ENTERPRISE APPLICATION DEVELOPMENT

Modul 2

Object Oriented Programming

2022

MODUL 2

INHERITANCE & POLIMORFISME (OVERIDING & OVERLOADING)

prepared by :
EAD Laboratory

ENTERPRISE APPLICATION DEVELOPMENT

ASISTEN PRAKTIKUM

Abiyoga Dhaniswara
ABII

Ahmad Syahid Danu
ASDW

Amilia Agata
MILY

Dharu Widhyanugrah
DINO

Erina Zahira Nurhasan
RYIN

Fadli Kurniawan
YORU

Farel Baihaky
YNWA

Fazar Arya Suwandi
RJEP

Gede Dipta Narayana
GING

Hudzaifah Afif
GGEZ

Indira Agustia Garini
DIRA

Luh Komang Devi
DEVS

Maulana Rakha R
MARK

Metta Triana Asri
JMET

Mohamad Fahrudin
LFTG

M. Nanda Nugraha
NANZ

M. Haris Sitompul
RTSU

Nabila Melsyana
LALA

Rafidah Jasmine L
MINT

Windy Kurniawan
WNDD

Yoga Raditya N
GOYS

Contents

Peraturan Praktikum

1. Setiap peserta praktikum harus datang tepat waktu sesuai dengan jadwal. Toleransi keterlambatan hadir 15 menit. Jika melebihi batas waktu tersebut diperkenankan mengikuti praktikum namun tidak mendapatkan tambahan waktu dan/atau nilai.
2. Perizinan Praktikum :
   a. Izin berkaitan dengan Sakit atau Kemalangan , maka praktikan dapat memberikan surat perizinan kepada pihak Komisi Disiplin maksimal 3 hari setelah jadwal (shift) praktikum.
   b. Izin lomba atau penugasan institusi tidak berlaku apabila tidak terdapat bukti dispensasi dari Igracias. NB : screenshot dispensasi dari igracias wajib dilampirkan.
3. Seragam Praktikum :
   a. Mahasiswa wajib menggunakan celana bahan hitam (bukan chino atau jeans) pada saat praktikum.
   b. Mahasiswi wajib menggunakan rok hitam/biru gelap panjang tidak ketat pada saat praktikum.
   c. Dresscode praktikum (Mengikuti Peraturan Telkom)
      i. Senin : Menggunakan kemeja merah telkom atau kemeja putih polos
      ii. Selasa s/d Rabu : Menggunakan kemeja putih polos
      iii. Kamis s/d Sabtu : Menggunakan kemeja formal berkerah (bukan kerah sanghai dan bukan polo)
      iv. Jumat : Menggunakan kemeja/ baju batik bukan outer.
   d. Jika terdapat kendala dalam baju seragam maka praktikan diperbolehkan mengganti menggunakan kemeja putih Telkom atau polos..
   e. Membuka sepatu saat memasuki ruangan lab.
4. Peraturan Pengerjaan
   a. Studi Kasus dikerjakan secara individu.
   b. Jawaban tidak boleh sama dengan setiap individu.
   c. Hasil pengerjaan di push ke repository github tiap individu.
   d. Submit hasil pengerjaan berupa file format PDF dan link github ke LMS tiap individu. Screenshot semua pengerjaan dan output dalam bentuk full screen dan menampilkan tanggal serta waktu.
   e. Format Pengumpulan Hasil Pengerjaan
      - **Format penamaan file yang dikumpulkan di LMS**
        - Format nama file Jurnal PDF:
          - **OOP_KODEASISTEN_NAMAPENDEK_NIM_SSMODULX**
          - Contoh: **OOP_MILY_AMILIA_1202200000_ SSMODUL1**

Peraturan Praktikum

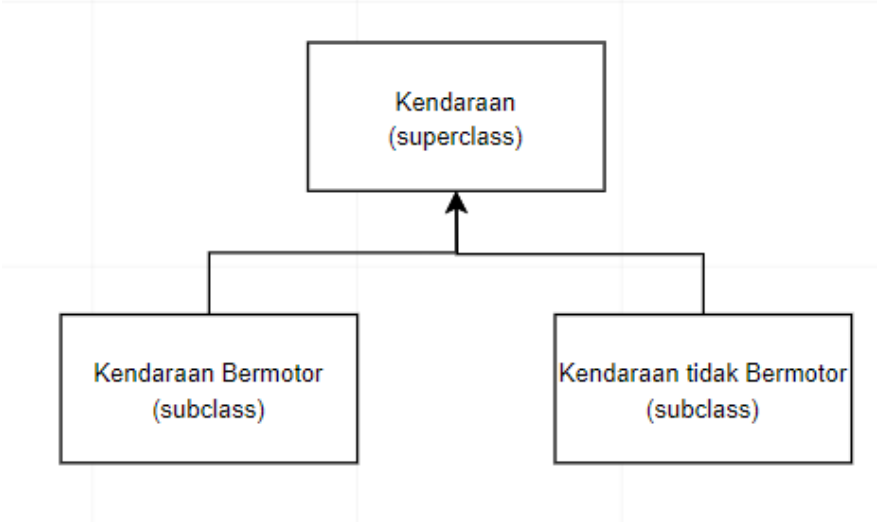
- Format nama file Tugas Pendahuluan PDF:
  - **OOP_KODEASISTEN_NAMAPENDEK_NIM_SSTPX**
  - Contoh: **OOP_MILY_AMILIA_1202200000_ SSTP1**

- **Format penamaan folder yang di push ke repository github**
  - Alur penamaan folder:
    - Folder Modul"**MODUL X**"
    Create Project untuk TP dan Jurnal :
      - Folder Project TP: "**TPMODULX_NAMAPENDEK**" (Di dalam Folder Modul)
      - Folder Project Jurnal: "**MODULX_NAMAPENDEK**" (Di dalam Folder Modul)
  - Format nama repository
    - **OOP-KODE ASISTEN-NAMA PENDEK PRAKTIKAN-NIM**
    - Contoh : **OOP-MILY-AMILIA-1202200000**
  - Format nama folder Jurnal:
    - **MODULX_NAMAPENDEK**
    - Contoh: **MODUL1_AMILIA**
  - Format nama folder Tugas Pendahuluan:
    - **TPMODULX_NAMAPENDEK**
    - Contoh: **TPMODUL1_AMILIA**

f. Salah penamaan pada file pengerjaan nilai modul akan dipotong sebesar **10%**
g. Terlambat mengumpulkan file pengerjaan nilai modul akan dipotong sebesar **15%.**
h. Terlambat commit pengerjaan nilai modul akan dipotong sebesar **15%.**
i. Segala alat komunikasi dikumpulkan di loker yang tersedia dalam ruangan.
j. Jika ada perangkat praktikum yang bermasalah dapat menghubungi asprak yang bertugas.
k. Segala bentuk kecurangan dan plagiarisme akan diproses ke komisi disiplin dan nilai akhir modul menjadi 0

Inheritance

Inheritance atau pewarisan adalah kemampuan untuk menurunkan sebuah class ke class lain, dalam artian, kita dapat membuat class Parent (*superclass*) dan class Child (*subclass*). Adapun ketentuan yang berlaku seperti class Child hanya bisa punya satu class Parent, namun satu class Parent bisa punya banyak class Child. Saat sebuah class diturunkan, maka semua atribut/field dan method yang ada di class Parent, secara otomatis akan dimiliki oleh class Child, tetapi hanya untuk modifier public dan protected saja. Untuk melakukan pewarisan, di class Child, kita harus menggunakan kata kunci ***extends*** lalu diikuti dengan nama class Parent nya.



Keterangan:
Class Kendaraan merupakan Parent Class (superclass) dari Class Kendaraan Bermotor dan Class Kendaraan Tak Bermotor, maka kedua Class tersebut memiliki ciri-ciri dan sifat yang mirip seperti Class kendaraan. Selain itu kedua Child Class (subclass) tersebut juga dapat memiliki ciri-ciri unik yang berbeda dari Class lainnya.

Polymorfisme

Polymorphism (Polimorfisme) berasal dari kata *poli* yang berarti "banyak" dan *morph* yang berarti bentuk sehingga secara sederhana polimorfisme merupakan kemampuan objek, variabel, atau fungsi yang dapat memiliki berbagai bentuk. Secara umum *polymorphism* dalam OOP terjadi ketika suatu SuperClass direferensikan ke dalam SubClass. Alhasil kita dapat mengembangkan sebuah program secara umum,bukan spesifik.

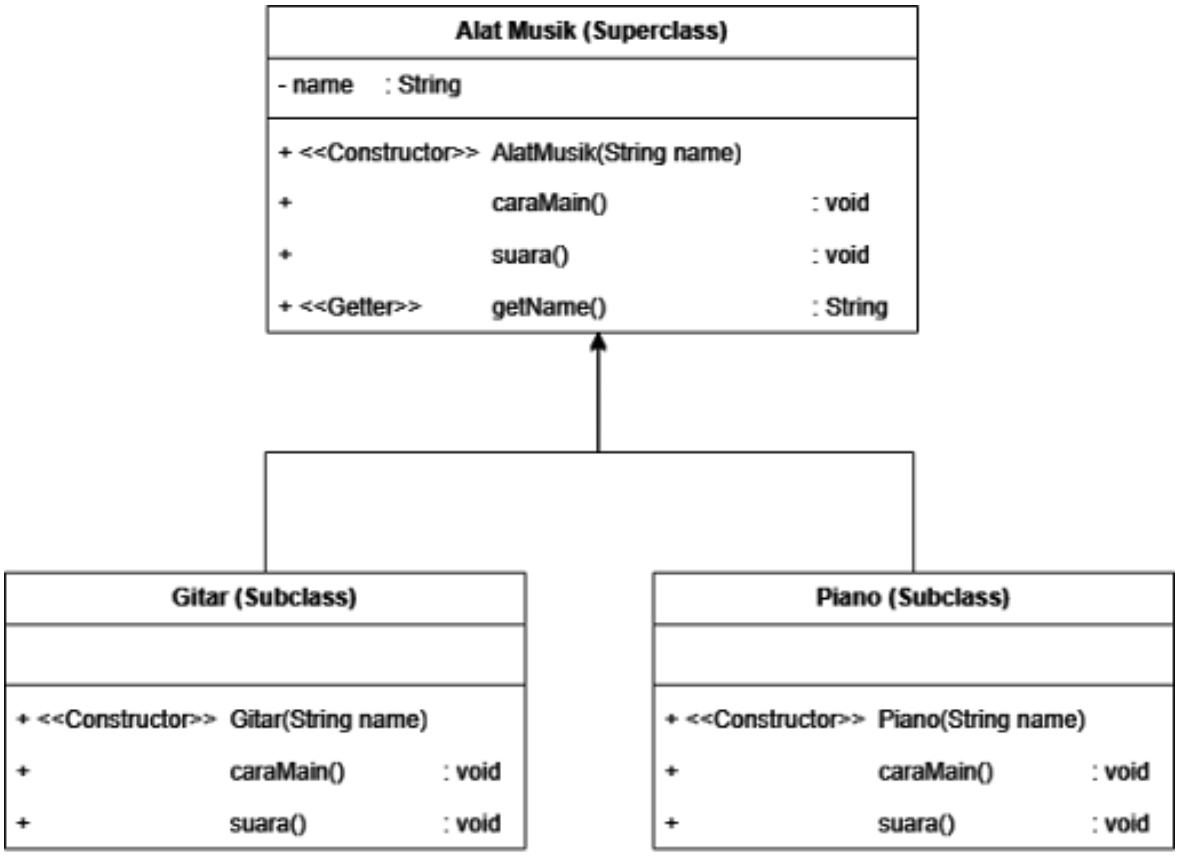
Polimorfisme dapat digambarkan seperti seorang yang menguasai dan mampu berbicara dalam beberapa bahasa. Nah di sini, seseorang bertindak sebagai obyek dan kemampuan dia berbicara merupakan sebuah polimorfisme. Contoh lainnya adalah smartphone. Selain sebagai alat komunikasi, smartphone yang bertindak sebagai objek dapat digunakan sebagai kamera, pemutar musik dan radio. Pada dasarnya, polimorfisme dapat dibagi menjadi dua, yaitu:

- Compile Time Polymorphism (**Overloading**)
- Runtime Polymorphism (**Overriding**)

Selain itu polimorfisme memiliki beberapa karakteristik, yaitu:

1. Dalam method overloading, jumlah dan tipe parameter serta urutannya atau dapat disebut dengan istilah method signature mengalami perubahan, sedangkan pada method overriding tidak berubah.
2. Method overloading dapat dilakukan pada class yang sama, namun method overriding hanya dapat dilakukan pada subclass dengan signature yang sama dan tipe return yang sama. Sedangkan pada method overloading, tipe return ini bisa sama atau berbeda.
3. Method private final atau static tidak dapat di override, namun method tersebut bisa overloading. Artinya suatu class dapat memiliki lebih dari satu method private final atau static dengan nama yang sama, namun subclass tidak dapat override private final atau static method tersebut dari super class.
4. Method overloading pada Java terikat oleh static binding sedangkan method overriding merupakan subjek dari dynamic binding.
5. Seperti pada keterangan jenis polimorfisme diatas method overloading terjadi pada saat compile time sedangkan overriding terjadi pada waktu runtime.

Adapun contoh dari penggunaan polimorfisme sebagai berikut:



Method **caraMain()** dan **suara()** di class-class child dari *class* **AlatMusik** adalah contoh *overriding*. Dengan begitu **caraMain()** dan **suara()** pada *subclass* akan didefinisikan ulang menggantikan method pada *superclass*. Berikut adalah contoh penerapan class diagram untuk *class* **AlatMusik** dan **Gitar**, dimana method **caraMain()** dan **suara()** di *class* **AlatMusik** di *Override* pada *class* **Gitar**.

Input :

```java
// Kelas superclass AlatMusik
class AlatMusik {
    private String name;

    // Konstruktor
    public AlatMusik(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void caraMain() {
        System.out.println("Cara bermain alat musik ini umumnya...");
    }

    public void suara() {
        System.out.println("Menghasilkan suara...");
    }
}
```

```java
// Kelas subclass Gitar
class Gitar extends AlatMusik {
    public Gitar(String name) {
        super(name);
    }

    @Override
    public void caraMain() {
        System.out.println("Cara bermain gitar adalah dengan cara dipetik.");
    }

    @Override
    public void suara() {
        System.out.println("Jreng.... Jrengg...");
    }
}
```

Output :

```
[Running] cd "d:\[Projects]\Portfolio\Portfolio Website\" && javac Main.java && java Main
Informasi alat musik:
Nama: Gitar Akustik
Cara bermain gitar adalah dengan cara dipetik.
Jreng.... Jrengg...
```

Overriding

R*untime* polimorfisme adalah proses di mana sebuah fungsi dipanggil pada saat *runtime*. Contoh dari runtime polimorfisme adalah *method Overriding*, yaitu sebuah kelas yang memiliki fungsi dengan nama yang sama dengan fungsi yang di dalam kelas induknya.

Secara sederhana, *method overriding* adalah sebuah metode yang memungkinan *subclass* mewarisi sebuah implementasi yang spesifik dari sebuah fungsi yang ada pada kelas induknya (*parent class*), implementasi tersebut berupa *method* yang sama dengan *parent class*-nya namun memiliki implementasi yang berbeda. Implementasi pada subclass akan menimpa atau mengganti implementasi pada *parent class*.

Berikut merupakan ciri-ciri dari *method overriding*:

1. Nama *method* dari *child class* harus sama dengan method di parent class.
2. Parameter dari *method child class* harus sama dengan *method* di *parent class*.
3. *Return Type* dari *method child class* harus sama dengan *method* di *parent class*.

Super

Dalam bahasa pemrograman Java, super adalah perintah khusus untuk mengakses *parent class*. Perintah ini biasanya dipakai untuk **mengakses method yang tertimpa (*overridden*)** atau **constructor milik parent class**. Kata super sendiri berasal dari superclass yang menjadi sebutan lain dari *parent class*.

Berikut adalah contoh dari penggunakan super

Class Parent

```java
package Parent;

public class Parent {
    String name;
    int age;

    Parent(String name, int age){
        this.name = name;
        this.age = age;
    }

    void sayHello(){
        System.out.println("Halo saya " + name +"Umurnya " + age);
    }
}
```

Class Child One

```java
package Parent;

public class ChildOne extends Parent {
    String school;

    ChildOne(String name, int age){
        super(name, age);
    }

    void sayHello(String school){

// Memanggil metode sayHello() dari super class menggunakan super
        super.sayHello();
        System.out.println("Sekolah di " + school);
    }
}
```

Class Main

```java
package Parent;

public class MainApp {
    public static void main(String[] args) {
        Parent parent = new Parent("Bapak ", 40);
        parent.sayHello();

        ChildOne childOne = new ChildOne("Anak Kesatu ", 17);
        childOne.sayHello("Telkom");

    }


}
```

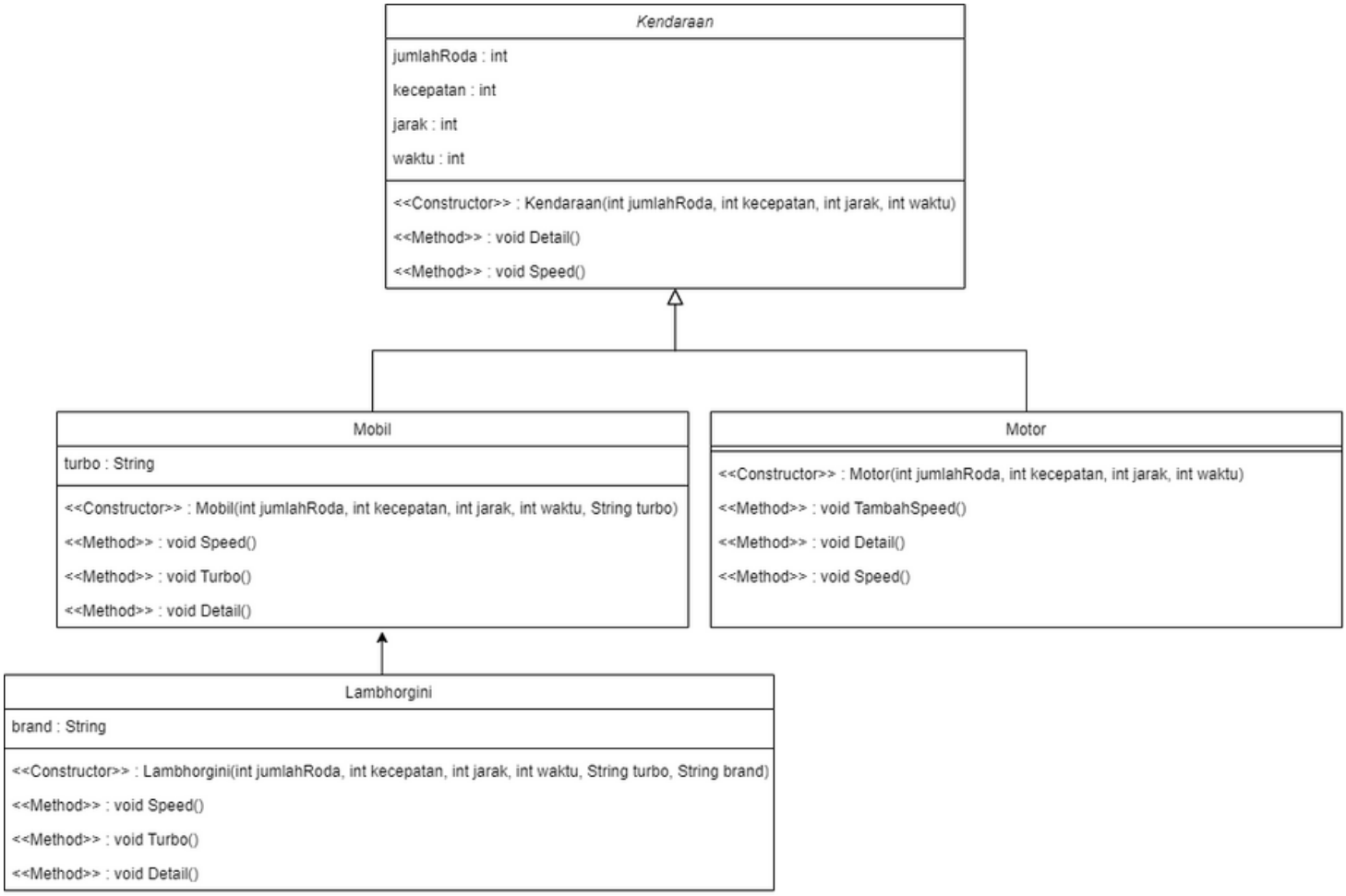
Output

```
Halo saya Bapak Umurnya 40
====================
Halo saya Anak Kesatu Umurnya 17
Sekolahnya di  Telkom
```

Pembahasan:

- Kelas Parent memiliki dua atribut, yaitu name dan age, serta sebuah konstruktor untuk menginisialisasi atribut tersebut.
- Terdapat juga metode **sayHello()** yang mencetak informasi mengenai nama dan usia**.**
- Kelas ChildOne merupakan subclass dari Parent.
- Konstruktor ChildOne menggunakan kata kunci **super(name, age)** untuk memanggil konstruktor dari super class (Parent) dan menginisialisasi atribut name dan age.
- Dalam kelas **ChildOne**, metode **sayHello(String school)** memanggil metode **sayHello()** dari super class menggunakan kata kunci **super.sayHello()**.
- Dengan memanggil **super.sayHello()**, kita dapat menjalankan metode **sayHello()** dari super class (**Parent**)
- Hasil keluaran dari metode **sayHello()** pada objek **childOne** akan mencetak informasi dari metode **sayHello()** di super class, diikuti dengan informasi tambahan dari metode **sayHello(String school)** di subclass.

Contoh dari penggunaan method overriding :



Terdapat 5 class seperti diatas yaitu *Class* Kendaraan, *Class* Mobil, *Class* Motor, dan *Class* Lamborghini, dan Class Main

Berikut adalah rincian dari keempat kelas tersebut :

1. *Class* Kendaraan
   a. Terdapat 4 variabel *protected* yaitu jumlahRoda(int), kecepatan(int), jarak(int), waktu(int).
   b. Satu *constructor* Kendaraan (int jumlahRoda, int kecepatan, int jarak, int waktu)
   c. Dua *Method public*
2. *Class* Mobil
   a. Terdapat satu variabel *protected* yaitu turbo(String).
   b. Satu *constructor* Mobil (int jumlahRoda, int kecepatan, int jarak, int waktu, String turbo)
   c. Dua *method override*
   d. Satu *method public*
3. Class Motor
   a. Satu *constructor* Motor(int jumlahRoda, int kecepatan, int jarak, int waktu)
   b. Dua *method override*
   c. Satu *method public*
4. Class Lambhorgini
   a. Terdapat satu variabel *private* yaitu brand (String).
   b. Satu *constructor* Lamborghini (int jumlahRoda, int kecepatan, int jarak, int waktu, String turbo, String brand)
   c. Tiga *method override*
5. Class Main
   a. Membuat objek dari keempat class diatas,
   b. Memberi value pada constructor
   c. Memanggil method yang sudah dibuat

Class Kendaraan :

```java
public class Kendaraan {
    protected int jumlahRoda, kecepatan, jarak, waktu; // attribut atau field

    // constructor
    public Kendaraan(int jumlahRoda, int kecepatan, int jarak, int waktu){
        this.jumlahRoda = jumlahRoda;
        this.kecepatan = kecepatan;
        this.jarak = jarak;
        this.waktu = waktu;
    }

    // method void Detail
    public void Detail(){
        System.out.println();
    }

    // method void Speed
    public void Speed(){
        System.out.println();
    }
}
```

Class Mobil :

```java
public class Mobil extends Kendaraan {
    protected String turbo; //  attribut atau field

    // constructor
    public Mobil(int jumlahRoda, int kecepatan, int jarak, int waktu, String
turbo){
        super(jumlahRoda, kecepatan, jarak, waktu);
        this.turbo = turbo;
    }

    // overriding method Detail dari class Kendaraan
    @Override
    public void Detail(){
        System.out.println("Ini adalah kendaraan beroda " + jumlahRoda + "
 namun jenisnya belum diketahui");
        System.out.println("Jarak yang bisa ditempuh sejauh " + jarak + "
km dengan waktu " + waktu + " ja ");
    }                       m

    // overriding method Speed dari class Kendaraan
    @Override
    public void Speed(){
        kecepatan = jarak * waktu;
        System.out.println("
Setelah dihitung, mobil ini memiliki kecepatan sebesar " + kecepatan + "km/jam
");
    }

    // method void Turbo
    public void Turbo(){
        System.out.println("Mobil ini belum memiliki turbo");
    }
}
```

Class Motor

```java
public class Motor extends Kendaraan {
    // tidak ada atribut atau field tambahan

    // contructor
    public Motor(int jumlahRoda, int kecepatan, int jarak, int waktu){
        super(jumlahRoda, kecepatan, jarak, waktu);
    }

    // overriding method Detail dari class Kendaraan
    @Override
    public void Detail(){
        System.out.println("Ini adalah kendaraan beroda " + jumlahRoda + "
 namun jenisnya belum diketahui");
        System.out.println("Jarak yang bisa ditempuh sejauh " + jarak + "
km dengan waktu " + waktu + " ja ");
    }                       m

    // overriding method Speed dari class Kendaraan
    @Override
    public void Speed(){
        kecepatan = jarak * waktu;
        System.out.println("
Setelah dihitung, mobil ini memiliki kecepatan sebesar " + kecepatan + "km/jam
");
    }

    // method void tambahSpeed
    public void tambahSpeed(int speedTambahan){
        kecepatan += speedTambahan;
        System.out.println("
 Setelah kecepatannya ditambah, kecepatannya menjadi " + kecepatan);
    }
}
```

Class Lambhorgini

```java
public class Lambhorgini extends Mobil {
    private String brand; // attribut atau field

    // constructor
    public Lambhorgini(int jumlahRoda, int kecepatan, int jarak, int waktu,
String turbo, String brand){
        super(jumlahRoda, kecepatan, jarak, waktu, turbo);
        this.brand = brand;
    }

    // overriding method Detail dari class Mobil
    @Override
    public void Detail(){
        System.out.println("Ini adalah kendaraan beroda " + jumlahRoda + "
 jenis " + "" + brand);
        System.out.println("Jarak yang bisa ditempuh sejauh " + jarak + "
km dengan waktu " + waktu + " ja ");
    }                       m

    // overriding method Speed dari class Kendaraan
    @Override
    public void Speed(){
        kecepatan = jarak * waktu;
        System.out.println("
Setelah dihitung, mobil ini memiliki kecepatan sebesar " + kecepatan + "km/jam
");
    }

    // method void Turbo
    public void Turbo(){
        System.out.println("Mobil ini belum memiliki turbo");
    }
}
```

Main Class :

```java
public class Main {
    // main class
    public static void main(String[] args) {
        System.out.println("Class Mobil");
        // membuat objek mobilsatu dari class Mobil
        Mobil mobilsatu = new Mobil(4, 0, 60, 2, "");
        mobilsatu.Detail();
        mobilsatu.Speed();
        mobilsatu.Turbo();
        System.out.println("");

        System.out.println("Class Motor");
        // membuat objek motorsatu dari class Motor
        Motor motorsatu = new Motor(2, 0, 30, 1);
        motorsatu.Detail();
        motorsatu.Speed();
        motorsatu.tambahSpeed(25);
        System.out.println("");

        System.out.println("Class Lambhorgini");
        // membuat objek lambo dari class Lambhorgini
        Lambhorgini lambo = new Lambhorgini(4, 0, 100, 1, "", "Aventador");
        lambo.Detail();
        lambo.Speed();
        lambo.Turbo();

    }
}
```

Output :

```
Class Mobil
Ini adalah kendaraan beroda 4 namun jenisnya belum diketahui
Jarak yang bisa ditempuh sejauh 60km dengan waktu 2 jam
Setelah dihitung, mobil ini memiliki kecepatan sebesar 120km/jam
Mobil ini belum memiliki turbo

Class Motor
Ini adalah kendaraan beroda 2 namun jenisnya belum diketahui
Jarak yang bisa ditempuh sejauh 30km dengan waktu 1 jam
Setelah dihitung, mobil ini memiliki kecepatan sebesar 30km/jam
Setelah kecepatannya ditambah, kecepatannya menjadi 55

Class Lambhorgini
Ini adalah kendaraan beroda 4 jenis Aventador
Jarak yang bisa ditempuh sejauh 100km dengan waktu 1 jam
Setelah dihitung, mobil ini memiliki kecepatan sebesar 100km/jam
Mobil ini belum memiliki turbo
```

Contoh dari penggunaan method overriding lainnya

```java
class Hewan {
    public void move() {
        System.out.println("Setiap hewan bisa bergerak");
    }
}

class Anjing extends Hewan {
    @Override
    public void move(){
        super.move();
        System.out.println("Anjing bergerak menggunakan kakinya")
;
    }
}

public class Test {
    public static void main(String[] args) {
        Anjing anjing = new Anjing();
        anjing.move();
    }
}

```

Output :

```
Setiap hewan bisa bergerak
Anjing bergerak menggunakan kakinya
```

Overloading

Compile time polymorphism adalah sebuah proses di mana sebuah *method* atau fungsi dipanggil saat kompilasi. Ini dapat terjadi karena sebuah konsep bernama method *overloading*.

Method overloading artinya *method* dengan nama yang sama, namun memiliki parameter yang berbeda, dan *method* ini berada dalam sebuah *class* yang sama atau bisa juga berada dalam *class* yang lain yang terkait dalam hirarki *inheritance*.

Adapun cara kerja dari *method overloading,* antara lain :

1. *Method overloading* harus memiliki nama yang sama seperti *parent class* nya.
2. *Method overloading* harus memiliki parameter yang berbeda dengan *parent class* nya.
3. Dapat dilakukan dengan mengubah jumlah, urutan dan atau tipe data parameternya.

Berikut apabila *method overloading* dapat diterima :

- **Kuantitas Parameter** : *Method Overloading* dapat diterima di dalam class mengingat jumlah/kuantitas parameter tidak sama.

Contoh:

```
public void luas(double p, double l){
        System.out.println(p * l);
    }

    public void luas(double p, double l, double t){
        System.out.println(p * l * t);
    }
```

- **Tipe Data Parameter** : *Method Overloading* dapat diterima di dalam *class* setidaknya satu dari setiap *method overloading* memiliki tipe data yang berbeda.

Contoh:

```
public void luas(double p, double l){
        System.out.println(p * l);
    }

    public void luas(int p, int l){
        System.out.println(p * l);
    }
```

- **Urutan Parameter** : *Method Overloading* dapat diterima di dalam sebuah class ketika urutan variabel tipe data tidak sama.

Contoh :

```
public void luas(double p, double l){
    System.out.println(p * l);
}

public void luas(int l, int p){
    System.out.println(p * l);
```

- **Return type** : Method overloading dapat diterima walaupun *return type* nya berbeda.

Contoh :

```
public void luas(double p, double l){
        System.out.println(p * l);
    }

    public int luas(int l, int p){
        return p * l;
```

Contoh dari penggunaan method overloading :

```java
public class App {
    public double luas(double p, double l){
        return p * l;
    }

    public int luas(int l, int p){
        return p * l;
    }

    public double luas(long p, double l){
        return p * l;
    }

    public long luas(long p, int l){
        return p * l;
    }

    public long luas(long p, int l, int t){
        return p * l * t;
    }
}
```

```java
public class Main extends App{
    public static void main(String[] args) {
        long panjang = 10;
        int lebar = 10;
        int tinggi = 10;

        App app = new App();
        System.out.println("Luas 1 yaitu : "+app.luas(panjang, lebar));
        System.out.println("Luas 2 yaitu : "+app.luas(panjang, lebar, tinggi));

    }
}
```

```
[Running] cd "d:\[Projects]\Portfolio\Portfolio Website\" && javac Main.java && java Main
Luas 1 yaitu : 100
Luas 2 yaitu : 1000
```

Dalam kode diatas dapat memanggil metode 'luas' dengan berbagai kombinasi tipe data dan jumlah parameter, dan Java akan memilih metode yang sesuai untuk dieksekusi berdasarkan tipe dan jumlah parameter yang diberikan. Dengan cara ini maka dapat menggunakan metode yang sama untuk berbagai kasus penggunaan.

Keuntungan Menggunakan *Overloading Method*:

- Dapat membuat banyak *method* dengan nama sama.
- Apabila tidak menggunakan *overloading method*, maka harus dibuatkan banyak method. Contoh : maks1 dengan 2 parameter, maks 2 untuk 3 parameter.
- *Overloading* juga dapat digunakan pada *constructor.*

Daftar Pustaka

[1] A. (2021, September 12). Tutorial Java 17 : Mengenal Overriding Java dan Superclass Subclass | CODEKEY. CODEKEY |. Retrieved December 23, 2021, from https://codekey.id/java/overriding-java

[2] Indonesia, D. (2021). Dicoding Indonesia. Dicoding. Retrieved December 23, 2021, from https://www.dicoding.com/academies/169/tutorials/7788

[3] Modul 3 Polymorphisme, Overloading, dan Overriding. (2019). EAD Laboratory.

[4] Muhardian, A. (2021, December 22). Belajar Java OOP: Memahami Inheritance dan Method Overriding. Petani Kode. Retrieved December 23, 2021, from https://www.petanikode.com/java-oop-inheritance

[5] Tutorial Java OOP Bahasa Indonesia - Programmer Zaman Now, from https://www.youtube.com/watch?v=f3ZhNnvtV-w&list=PL-CtdCApEFH-p_Q2GyK4K3ORoAT0Yt7CX&index=3

“if you don't walk today, you will have to run tomorrow”

find us on :

@eadlaboratory

@ozc7189g