VERSION 1.6

20 Februari 2025

PEMROGRAMAN BERORIENTASI OBJEK

MODUL 2 – OOP (CLASS, OBJECT, METHOD, ATTRIBUTE)

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 dasar Pemrograman Berorientasi Objek (OOP) dalam Java.
2. Mahasiswa memahami cara mendefinisikan dan menggunakan class, object, method, dan attribute dalam program Java.

TARGET MODUL

1. Mahasiswa dapat membuat program sederhana yang menerapkan class, object, method, dan attribute dalam Java.

PERSIAPAN

1. Device (Laptop/PC)
2. IDE (IntelliJ)

KEYWORDS

OOP Java, Class, Methods, Object, Attribute

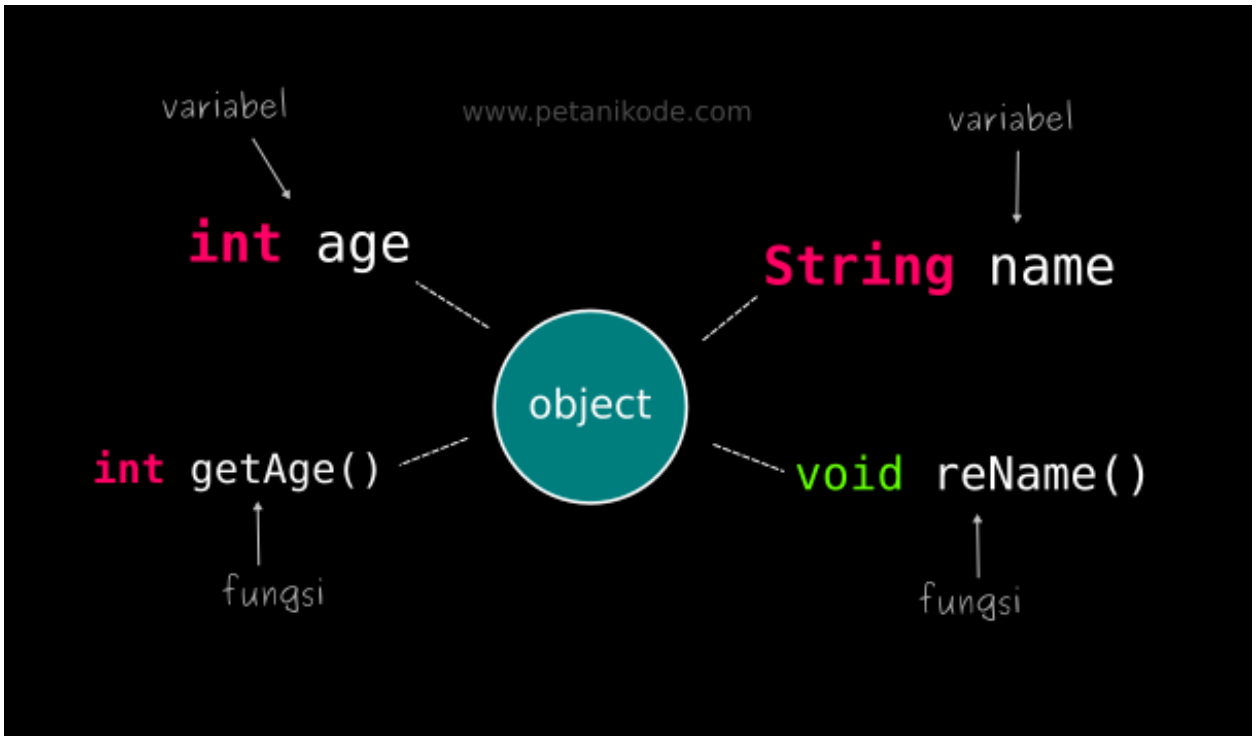
TABLE OF CONTENTS

OOP (Object Oriented Programming)

TEORI

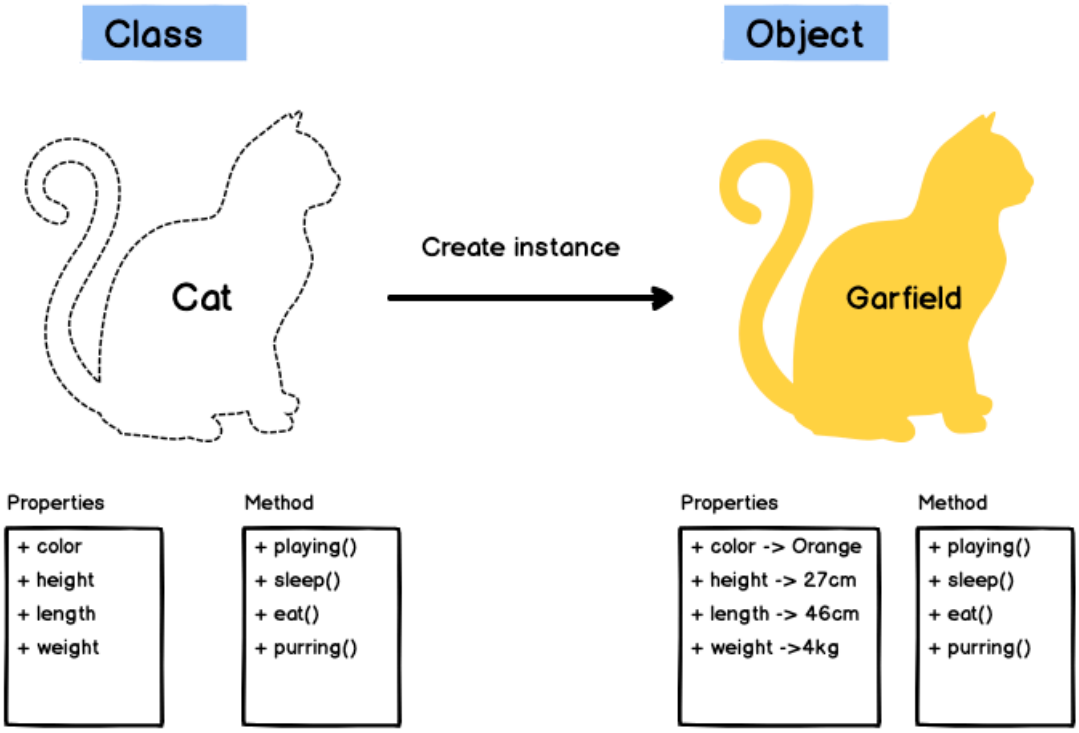
Konsep OOP (Object Oriented Programming) / PBO (Pemrograman Berorientasikan Objek)

Pemrograman Berorientasi Objek (OOP) adalah paradigma pemrograman yang berfokus pada **objek**. **Objek** adalah entitas yang menggabungkan **data** dan **fungsi** yang terkait. **OOP** membantu kita membuat program yang lebih **terstruktur**, **mudah dipahami**, dan **mudah** diubah. Program-program yang telah ada merupakan gabungan dari beberapa komponen-komponen kecil yang sudah ada sebelumnya. Hal itu dapat mempermudah pekerjaan seorang programmer dalam melakukan pengembangan program. **Objek-objek** yang saling berkaitan dan disusun kedalam satu kelompok ini disebut dengan **class**. Nantinya, objek-objek tersebut akan saling berinteraksi untuk menyelesaikan masalah program yang rumit.



MATERI

Class



Class adalah **"blueprint"** atau **"cetakan biru"** untuk menciptakan suatu **object** yang mendefinisikan **struktur** dan **perilaku** suatu **objek** dalam ***object oriented programming* (OOP)**. **Class** sebenarnya bertugas untuk mengumpulkan **fungsi/method** dan **variabel** dalam suatu tempat. Dalam contoh di sini kita ibaratkan **class** dengan sebuah **mobil**, yang artinya **Mobil** bisa memiliki sebuah variabel **warna**, **nama**, **kecepatan** dan juga bisa memiliki sebuah **method** untuk melakukan sesuatu seperti **bergerak maju**, **mundur**, dan **berbelok**. Contoh jika kita membuat sebuah **class Mobil**:

```
//class mobil
public class Mobil {

}
```

Class sebenarnya berisi **variabel** yang disebut sebagai **atribut** atau **properti** dan **fungsi** yang disebut sebagai **method**. Sebuah **class** berbeda dengan sebuah **fungsi**, **class** tidak memerlukan sebuah **()** tetapi langsung pada **body class** yang dimulai dengan **”{”** dan diakhiri dengan **”}”**.

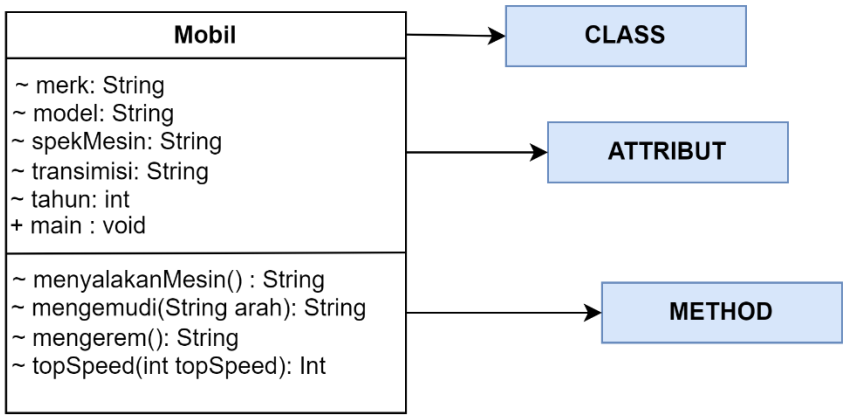
- Diagram class

  Class dapat diibaratkan sebagai cetakan untuk membuat objek. Cetakan ini mendefinisikan karakteristik dan kemampuan yang dimiliki oleh semua objek yang dibuatnya. Class diagram adalah sebuah gambar yang menunjukkan struktur dan hubungan antar class. Ibarat denah rumah, diagram ini membantu kita memahami bagaimana berbagai bagian sistem saling terhubung.

  Bagian-bagian penting class diagram:

  1. Nama (dan stereotype) : Identitas dan jenis class
  2. Atribut : karakteristik yang dimiliki oleh class
  3. Method : kemampuan yang dimiliki oleh class

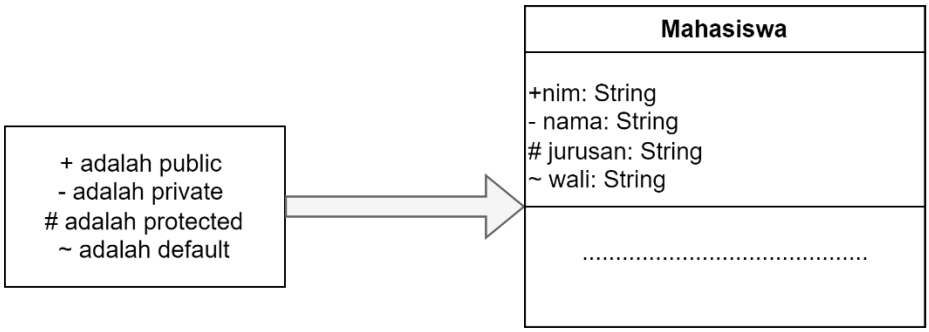
  Contoh:



  Penjelasan:

  - Tipe data setelah nama method adalah tipe pengembalian (return type) dari function/method
  - Access modifier. Fungsi dari akses modifier pada java adalah untuk membatasi scope dari sebuah class, constructor , atribut, method, atau anggota data lain yang terdapat dalam sebuah class Java. Selengkapnya tentang modifier akan dijelaskan di modul selanjutnya.

Contoh:



- Hubungan antar class

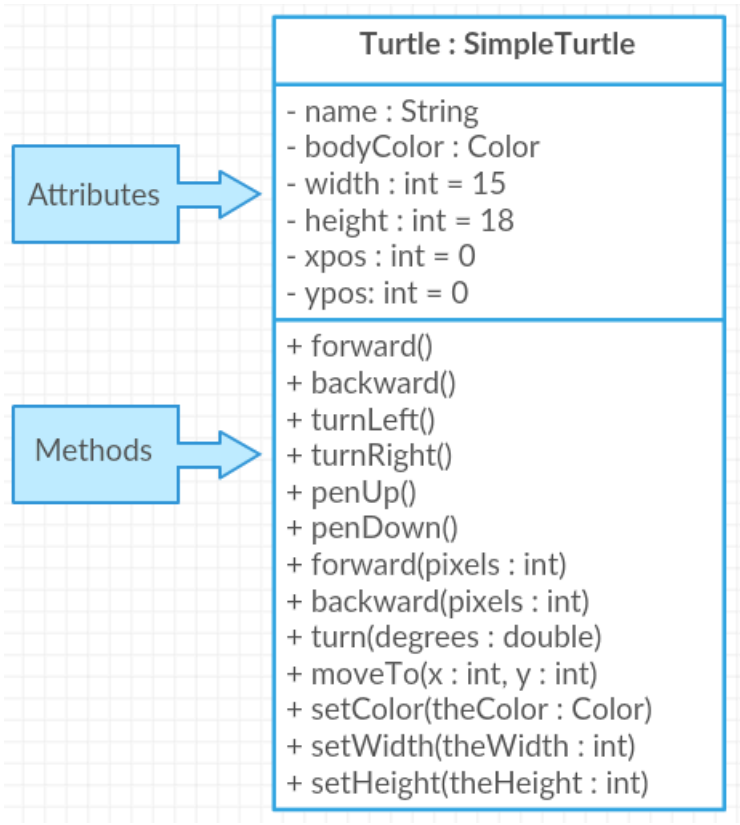
  Asosiasi, yaitu hubungan statis antar class yang biasanya menggambarkan class yang memiliki atribut berupa class lain. Terdapat beberapa jenis asosiasi, seperti:

  1. Asosisasi Sederhana: Bentuk asosiasi ——— sederhana ( )
  2. Agregasi yaitu hubungan yang menyatakan bagian, biasanya hubungan data master dan detailnya. Misal satu pembelian terdiri dari sejumlah item barang (◆———).
  3. Navigability : menunjukan arah query/komunikasi antar objek, bisa satu atau dua arah, terlihat pada tanda panahnya (——→)
  4. Campuran / Composit : campuran asosiasi (◆——→)

  Manfaat dari class diagram:

  1. Memahami struktur sistem: Class diagram membantu kita melihat gambaran besar sistem dan bagaimana berbagai bagiannya saling terhubung.
  2. Meningkatkan komunikasi: Class diagram membantu tim developer untuk berkomunikasi dengan lebih efektif tentang desain sistem.
  3. Mempermudah pengembangan: Class diagram membantu developer untuk membuat kode yang lebih terstruktur dan mudah dipahami

Attribute



Atribut atau **Properties** merupakan **identitas data** atau **informasi** dari **object class** yang sudah kita buat, atau di bahasa **C** sebelumnya disebut sebagai **variabel**. Ada beberapa hal yang terkait dengan **atribut** yaitu:

1. **Atribut** adalah **data** atau **properti** yang dimiliki oleh sebuah **class**
2. **Atribut** dapat berupa **variabel** seperti **nama**, **usia**, **warna**, dan lain sebagainya
3. **Atribut** mendefinisikan keadaan suatu **objek**

Misalnya pada suatu **mobil** memiliki spesifikasi seperti **merk**, **tahun**, **kecepatan**, **warna**. Maka kita bisa menulis pada **class Mobil** seperti ini:

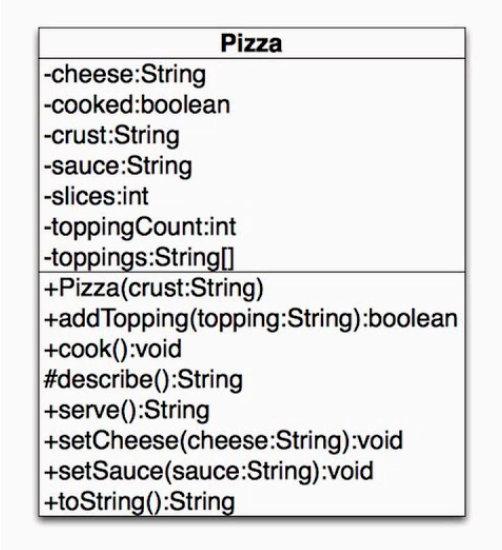
```java
//class mobil
public class Mobil {
    //attribute mobil
    String merk;
    String model;
    int tahun;
    String spekMesin;
    String transmisi;
}
```

Atau bisa juga dengan cara seperti gambar di bawah, hal ini juga bisa dilakukan untuk mengurangi pengulangan kode:

```java
//class mobil
public class Mobil {
    //attribute mobil
    String merk, model, spekMesin, transmisi;
    int tahun;
}
```

Method

Dengan melihat ilustrasi pada sub bab sebelumnya tentang attributes, bisakah anda sebutkan yang mana yang merupakan method/behaviour dari gambar dibawah ini?



Method atau bisa juga disebut sebagai **behavior** adalah tindakan atau perilaku yang dapat dilakukan oleh sebuah **objek** yang menjadi **instance** dari sebuah **class**. **Method**

merepresentasikan cara kerja atau bagaimana suatu **objek** dapat berjalan atau beroperasi. Contoh:

```java
//class mobil
public class Mobil {
    //attribute mobil
    String merk;
    String model;
    int tahun;
    String spekMesin;
    String transmisi;

    //method menyalakanMesin
    public void menyalakanMesin() {
        System.out.println("Mesin mobil Menyala");
    }

    //method mengemudi
    public String mengemudi(String arah) {
        return "Mobil bergerak ke arah " + arah;
    }
```

```java
    //method mengerem
    public String mengerem() {
        return "Berhenti";
    }

    //method topSpeed
    public int topSpeed(int topSpeed) {
        return topSpeed;
    }
}
```

Sebenarnya **method** sama dengan **fungsi** (di bahasa C) yang dapat melakukan sesuatu. Untuk contoh kode di atas **method** adalah sesuatu yang dapat dilakukan oleh **instance objek** mobil nantinya, berikut penjelasannya:

- **menyalakanMesin()** : adalah sebuah method yang dimana membuat sebuah instance nantinya bisa menyalakan mesin
- **mengemudi()** : menggerakkan mobil ke depan, belakang, dan juga berbelok

- **mengerem()** : menghentikan mobil

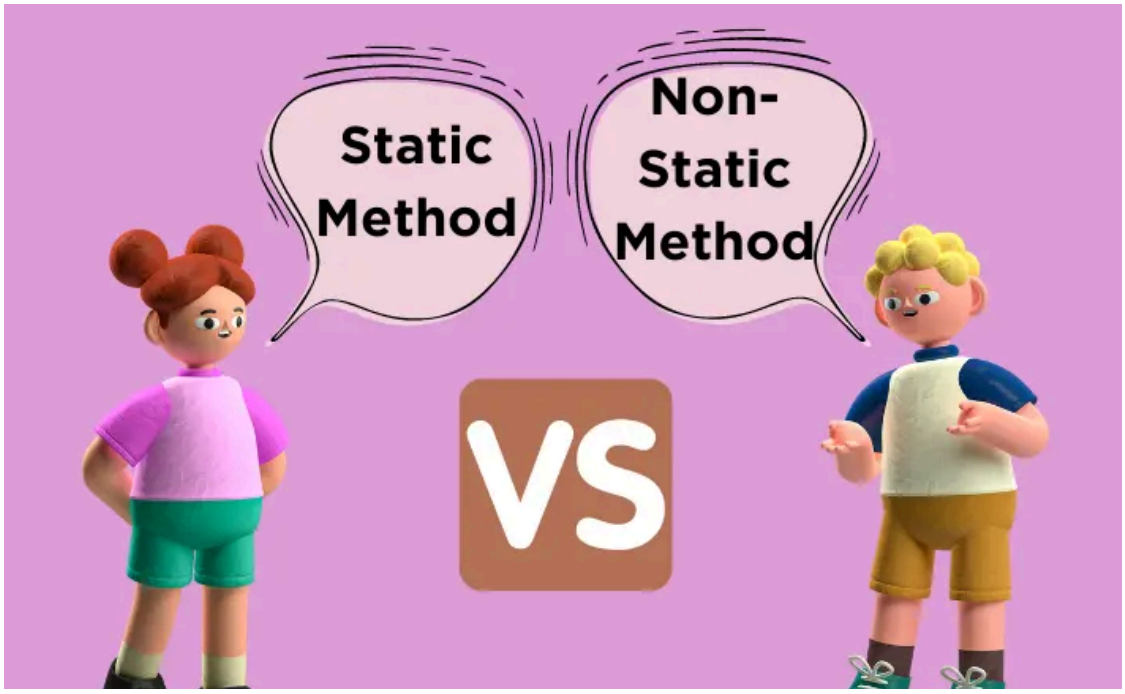
Karakteristik method:

- **Nama** : nama method sama dengan nama fungsi yang bisa digunakan untuk melakukan sesuatu. Dalam contoh di sini adalah sebuah mobil bisa melakukan **mengemudi()**, **mengerem(**), dan **menyalakanMesin()**.
- **Parameter** : **parameter** adalah sebuah **data** yang dilakukan **input** ketika sebuah **method** itu dipanggil. Dalam contoh di sini terdapat pada method **mengemudi(String arah)**, yang dimana ***String arah*** adalah sebuah **parameter**.
- **Return type**: **return type** menentukan **data** yang dihasilakn oleh sebuah **method**. Dalam contoh di sini adalah method **topSpeed(int topSpeed)**, yang dimana **method** ini me-*return* sebuah **nilai integer**.
- **Body**: body berisi kode yang akan dijalankan oleh sebuah method. Body dari sebuah method diawali dengan "{" dan diakhiri dengan "}".

Manfaat menggunakan method:

- Meningkatkan reusability kode. Contoh fungsi mengemudi() yang dapat digunakan berulang kali untuk menggerakkan mobil.
- Meningkatkan readability kode. Kode akan lebih terstruktur dengan baik sehingga membuat kode lebih mudah dibaca dan dipahami.
- Membantu membagi program menjadi bagian-bagian yang lebih kecil. Hal ini membuat program lebih mudah dikelola dan diubah.
- Meningkatkan maintainability. Program yang menggunakan method akan lebih mudah diperbaiki dan dipelihara pada waktu yang mendatang.

- **Static dan Non-static method**



Static method adalah method yang dapat dieksekusi atau dipanggil tanpa harus membuat sebuah instance objek. Sedangkan method **non**-**static** adalah method yang harus membuat **instance** objek untuk menggunakan atau memanggilnya, di mana method ini yang akan berkaitan erat dengan konsep OOP.

Cara mendeklarasikan sebuah method static di Java adalah dengan menambahkan keyword static. Contohnya seperti ini:

```
static returnType namaMethod(){
    // body dari method
}
```

Contoh implementasi dari static method kita akan coba ubah method mengerem dengan static menjadi seperti ini:

```
static String mengerem() {
    return "Berhenti";
}
```

Untuk cara pemanggilannya seperti ini pada main method:

```
public static void main(String[] args) {
    // tidak perlu membuat instance objek mobil
    System.out.println(mengerem());
}
```

Pada kode di atas kita bisa langsung memanggil atau menggunakan method mengerem() tanpa membuat sebuah instance objek dari class Mobil.

Untuk proses mendeklarasikan sebuah method static sebenarnya pada materi di atas sudah kita buat sebuah method non-static. Untuk pembuatan method static dideklarasikan tanpa menggunakan kata kunci *static* seperti ini:

```
returnType namaMethod(){
    // body method
}
```

Untuk implementasinya bisa cek lagi pada class Mobil yang sudah kita buat sebelumnya seperti ini:

```
public class Mobil{
    String merk;
    String model;


    void menyalakanMesin(){
        System.out.println("Mesin mobil menyala");
    }


    String mengemudi(String arah){
        return "Mobil bergerak ke arah" + arah;
    }
```

```
    String mengerem(){
        return "Berhenti";
    }


    int topSpeed(int topSpeed){
        return topSpeed;
    }
}
```

Pada kode class Mobil di atas, semua method yang ada bentuknya adalah non-static. Hal yang utama dari method non-static adalah pemanggilan menggunakan ‘.’ setelah nama objek.

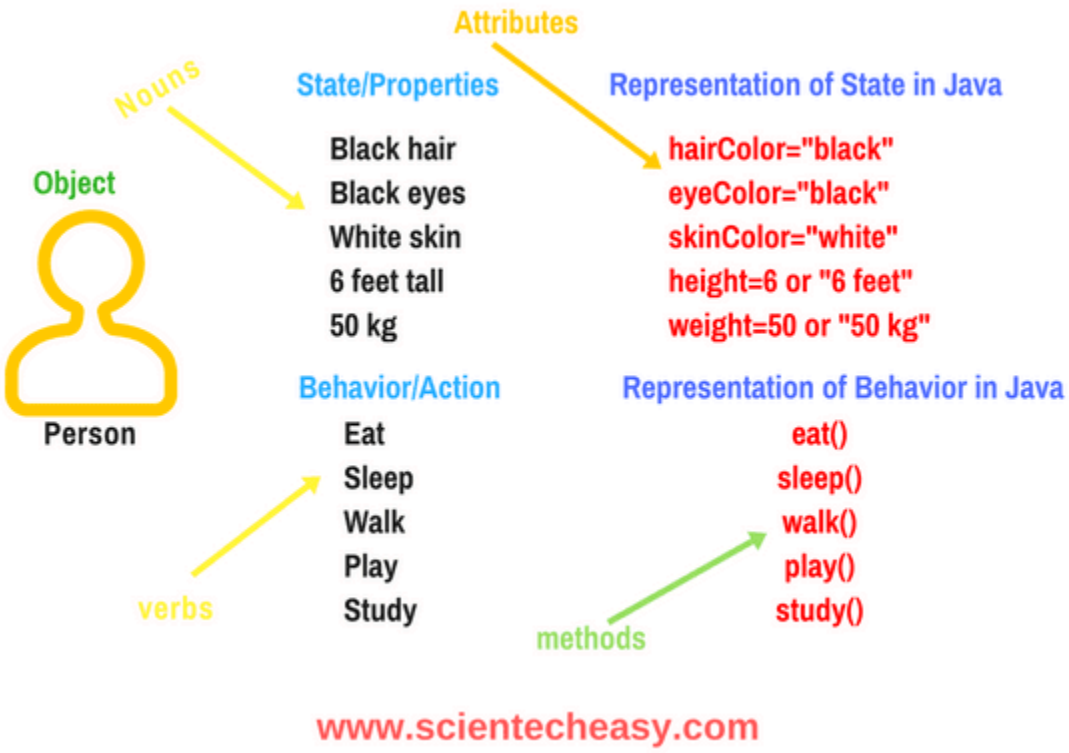
Catatan untuk static method, bisa dipanggil langsung dengan nama method selama masih di lingkup class tersebut (satu file) dan jika dari class lain kita bisa memanggil dengan nama kelas lalu nama method. Contohnya:

```
class classKedua {
    static void callStaticMethod() {
        System.out.println("Hola Ini Adalah class Kedua Dengan Static method");
    }

    void callNonStaticMethod() {
        System.out.println("Hola Ini Adalah class Kedua Dengan Non-Static method");
    }
}

public class classPertama {
    public static void main(String[] args) {
        // static method
        classKedua.callStaticMethod();

        classKedua objectClassPertama = new classKedua();
        // non-static method
        objectClassPertama.callNonStaticMethod();
    }
}
```

Output program:

```
Hola Ini Adalah class Kedua Dengan Static method
Hola Ini Adalah class Kedua Dengan Non-Static method

Process finished with exit code 0
```

Object



Object atau **objek** adalah hasil dari **deklarasi** atau **instance** dari sebuah **class**. Dalam **objek**, kita dapat mengakses dan memanipulasi isi dari sebuah **class** yang sudah kita buat sebelumnya. Sebagai contoh jika kita memiliki **class Mobil** sebelumnya, maka kita bisa membuat sebuah **objek** dari **class** tersebut di dalam **method main** seperti ini:

```java
public class Mobil{
    String merk;
    String model;
    String spekMesin;
    int tahun;

    void menyalakanMesin(){
        System.out.println("Mesin mobil menyala");
    }

    String mengemudi(String arah){
        return "Mobil bergerak ke arah" + arah;
    }

    String mengerem(){
        return "Berhenti";
    }

    int topSpeed(int topSpeed){
        return topSpeed;
    }

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

Ketika kita jalankan programnya maka tidak akan ada sesuatu yang terjadi, karena kita hanya membuat sebuah **instance objek** saja dari **class Mobil** dengan nama **objek mobil**. Setelah kita membuat sebuah **objek**, kita bisa memanggil **method** yang sudah dibuat dengan cara seperti ini pada **main method**:

```java
public static void main(String[] args) {
    Mobil mobil = new Mobil();
    mobil.menyalakanMesin();
}
```

Maka **output** program akan seperti ini:

```
Mesin mobil menyala

Process finished with exit code 0
```

Selain itu, jika kita mengubah isi dari **atribut** pada **class Mobil** seperti ini:

```
mobil.merk = "Honda";
mobil.model = "X100";
```

Maka kita juga bisa memanggil isi dari masing-masing **atribut** dengan cara seperti ini:

```
System.out.println(mobil.merk);
System.out.println(mobil.model);
```

Output dari program ketika dijalankan:

```
Mesin mobil menyala
Honda
X100

Process finished with exit code 0
```

Apakah ketika kita membuat sebuah **class**, kita hanya bisa membuat **1 instance objek**? Jawabannya **tidak**. Kita bisa membuat sebuah **instance** objek sebanyak yang kita butuhkan, contohnya disini kita akan coba untuk membuat **3 instance objek** dari **class Mobil**, maka bisa kita tulis kodenya seperti ini pada **main method**:

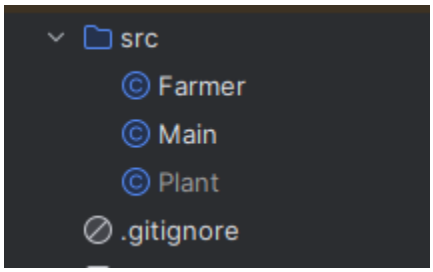
```
public static void main(String[] args) {
    Mobil mobil1 = new Mobil();
    Mobil mobil2 = new Mobil();
    Mobil mobil3 = new Mobil();
}
```

Masing-masing **instance objek** terpisah satu sama lain dan **berbeda**, maksudnya adalah **objek mobil1** berdiri sendiri begitupun dengan **objek mobil2** dan **mobil3**. Untuk cara memanggil **method** dan **atribut** pada masing-masing **objek** ialah sama dengan cara sebelumnya.

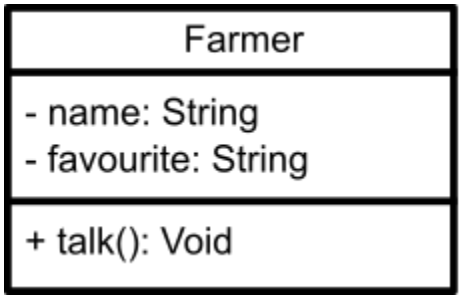
```
mobil1.merk = "Honda";
mobil2.merk = "Toyota";

System.out.println("Merk mobil1: " + mobil1.merk);
System.out.println("Merk mobil2: " + mobil2.merk);
```

PRAKTEK

Mari kita melakukan percobaan dengan melanjutkan praktek kita di modul 1 kemarin! Silahkan buka file project dari praktek modul 1 sebelumnya yang sudah kita buat. Lalu silahkan buat class baru dengan nama Plant dan Farmer. Lupa cara buat class? Silahkan kembali ke modul 1 dan baca lagi 😀. Jika sudah pasti akan terlihat seperti ini:



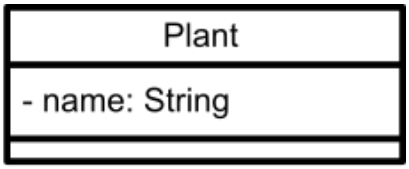
Class Farmer memiliki spesifikasi sebagai berikut:



Atau dapat kalian lihat disini:

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

Sementara class Plant memiliki spesifikasi sebagai berikut:



Atau seperti ini:

```
4 usages
public class Plant {
    4 usages
    String name;
}
```

Lalu pada class main silahkan buat 2 object dari class Farmer dan Plant seperti ini:

```
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();

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

Lalu tetapkan nama pada ke-empat objek itu seperti ini:

```
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";

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

Jangan lupa menambahkan nilai pada variabel Favourite di kedua objek Farmer seperti ini:

```
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());
    }
}
```

Apa lagi yang kurang ya? Oh iya, method pada class Farmer belum dipanggil. Silahkan panggil method pada kedua object dari class Farmer tersebut seperti ini.

```
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();
    }
}
```

Lalu jalankan programnya

```
Hello, World!
Current date and time: Thu Feb 20 19:29:55 WIB 2025
Hi! My name is: Crazy Dave. My favourite plant is: Sunflower
Hi! My name is: Sober Dave. My favourite plant is: Mushroom

Process finished with exit code 0
```

TIPS

Video singkat pengantar PBO:

Video

Video singkat konsep class dan object

Video

Video singkat property dan method:

Video

CODELAB & TUGAS

CODELAB 1

Buatlah sebuah program dalam bahasa Java yang terdiri dari dua kelas, yaitu **Hewan** dan **Main**.

1. Kelas **Hewan** harus memiliki tiga atribut dengan tipe data **String**, yaitu:
    - Nama
    - Jenis
    - Suara
2. Kelas **Hewan** juga harus memiliki sebuah metode **tampilkanInfo()** yang mencetak informasi hewan seperti **Nama**, **Jenis**, dan **Suara**
3. Kelas **Main** harus memiliki metode **main(String[] args)**, di mana:
    - Dibuat dua objek **Hewan**, yaitu **hewan1** dan **hewan2**.
    - hewan1 memiliki atribut:
      Nama = "Kucing"

      Jenis = "Mamalia"

Suara = "Nyann~~"

- hewan2 memiliki atribut:
  nama = "Anjing"

  Jenis = "Mamalia"

  Suara = "Woof-Woof!!"
- Memanggil metode **tampilkanInfo()** dari kedua objek tersebut.

4. Contoh **output** yang diharapkan:

```
Nama: Kucing
Jenis: Mamalia
Suara: Nyann~~

Nama: Anjing
Jenis: Mamalia
Suara: Woof-Woof!!


Process finished with exit code 0
```

CODELAB 2

Buatlah sebuah program dalam bahasa Java yang terdiri dari dua kelas, yaitu **RekeningBank** dan **Main**.

Ketentuan Kelas **RekeningBank**:

1. Tiga **atribut** sebagai berikut:
   - nomorRekening **(tipe String)** → Nomor rekening nasabah
   - namaPemilik **(tipe String)** → Nama pemilik rekening
   - saldo **(tipe double)** → Saldo dalam rekening
2. Tiga **method** sebagai berikut:
   - **tampilkanInfo()** → Menampilkan informasi **rekening**.
   - **setorUang(double jumlah)** → Menambahkan jumlah ke **saldo,** dan tampilkan informasi transaksi.
   - **tarikUang(double jumlah)** → Mengurangi jumlah dari saldo jika saldo mencukupi, dan tampilkan informasi transaksi.

Ketentuan Kelas **Main**

1. Kelas ini harus memiliki metode **main(String[] args)**
2. Buat dua objek **RekeningBank**, yaitu **rekening1** dan **rekening2**.
3. **rekening1** memiliki atribut:
   nomorRekening = "NIM Kalian"
   namaPemilik = "Nama kalian"
   saldo = “terserah, sesuai saldo ATM kalian jg gpp **(harus angka positif)**”
4. **rekening2** memiliki atribut:
   nomorRekening = "NIM teman kalian"
   namaPemilik = "Nama teman kalian"
   saldo = “terserah”
5. Lalu panggil metode pada kedua objek tersebut. Contoh **output** yang diharapkan:

```
Nomor Rekening: 202310370311129
Nama Pemilik: Maharani
Saldo: Rp500000.0

Nomor Rekening: 202310370311307
Nama Pemilik: Amelia
Saldo: Rp1000000.0

Maharani menyetor Rp200000.0. Saldo sekarang: Rp700000.0
Amelia menyetor Rp500000.0. Saldo sekarang: Rp1500000.0

Maharani menarik Rp800000.0. (Gagal, Saldo tidak mencukupi) Saldo saat ini: Rp700000.0
Amelia menarik Rp300000.0. (Berhasil) Saldo sekarang: Rp1200000.0

Nomor Rekening: 202310370311129
Nama Pemilik: Maharani
Saldo: Rp700000.0

Nomor Rekening: 202310370311307
Nama Pemilik: Amelia
Saldo: Rp1200000.0

Process finished with exit code 0
```

TUGAS

Pada tugas ini, sistem **login** yang sebelumnya dikembangkan dalam satu file, kini diubah menjadi sistem berbasis objek dengan pemisahan kelas yang lebih baik.

Struktur Program

- **Kelas Admin**
    - Berisi atribut **username** dan **password** yang telah ditentukan pada modul sebelumnya.
    - Memiliki metode **login()** yang memverifikasi input pengguna dengan data admin yang valid.
- **Kelas Mahasiswa**
    - Berisi atribut **nama** dan **nim** yang telah ditentukan pada modul sebelumnya.
    - Memiliki metode **login()** yang memverifikasi input pengguna dengan data mahasiswa yang valid.
    - Memiliki metode **displayInfo(**) untuk menampilkan informasi mahasiswa setelah login berhasil.
- **Kelas LoginSystem (kelas utama)**
    - Memuat logika utama program, termasuk input dari pengguna.
    - Membuat objek **Admin** dan **Mahasiswa** untuk mengakses metode login masing-masing.
    - Menyediakan menu bagi pengguna untuk memilih **login** sebagai Admin atau Mahasiswa.
    - Menggunakan **metode** dari objek **Admin** dan **Mahasiswa** untuk memverifikasi login.

Cara Kerja Program

- Program meminta pengguna memilih jenis login: Admin atau Mahasiswa.
- Jika memilih Admin:
    - Program meminta input **username** dan **password**.
    - Jika sesuai dengan data admin, login berhasil.
    - Jika tidak sesuai, muncul pesan error.
- Jika memilih Mahasiswa:

    - Program meminta input **nama** dan **NIM**.
    - Jika sesuai dengan data mahasiswa, login berhasil dan informasi mahasiswa ditampilkan.
    - Jika tidak sesuai, muncul pesan error.
- Jika input pilihan tidak valid, program akan menampilkan pesan error.

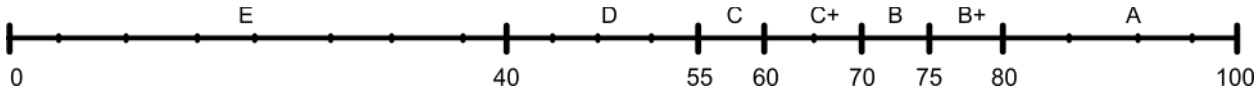
PENILAIAN

RUBRIK PENILAIAN

Aspek Penilaian	Poin
CODELAB 1	**Total 20%**
Kerapian kode	5%
Ketepatan kode & output	5%
Kekreativitasan kode	5%
Kode orisinal (tidak nyontek)	5%
CODELAB 2	**Total 20%**
Kerapian kode	5%
Ketepatan kode & output	5%
Kekreativitasan kode	5%
Kode orisinal (tidak nyontek)	5%
TUGAS	**Total 60%**
Kerapian kode	5%
Ketepatan kode & output	10%
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

Oke, bagaimana modul 2-nya? Masih terasa mudah? Sebenarnya, jika kalian perhatikan, tugas di modul 1 dan 2 menghasilkan output yang sama. Namun, ada yang berbeda. Kira-kira apa ya? Benar, source code-nya. Di modul 2 ini, kalian diminta untuk menerapkan konsep dari PBO.

Mungkin kalian akan bertanya: *"Bang untuk apa pakai cara yang susah kalau ada cara yang lebih mudah dan menghasilkan output yang sama?."* Well, pertanyaan kalian itu akan terjawab nanti di Modul 6.

Jangan khawatir jika kalian merasa konsep ini sulit dipelajari. Semakin banyak kalian berlatih, semakin terlihat polanya. Ibarat belajar naik sepeda, sebanyak apapun buku yang kalian baca tentang cara naik sepeda, akan percuma jika kalian tidak mulai mengayuh pedalnya.

Jika ada kebingungan jangan ragu untuk bertanya. Karena malu bertanya, tak jatuh jauh dari pohonya. Sekian dan terima gaji.