){
        System.out.println(getName() + " terbang...");
    }

    @Override
    public void melindungi(){
        System.out.println(getName() + " melindungi masyarakat bumi dari serangan monster");
    }
}
```

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

```

public class Main {
    public static void main(String[] args) {
        // pembuatan object
        Spiderman spiderman = new Spiderman();
        Superman superman = new Superman();
        Deadpool deadpool = new Deadpool();

        // memanggil method parent class melalui child class
        spiderman.setName("Tobey maguaire");
        superman.setName("Cristopher");
        deadpool.setName("Ryan Reynolds");

        // memanggil method yang ada di child class
        spiderman.melindungi();
        superman.melindungi();
        deadpool.melindungi();
    }
}

```

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

```

Tobey maguaire melindungi masyarakat
Cristopher melindungi masyarakat bumi dari serangan monster
Ryan Reynolds melindungi masyarakat

```

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 proyek kita! Sebelum mulai, silahkan jawab pertanyaan ini: Kira-kira, teknik overriding ini dapat kita terapkan di bagian mana pada proyek 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:

```
public void talk() {
    System.out.println("I am a plant named " + name);
}
```

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

```
@Override
public void talk() {
    System.out.println("Hi! I'm " + name + "! And I'm a Fungus!");
}
```

```
@Override
public void talk() {
    System.out.println("Hi! I'm " + name + "! And I'm a Flower!");
}
```

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:

before

```
((Flower) plant1).talk();
((Fungus) plant2).talk();
((Flower) plant3).talk();
((Fungus) plant4).talk();
```

after

```
plant1.talk();
plant2.talk();
plant3.talk();
plant4.talk();
```

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:

```

public class Main {
    public static void main(String[] args) {
        Plant plant1 = new Flower( name: "Sunflower");
        Plant plant2 = new Fungus( name: "Sun-shroom");
        Plant plant3 = new Flower( name: "Marigold");
        Plant plant4 = new Fungus( name: "Puff-shoom");
        Plant plant5 = new Plant ( name: "Cactus"); //new instance object without child class
        Farmer farmer1 = new Farmer( name: "Crazy Dave", plant1.getName());
        Farmer farmer2 = new Farmer( name: "Sober Dave", plant2.getName());
    }
}

```

Output:

```

Hello, World!
Current date and time: Mon Mar 10 15:31:05 WIB 2025
Hi! My name is: Crazy Dave. My favourite plant is: Sunflower
Hi! My name is: Sober Dave. My favourite plant is: Sun-shroom
Hi! I'm Sunflower! And I'm a Flower!
Hi! I'm Sun-shroom! And I'm a Fungus!
Hi! I'm Marigold! And I'm a Flower!
Hi! I'm Puff-shoom! And I'm a Fungus!
I am a plant named Cactus

```

TIPS

Video singkat penjelasan teknik overriding:

[Video](#)

[Material](#)

SUPER KEYWORDS

TEORI



List Of Keywords In Java

else	abstract	char	goto	long	protected	super	throws	catch
enum	assert	case	continue	try	public	switch	const	double
byte	boolean	var	short	for	requires	return	void	interface
final	extends	if	import	new	synchronized	package	volatile	private
finally	byte	int	static	null	implements	throw	while	strictfp
float	class	do	default	this	instanceof	module	native	transient

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`

```

public class Hero {
    private String name;
    public int umur;

    public Hero(String name, int umur){
        this.name = name;
        this.umur = umur;
    }

    public void melindungi(){
        System.out.println(name + " melindungi masyarakat");
    }

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }
}

```

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

File: Spiderman.java

```

public class Spiderman extends Hero{
    public Spiderman(String name, int umur) {
        super(name, umur);
    }

    public void memanjat(){
        System.out.println(getName() + " memanjat tebing");
    }
}

```

File: Deadpool.java

```

public class Deadpool extends Hero{
    public Deadpool(String name, int umur) {
        super(name, umur);
    }

    public void menyembuhkan(){
        System.out.println(getName() + " menyembuhkan diri");
    }
}

```

File: Superman.java

```

public class Superman extends Hero{

    public Superman(String name, int umur) {
        super(name, umur);
    }

    public void terbang(){
        System.out.println(getName() + " terbang...");
    }

    @Override
    public void melindungi(){
        System.out.println(getName() + " melindungi masyarakat bumi dari serangan monster");
    }
}

```

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:

```

public class Deadpool extends Hero{
    public Deadpool(String name, int umur) {
        super(name, umur);
    }

    public void menyembuhkan(){
        System.out.println(super.getName() + " menyembuhkan diri");
    }

    public void infoUmur(){
        System.out.println(getName() + " berumur: " + super.umur);
    }
}

```

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:

```
public class Main {  
    public static void main(String[] args) {  
        // pembuatan object disertai dengan argumen yang sesuai  
        Spiderman spiderman = new Spiderman("Tobey maguaire", 34);  
        Superman superman = new Superman("Cristopher", 43);  
        Deadpool deadpool = new Deadpool("Ryan Reynolds", 36);  
  
        // coba panggil  
        System.out.println(superman.getName());  
        System.out.println(spiderman.getName());  
        System.out.println(deadpool.getName());  
    }  
}
```

Adapun outputnya akan seperti ini:

```
Cristopher  
Tobey maguaire  
Ryan Reynolds
```

TIPS

Materi tentang super keywords:

[Materi](#)

[Video](#)

THIS KEYWORDS**TEORI**


else	abstract	char	goto	long	protected	super	throws	catch
enum	assert	case	continue	try	public	switch	const	double
byte	boolean	var	short	for	requires	return	void	interface
final	extends	if	import	new	synchronized	package	volatile	private
finally	byte	int	static	null	implements	throw	while	strictfp
float	class	do	default	this	instanceof	module	native	transient

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:

```

public class Hero {
    private String name;
    public int umur;

    public Hero(String name, int umur){
        this.name = name;
        this.umur = umur;
    }

    public void melindungi(){
        System.out.println(name + " melindungi masyarakat");
    }

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }
}

```

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](#)

[Video](#)

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:
 - 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:
 - **Brimstone** menyerang **Viper** menggunakan Orbital Strike.
 - **Viper** menyerang **Brimstone** menggunakan Snake Bite.

5. Contoh **output** yang diharapkan:

```
Status awal:
Brimstone memiliki kesehatan: 150
Viper memiliki kesehatan: 200
Brimstone menyerang Viper menggunakan Orbital Strike!
Viper sekarang memiliki kesehatan 180
Viper menyerang Brimstone menggunakan Snake Bite!
Brimstone sekarang memiliki kesehatan 135

Process finished with exit code 0
```

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)
 - 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.
2. **Kelas Admin** (sebagai subclass dari User)
 - 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.

3. Kelas Mahasiswa (sebagai subclass dari User)

- 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.

4. Kelas LoginSystem (sebagai program utama)

- 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:

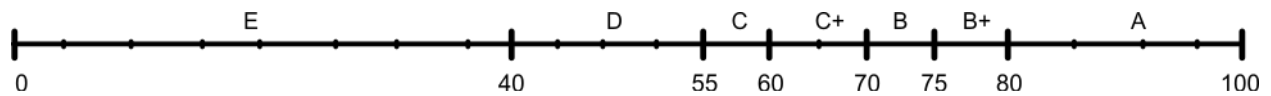
- Gunakan **inheritance** untuk membuat Admin dan Mahasiswa sebagai subclass dari User.
- Gunakan **overriding** pada method `login()` dan `displayInfo()`.
- Gunakan **encapsulation** dengan menerapkan modifier `private` pada atribut dan menyediakan getter & setter.

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

A = (81 - 100) → Sepuh

B+ = (75 - 80) → Sangat baik

B = (70 - 74) → Baik

C+ = (60 - 69) → Cukup baik

C = (55 - 59) → Cukup

D = (41 - 54) → Kurang

E = (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 mempraktekannya 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.