Alpro 2

Praktikum Day #5

Class dan Object

Lab Assistant :
Fikri & Jo

Overview

- Kenalan sama OOP
- Konsep Class dan Object
- Constructor dalam Class
- Access Modifier dan Method
- Static vs Instance Member
- Contoh kasus ft. UML Diagram

APA SIH OOP ITU?

Object Oriented Programming

Program yang terdiri dari objek-objek yang seringkali merepresentasikan apa yang ada di dunia nyata.

Konsep Class dan Object

1. **Class → "Blueprint/Cetakan"**

- **Class adalah rancangan atau template yang mendefinisikan atribut (data) dan perilaku (metode/fungsi) dari suatu objek.**
  → Seperti cetakan kue yang menentukan bentuk dan bahan kue, tapi belum jadi kue sungguhan.

```
class Mobil {
    // Atribut (data)
    String merk;
    String warna;

    // Metode (perilaku)
    void jalan() {
        System.out.println("Mobil " + merk + " sedang jalan!");
    }
}
```

Konsep Class dan Object

2. Object (Objek) → "Benda Nyata"

- Object adalah instance (perwujudan nyata) dari sebuah class.

  → Seperti kue yang sudah jadi dari cetakannya. Satu class bisa dipakai untuk membuat banyak objek.

```
Mobil avanza = new Mobil();  // Membuat objek 'avanza' dari class 'Mobil'
avanza.merk = "Toyota";      // Mengisi atribut
avanza.warna = "Hitam";
avanza.jalan();              // Memanggil metode
```

Output : Mobil Toyota sedang jalan!

Constructor dalam Class

Apa itu Constructor?

Constructor adalah metode khusus yang otomatis dipanggil saat sebuah objek dibuat (dengan keyword new). Fungsinya:

1. **Menginisialisasi nilai awal atribut objek.**
2. **Memastikan objek siap digunakan saat dibuat.**

Ciri-Ciri Constructor:

1. **Namanya harus sama persis dengan nama class.**
2. **Tidak memiliki return type (bahkan void juga tidak boleh).**
3. **Bisa memiliki parameter (constructor berparameter) atau tidak (default constructor).**

Contoh Constructor

1. Default Constructor (Tanpa Parameter)

```
class Mobil {
    String merk;
    String warna;

    // Constructor
    Mobil() {
        merk = "Toyota";  // Nilai default
        warna = "Hitam";
    }
}

// Saat membuat objek:
Mobil avanza = new Mobil();  // Constructor Mobil() otomatis dipanggil
System.out.println(avanza.merk);  // Output: "Toyota"
```

Contoh Constructor

2. Parameterized Constructor (Dengan Parameter)

```
class Mobil {
    String merk;
    String warna;

    // Constructor berparameter
    Mobil(String inputMerk, String inputWarna) {
        merk = inputMerk;
        warna = inputWarna;
    }
}

// Membuat objek dengan nilai custom:
Mobil avanza = new Mobil("Toyota", "Merah");
System.out.println(avanza.warna);  // Output: "Merah"
```

Kenapa Pakai Constructor?

1. **Efisiensi: Langsung set nilai awal saat objek dibuat, tanpa harus memanggil method terpisah.**
2. **Kontrol: Memastikan objek tidak bisa dibuat tanpa data wajib (misal: Mobil harus punya merk dan warna).**

Bahasa Bayi:

- **Constructor seperti tukang bangunan yang langsung menyiapkan rumah (objek) dengan pondasi (atribut) yang lengkap saat dibangun.**
  - **Tanpa constructor: Rumah kosong, harus diisi furniture manual.**
  - **Dengan constructor: Rumah sudah ada furniture dasar saat selesai dibangun.**

Catatan Penting:

- **Jika tidak buat constructor, Java akan otomatis membuat default constructor kosong (tanpa parameter).**
- **Constructor bisa lebih dari satu (overloading), contoh:**

```
Mobil() { ... }  // Constructor 1 (default)
Mobil(String merk) { ... }  // Constructor 2 (1 parameter)
```

Access Modifier (Pengubah Akses) dalam OOP

Access modifier adalah kata kunci yang menentukan tingkat akses dari class, atribut, atau method dalam Java. Tujuannya untuk mengontrol visibilitas dan keamanan data.

4 Jenis Access Modifier di Java

Modifier	Class	Package	Subclass (Warisan)	World (Luar Class)
`public`	✅	✅	✅	✅
`protected`	✅	✅	✅	❌
`default`	✅	✅	❌	❌
`private`	✅	❌	❌	❌

1. **Public (+) → Bisa Diakses di Mana Saja**
   - **Atribut/method bisa dipanggil dari class mana pun, bahkan dari package berbeda.**
   - **Class yang dideklarasikan public harus disimpan dalam file dengan nama yang sama.**

```
public class Mobil {
    public String merk;  // Bisa diakses di mana saja

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

// Di class lain:
Mobil toyota = new Mobil();
toyota.merk = "Toyota";  // Bisa diakses karena public
toyota.jalan();          // Output: "Mobil jalan!"
```

2. Protected (#) → Hanya untuk Class Turunan & Package yang Sama

- Bisa diakses oleh:
  - Class itu sendiri
  - Subclass (class turunan)
  - Class dalam package yang sama

```
public class Kendaraan {
    protected String merk;  // Hanya bisa diakses di package yang sama atau subclass
}

class Mobil extends Kendaraan {
    void setMerk(String merk) {
        this.merk = merk;  // Bisa diakses karena Mobil adalah subclass
    }
}
```

3. Default (Tidak Ditulis) → Hanya untuk Package yang Sama

- Jika tidak menulis modifier, maka secara otomatis menjadi default.
- Hanya bisa diakses dalam package yang sama.

```
class Mobil {
    String merk;  // Default (hanya bisa diakses dalam package yang sama)
}
```

4. Private (-) → Hanya untuk Class Itu Sendiri

- **Tidak bisa diakses di class lain, bahkan di package yang sama atau subclass.**
- **Biasanya digunakan untuk atribut yang sensitif (misal: password).**

```
public class User {
    private String password;  // Hanya bisa diakses di dalam class User

    // Method untuk mengakses private attribute (getter/setter)
    public void setPassword(String pass) {
        this.password = pass;
    }

    public String getPassword() {
        return "*****";  // Tidak mengembalikan password asli
    }
}

// Di class lain:
User user1 = new User();
user1.setPassword("12345");
System.out.println(user1.getPassword());  // Output: "*****"
// user1.password = "123";  // ERROR! Karena private
```

Kapan Menggunakan Access Modifier?

Modifier	Penggunaan Umum
`public`	Method/class yang ingin diakses secara global.
`protected`	Atribut/method yang hanya boleh diubah oleh subclass.
`default`	Class/atribut internal dalam satu package.
`private`	Data sensitif yang tidak boleh diakses langsung dari luar.

Kenapa Access Modifier Penting?

1. **Encapsulation (Pembungkusan Data)**: **Membatasi akses langsung ke data untuk menghindari perubahan yang tidak valid.**
2. **Keamanan**: **Misal, private mencegah manipulasi data sensitif.**
3. **Kode Lebih Terstruktur**: **Memisahkan bagian yang boleh diakses (public) dan yang tidak (private).**

Contoh Nyata:

- **private String saldo; di class Bank → Tentu kita mau agar data tidak bisa diubah sembarangan, sehingga untuk mengubah harus lewat method setSaldo().**
- **public void startEngine() di class Mobil → Bisa dipanggil di mana saja sehingga tidak perlu repot membuat method lagi tinggal panggil**

INSTANCE VS STATIC MEMBER ?

Instance Member → **atribut (variabel) atau method (fungsi) yang *terikat ke setiap objek yang dibuat dari class.***

Ciri-ciri :

- **Untuk menggunakan instance member, harus membuat object terlebih dahulu.**
- **Setiap object bisa punya nilai berbeda untuk setiap instance field.**

```java
package instancemember;

public class mahasiswa{
    String nama;
    long NRP;

    void info(){
        System.out.println("Nama mhs : "+nama);
        System.out.println("NRP : 500222"+NRP);
    }

    Run | Debug
    public static void main(String[] args) {
        mahasiswa mhs1 = new mahasiswa();
        mhs1.nama = "Fikri Ramadhan";
        mhs1.NRP = 1001;

        mhs1.info();

        mahasiswa mhs2 = new mahasiswa();
        mhs2.nama = "Aldi Hermawan";
        mhs2.NRP = 1000;

        mhs2.info();
    }
}
```

```
Nama mhs : Fikri Ramadhan
NRP : 5002221001
Nama mhs : Aldi Hermawan
NRP : 5002221000
```

Static Member → atribut (variabel) atau method (fungsi) yang ***terikat langsung ke class.***

Ciri-ciri :

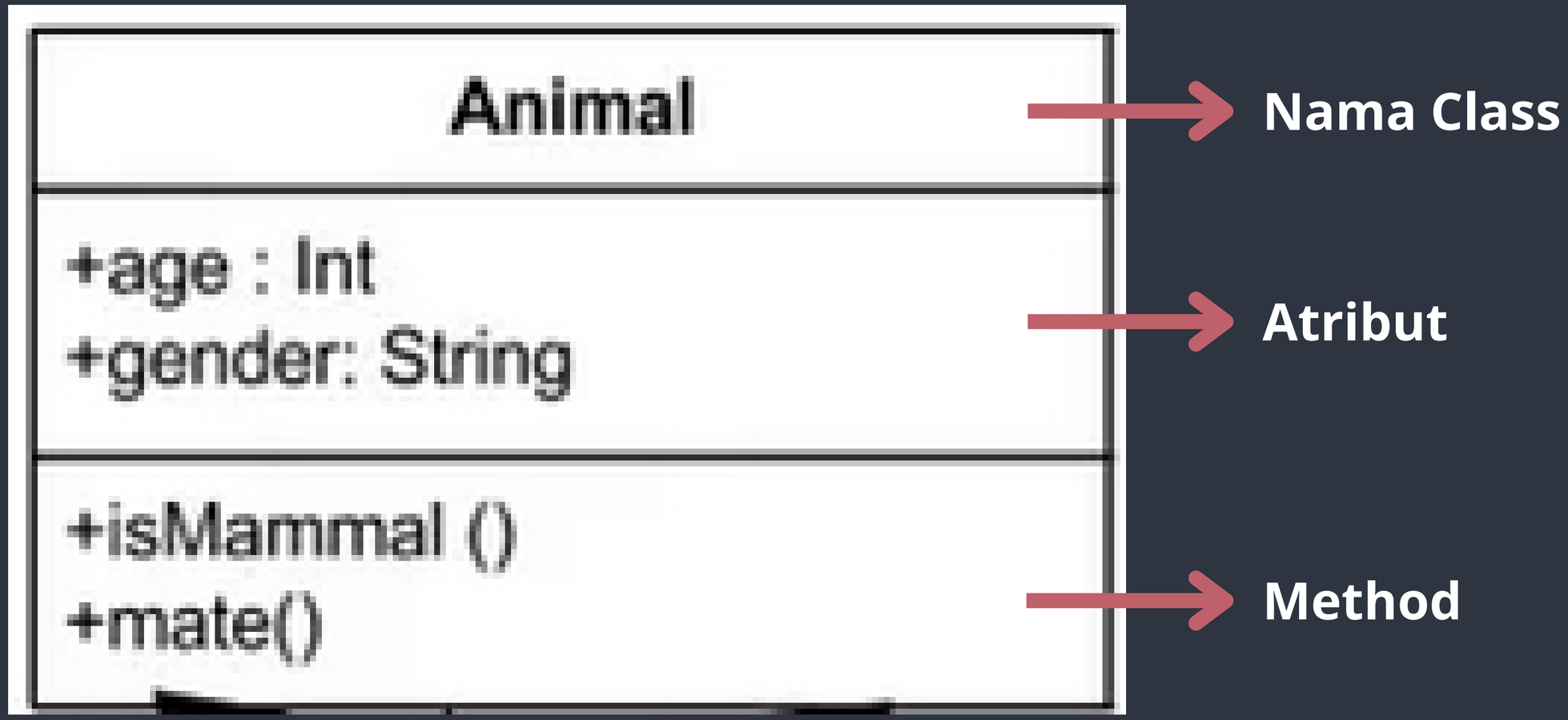
- **Tidak perlu membuat object untuk mengakses static member.**
- **Static member akan berbagi satu nilai yang sama antar semua object.**

```java
package member;

public class kalkulator {
    static int jumlahOperasi = 0; // static variable

    static int tambah(int a, int b) { // static method
        jumlahOperasi++;
        return a + b;
    }
    Run | Debug
    public static void main(String[] args) {
        int hasil = kalkulator.tambah(a:2, b:3);
        System.out.println(hasil);
    }
}
```

UML DIAGRAM

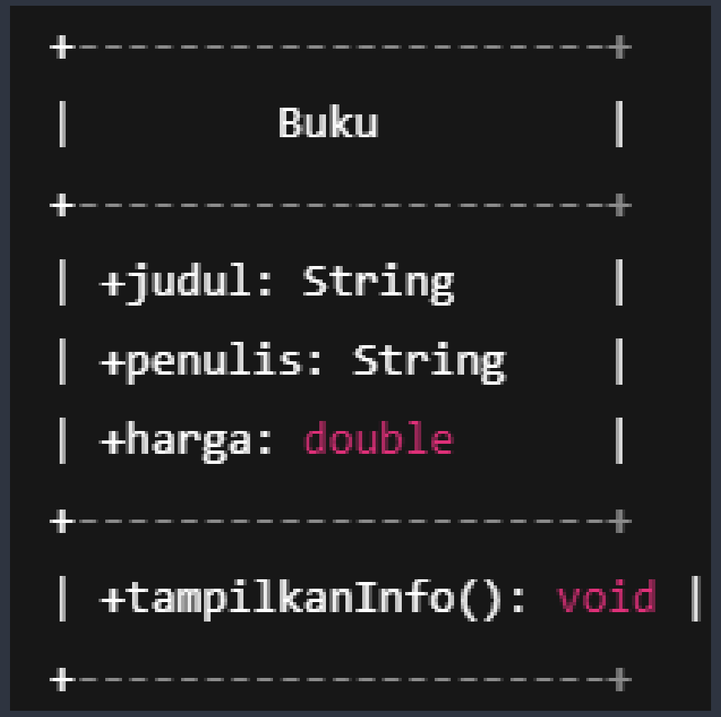

Animal
+age : Int
+gender: String
+isMammal ()
+mate()
Nama Class
Atribut
Method

BEST PRATICES !

Tips Membuat Class dan Object yang Baik:

- **Gunakan huruf kapital untuk nama class (Mobil, Mahasiswa).**
- **Gunakan huruf kecil untuk nama object (mobilSaya, mhs1).**
- **Selalu gunakan access modifier (private, public).**
- **Pisahkan tugas logika ke dalam method.**
- **Gunakan constructor untuk inisialisasi data.**

MINI QUIZ



Buatlah kode java untuk UML diagram disamping :