

## MODUL 5

### OBJECT ORIENTED PROGRAMMING JAVA II

#### 5.1 Cakupan Variabel (Variable Scope)

Cakupan variabel (variable scope) dalam Java mengacu pada di mana variabel tersebut dapat diakses dan digunakan dalam suatu program. Variabel dalam Java memiliki ruang lingkup tertentu tergantung pada di mana variabel tersebut dideklarasikan.

Jenis-jenis variable scope dalam java:

- Variabel Lokal (Local Variable)
- Variabel Instance (Instance Variable)
- Variabel Kelas / Statis (Class/Static Variable)
- Variabel Parameter (Parameter Variable)

##### a) Variable Lokal

Variabel yang dideklarasikan di dalam sebuah method, konstruktor, atau blok kode tertentu dan hanya bisa diakses dalam lingkup tersebut.

Ciri-ciri variabel lokal:

- Dideklarasikan di dalam method, konstruktor, atau blok tertentu.
- Tidak memiliki modifier akses (private, public, dll.).
- Hanya bisa digunakan di dalam blok tempat variabel tersebut dideklarasikan.
- Tidak memiliki nilai default, sehingga harus diinisialisasi sebelum digunakan.

Contoh:



```
public class Mahasiswa {
    public void tampilkanNama() {
        String nama = "Budi"; // Variabel lokal
        System.out.println("Nama: " + nama);
    }

    public static void main(String[] args) {
        ContohLocal obj = new ContohLocal();
        obj.tampilkanNama();

        // System.out.println(nama); // ERROR: 'nama' hanya bisa diakses dalam tampilkanNama()
    }
}
```

- Variabel nama hanya bisa digunakan di dalam method tampilkanNama().
- Jika kita coba mengaksesnya di main(), akan muncul error karena nama adalah variabel lokal.

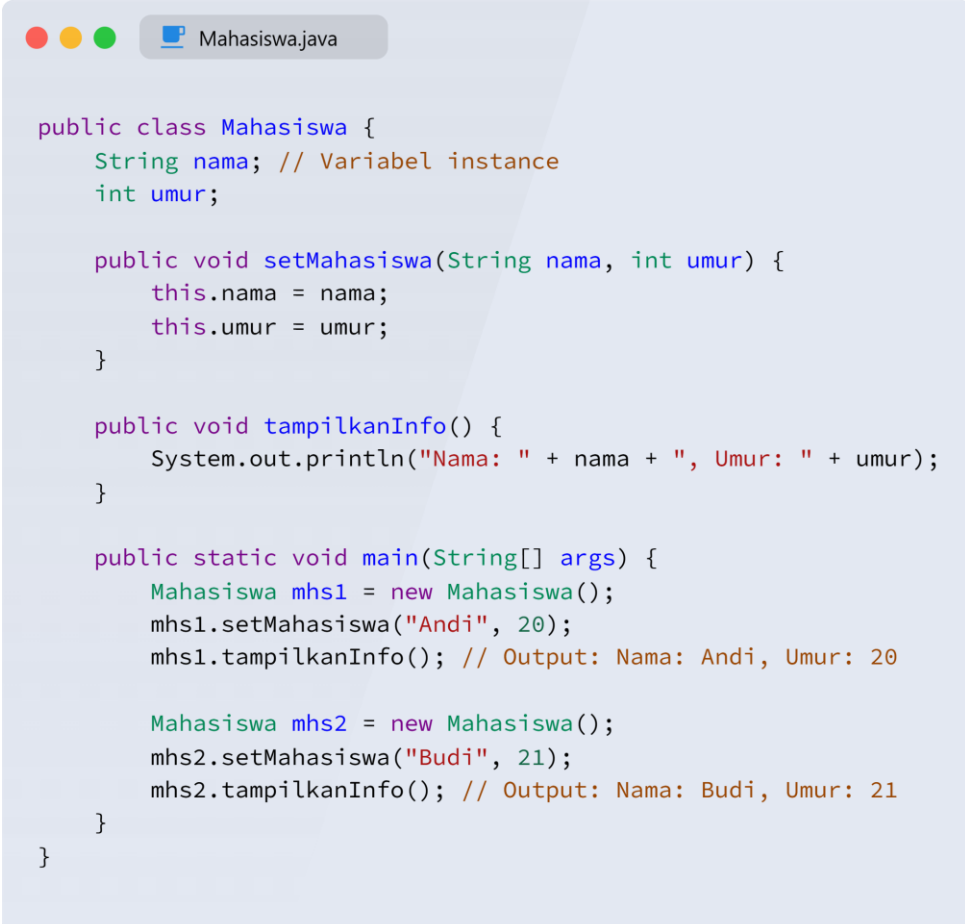
##### b) Variabel Instance (Instance variables)

Variabel yang dideklarasikan di dalam kelas tetapi di luar method, dan setiap objek (instance) dari kelas akan memiliki salinan sendiri dari variabel ini.

Ciri-ciri variabel instance:

- Dideklarasikan di dalam kelas, tetapi di luar method.
- Milik masing-masing objek dan setiap objek memiliki nilai variabel sendiri.
- Memiliki nilai default jika tidak diinisialisasi secara eksplisit:
  - int → 0
  - double → 0.0
  - boolean → false
  - String / objek → null
- Dapat memiliki modifier akses (private, public, protected).

Contoh:



```
public class Mahasiswa {
    String nama; // Variabel instance
    int umur;

    public void setMahasiswa(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
    }

    public void tampilkanInfo() {
        System.out.println("Nama: " + nama + ", Umur: " + umur);
    }

    public static void main(String[] args) {
        Mahasiswa mhs1 = new Mahasiswa();
        mhs1.setMahasiswa("Andi", 20);
        mhs1.tampilkanInfo(); // Output: Nama: Andi, Umur: 20

        Mahasiswa mhs2 = new Mahasiswa();
        mhs2.setMahasiswa("Budi", 21);
        mhs2.tampilkanInfo(); // Output: Nama: Budi, Umur: 21
    }
}
```

- Setiap objek (mhs1 dan mhs2) memiliki nilai nama dan umur masing-masing.
- Variabel instance dapat digunakan di seluruh kelas dan memiliki nilai default jika tidak diinisialisasi.

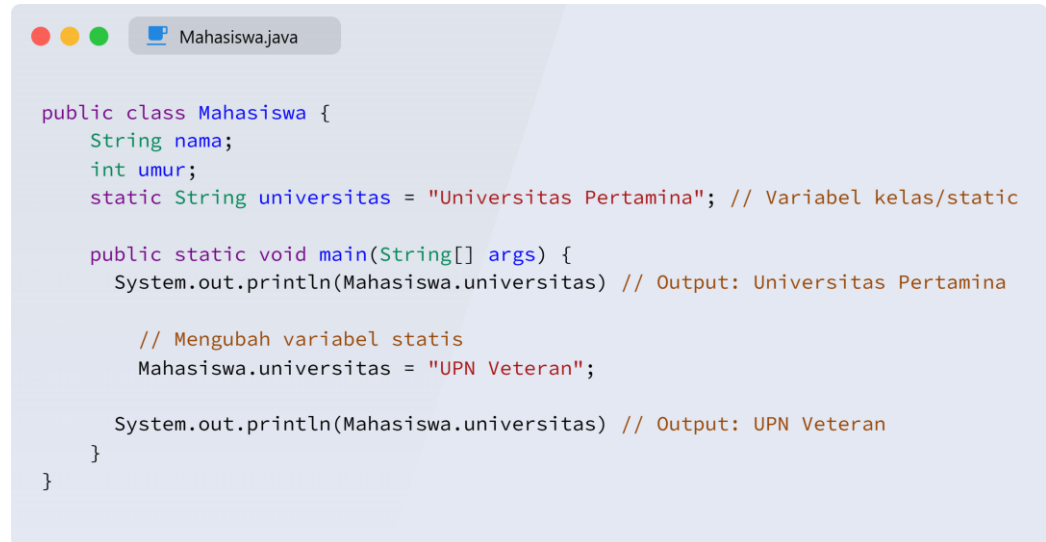
c) Variabel kelas / Static variable

Variabel yang dideklarasikan dengan keyword static, yang berarti nilai variabel bersama untuk semua objek dari kelas tersebut.

Ciri-ciri variable kelas:

- Dideklarasikan dengan keyword static.
  - Nilai variabel berbagi di semua instance (objek) dari kelas.
  - Hanya ada satu salinan di memori (shared memory).
  - Bisa diakses menggunakan nama kelas tanpa perlu membuat objek.
  - Memiliki nilai default seperti variabel instance.

Contoh:



```

public class Mahasiswa {
    String nama;
    int umur;
    static String universitas = "Universitas Pertamina"; // Variabel kelas/static

    public static void main(String[] args) {
        System.out.println(Mahasiswa.universitas) // Output: Universitas Pertamina

        // Mengubah variabel statis
        Mahasiswa.universitas = "UPN Veteran";

        System.out.println(Mahasiswa.universitas) // Output: UPN Veteran
    }
}

```

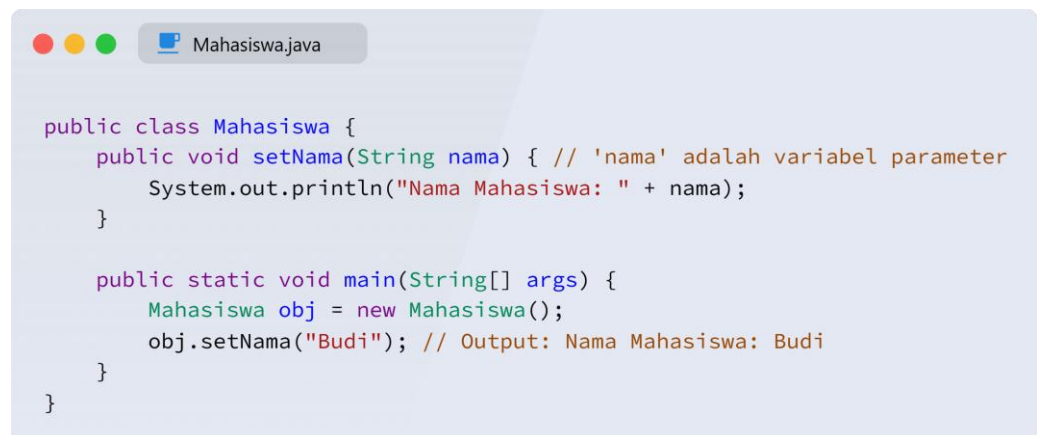
#### d) Variabel Parameter

Variabel yang dideklarasikan di dalam parameter method atau konstruktor dan hanya bisa digunakan dalam method atau konstruktor tersebut.

Ciri-ciri variabel parameter:

- Dideklarasikan dalam tanda kurung () pada method atau konstruktor.
- Bersifat lokal dan hanya bisa digunakan dalam method tempatnya dideklarasikan.
- Tidak memiliki nilai default, sehingga harus diberikan nilai saat method dipanggil.

Contoh:



```

public class Mahasiswa {
    public void setNama(String nama) { // 'nama' adalah variabel parameter
        System.out.println("Nama Mahasiswa: " + nama);
    }

    public static void main(String[] args) {
        Mahasiswa obj = new Mahasiswa();
        obj.setNama("Budi"); // Output: Nama Mahasiswa: Budi
    }
}

```

- **nama** adalah variabel parameter yang hanya bisa digunakan dalam method `setNama()`.

e) Perbandingan Semua Jenis Variabel:

Jenis Variabel	Lokasi Deklarasi	Cakupan	Nilai Default	Modifier Akses
Lokal	Dalam method/konstruktor	Hanya dalam blok tersebut	Tidak ada (harus diinisialisasi)	Tidak bisa pakai modifier akses
Instance	Dalam kelas, di luar method	Berlaku untuk tiap objek	Ada (0, null, false, dll.)	Bisa pakai private, public, protected
Kelas (Static)	Dalam kelas dengan static	Berlaku untuk semua objek (shared)	Ada	Bisa pakai private, public, protected
Parameter	Dalam tanda kurung method/konstruktor	Hanya dalam method tersebut	Tidak ada (harus diberikan nilai saat dipanggil)	Tidak bisa pakai modifier akses

## 5.2 Java Packages

*Package* adalah mekanisme untuk mengelompokkan kelas, antarmuka (interface), dan sub-package yang saling terkait ke dalam satu unit logis. Package dalam Java digunakan untuk mengelompokkan kelas-kelas yang saling terkait. Kita dapat membayangkannya package seperti sebuah **folder dalam direktori file**. Tujuan utama penggunaan package adalah untuk menghindari konflik nama (name conflicts) antar kelas, serta untuk membuat kode yang lebih terorganisir dan mudah dipelihara.

Package dalam Java dibagi menjadi dua kategori utama:

- **Built-in Packages** (Package bawaan dari Java API): Package yang sudah tersedia dalam Java API. Contohnya termasuk `java.util`, `java.io`, dan `java.lang`
- **User-defined Packages** (Package yang dibuat oleh pengguna): Package yang dirancang dan dibuat sendiri oleh programmer. Digunakan untuk mengorganisasi kode berdasarkan kebutuhan spesifik proyek.

Berikut akan dijelaskan lebih lanjut mengenai masing-masing package dalam java:

### 1) Built-in Packages

Java API adalah perpustakaan berisi kelas-kelas yang telah ditulis sebelumnya, yang dapat digunakan secara gratis dan sudah termasuk dalam Java Development Environment.

Perpustakaan ini mencakup berbagai komponen untuk mengelola input, pemrograman database, dan banyak lagi. Daftar lengkapnya dapat ditemukan di situs web Oracle: [Java API Documentation](https://docs.oracle.com/javase/10/docs/api/).

Perpustakaan ini dibagi menjadi paket (package) dan kelas (class). Artinya, Kita dapat mengimpor satu kelas tertentu (beserta metode dan atributnya), atau mengimpor seluruh paket yang berisi semua kelas yang termasuk dalam paket tersebut.

Untuk menggunakan kelas atau paket dari perpustakaan ini, kita perlu menggunakan kata kunci **import**.

Sintaks dasar:

```
import package.name.Class;    // Import a single class
import package.name.*;       // Import the whole package
```

Contoh:

```
import java.util.Scanner;    // Mengimpor kelas Scanner dari paket java.util
import java.io.*;           // Mengimpor semua kelas dari paket java.io
```

- Mengimpor Class: `import java.util.Scanner;`



```
import java.util.Scanner;

class MyClass {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Masukkan nama kamu: ");

        String nama = input.nextLine();
        System.out.println("Nama kamu adalah: " + nama);
    }
}
```

- Contoh Import package: `import java.util.*;`

```
import java.util.*;
```

## 2) User-defined Packages

Dalam java, package sama dengan folder dalam direktori. Jadi jika kita ingin membuat package maka sama halnya dengan membuat folder. Ada beberapa aturan

dan ketentuan yang harus diperhatikan saat membuat **package** agar program berjalan dengan baik. Berikut adalah ketentuannya:

a) Struktur Direktori Harus Sesuai dengan Nama Package:

Contoh:

Jika mendeklarasikan package mypackage;, maka file .java harus disimpan dalam folder **mypackage/**.

```
project-folder/  
├─ mypackage/  
│   ├─ MyClass.java  
│   └─ AnotherClass.java  
└─ Main.java
```

b) Menggunakan Konvensi Penamaan Package

- Nama package sebaiknya menggunakan huruf **kecil semua** untuk menghindari konflik dengan nama kelas atau interface.
- Untuk package yang dibuat oleh perusahaan atau tim besar, biasanya menggunakan **domain terbalik**. Misalnya, perusahaan example.com membuat package Java:

```
package com.example.mypackage;
```

- Nama package **tidak boleh mengandung spasi atau karakter khusus** selain '\_' dan '.'.

c) Penulisan Pernyataan package di Awal File

Pernyataan package harus ditulis di baris pertama sebelum kode lainnya (kecuali komentar).

Contoh:

```
package mypackage; // Deklarasi package harus di baris pertama  
  
public class MyClass {  
    public void showMessage() {  
        System.out.println("Hello from MyClass in mypackage!");  
    }  
}
```

d) Mengimpor Package untuk Digunakan di Kelas Lain

Untuk menggunakan kelas dalam package lain, gunakan import.

Contoh:

```
import mypackage.MyClass; // Mengimpor kelas dari package

public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.showMessage();
    }
}
```

Bisa juga mengimpor semua kelas dalam package dengan \*:

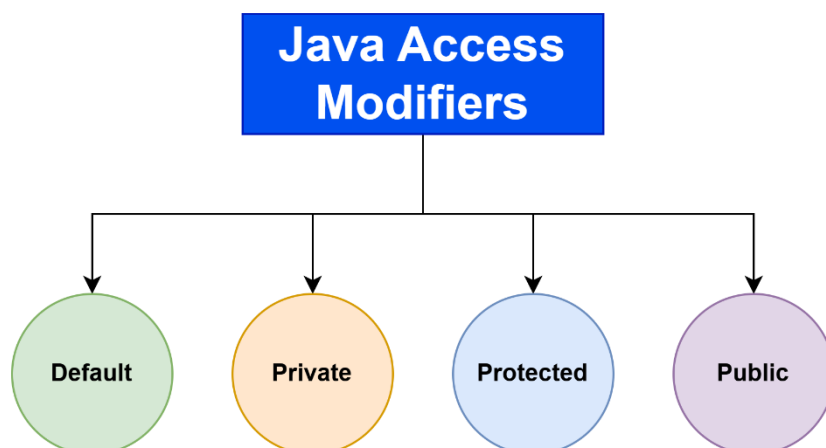
```
import mypackage.*;
```

### 5.3 Access Modifiers

Dalam Java, *access modifiers* adalah mekanisme yang digunakan untuk mengontrol aksesibilitas kelas, variabel, konstruktor, dan metode. Dengan menggunakan access modifiers, kita dapat membatasi bagian mana saja dari program yang dapat mengakses suatu member (atribut atau metode) sehingga mendukung prinsip enkapsulasi dan keamanan data pada pemrograman berorientasi objek.

Ada 4 jenis access modifiers dalam java, yakni:

- Default (tidak diperlukan kata kunci)
- Private
- Protected
- Public

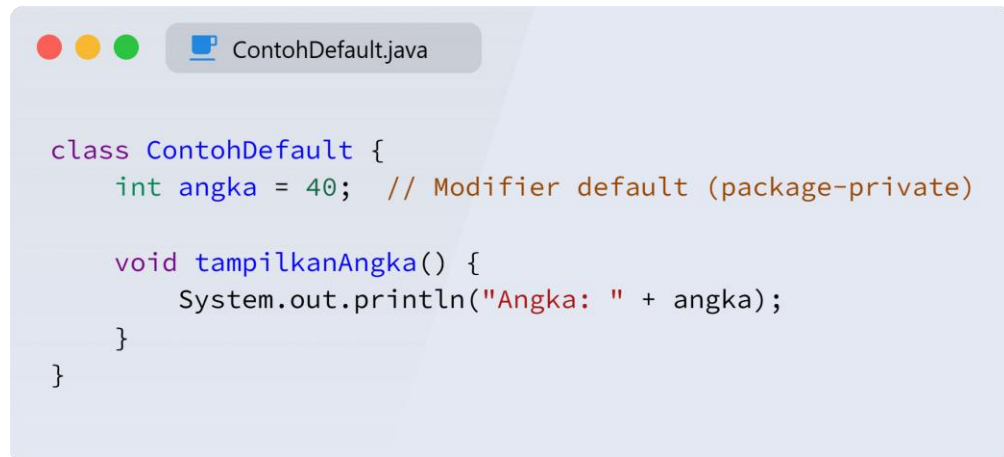


#### 1) Default

Artinya kita tidak mendeklarasikan access modifier secara eksplisit untuk class, field, method, konstruktor, dll. Jika tidak ada modifier akses yang dideklarasikan

(alias tidak ada public, private, atau protected), maka akses bersifat **default** atau **package-private**. Anggota tersebut hanya dapat diakses oleh kelas-kelas yang berada dalam **package yang sama**.

Contoh:



```
class ContohDefault {  
    int angka = 40; // Modifier default (package-private)  
  
    void tampilkanAngka() {  
        System.out.println("Angka: " + angka);  
    }  
}
```

Access modifier default cocok untuk situasi di mana anggota hanya relevan untuk digunakan oleh kelas-kelas dalam satu package saja.

## 2) Private

Metode, variabel, dan konstruktor yang dideklarasikan sebagai private hanya dapat diakses di dalam kelas yang dideklarasikan itu sendiri. Access modifiers private adalah tingkat akses yang paling ketat. Kelas dan interface tidak dapat dideklarasikan sebagai private.

Variabel yang dideklarasikan sebagai private dapat diakses dari luar kelas jika metode getter publik tersedia di dalam kelas tersebut.

Penggunaan modifier private adalah cara utama bagi suatu objek untuk melakukan enkapsulasi dirinya sendiri dan menyembunyikan data dari dunia luar.



```
public class ContohPrivate {  
    private int angka = 20;  
  
    private void tampilkanAngka() {  
        System.out.println("Angka: " + angka);  
    }  
  
    public void aksesTampil() {  
        tampilkanAngka(); // Boleh dipanggil di dalam kelas yang sama  
    }  
}
```

## 3) Protected

Variabel, metode, dan konstruktor yang dideklarasikan sebagai **protected** dalam sebuah superclass hanya dapat diakses oleh subclass di package lain atau oleh kelas



mana pun dalam package yang sama dengan kelas yang memiliki anggota **protected**. Modifier akses **protected** tidak dapat diterapkan pada kelas dan antarmuka (**interface**). Namun, metode dan variabel dapat dideklarasikan sebagai **protected**, tetapi metode dan variabel dalam sebuah **interface** tidak dapat menggunakan modifier ini. Akses **protected** memberikan kesempatan kepada subclass untuk menggunakan metode atau variabel pembantu, sekaligus mencegah kelas yang tidak terkait mencoba menggunakannya.



```
public class ContohProtected {
    protected int angka = 30;

    protected void tampilkanAngka() {
        System.out.println("Angka: " + angka);
    }
}

// Kelas turunan di package yang sama atau berbeda
public class Turunan extends ContohProtected {
    public void cetakAngka() {
        // Dapat mengakses anggota protected dari kelas induk
        System.out.println("Angka dari Turunan: " + angka);
    }
}
```

Penggunaan **protected** berguna ketika kita ingin agar anggota kelas bisa diakses oleh subclass, namun tidak terbuka untuk semua kelas di luar paket jika tidak berada dalam hubungan pewarisan.

#### 4) Public

Sebuah kelas, metode, konstruktor, antarmuka, dan lainnya yang dideklarasikan sebagai **public** dapat diakses dari kelas mana pun. Oleh karena itu, field, metode, dan blok yang dideklarasikan di dalam kelas **public** dapat diakses dari kelas mana pun yang termasuk dalam ekosistem Java.

Namun, jika kelas **public** yang ingin kita akses berada di package yang berbeda, maka kelas tersebut tetap perlu diimpor terlebih dahulu. Karena adanya pewarisan kelas (inheritance), semua metode dan variabel **public** dari suatu kelas akan diwarisi oleh subclass-nya.

```
ContohPublic.java

public class ContohPublic {
    public int angka = 10;

    public void tampilkanAngka() {
        System.out.println("Angka: " + angka);
    }
}
```

Penggunaan public cocok untuk metode atau variabel yang memang ingin diakses secara luas.

Tabel berikut memberikan gambaran ringkas tentang cakupan akses dari masing-masing access modifiers:

Modifier	Akses dalam Kelas	Akses dalam Package	Akses dalam Subclass (beda package)	Akses dari Luar Package
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

- **public:** Anggota dengan modifier ini dapat diakses dari kelas manapun, baik dalam satu package maupun di luar package.
- **protected:** Anggota dengan modifier ini dapat diakses dalam kelas yang sama, oleh subclass (meskipun berada di package berbeda), dan oleh kelas lain dalam package yang sama. Namun, anggota protected tidak dapat diakses langsung oleh kelas non-subclass di luar package.
- **default:** Jika tidak ada modifier yang ditentukan, aksesnya bersifat package-private, artinya hanya dapat diakses dalam package yang sama.
- **private:** Anggota dengan modifier ini hanya dapat diakses di dalam kelas itu sendiri.


## 5.4 Implementasi Encapsulation (Enkapsulasi)

Enkapsulasi adalah salah satu konsep fundamental dalam pemrograman berorientasi objek (OOP) yang berfokus pada menyembunyikan data internal dari suatu objek dan hanya memperbolehkan akses melalui metode yang telah ditentukan. Dengan cara ini, data dilindungi dari modifikasi langsung yang tidak diinginkan dari luar kelas, dan pengembang dapat menyisipkan logika validasi atau aturan tertentu ketika data tersebut diubah.

Untuk mengimplementasikan enkapsulasi, biasanya atribut atau data pada kelas dibuat dengan modifier akses `private`, sehingga tidak bisa diakses secara langsung dari luar kelas. Untuk memberikan akses yang terkendali terhadap data tersebut, digunakan metode publik yang disebut `getter` dan `setter`.

- **Getter** adalah metode yang bertugas mengambil atau mengembalikan nilai atribut. Metode ini biasanya diberi nama dengan awalan “`get`” diikuti nama atribut (misalnya `getNama` untuk mengambil nilai variabel `nama`). `Getter` memungkinkan kelas lain untuk membaca nilai dari atribut `private` tanpa bisa mengubah nilainya secara langsung.
- **Setter** adalah metode yang digunakan untuk mengubah nilai atribut dengan cara yang terkontrol. Metode ini biasanya diberi nama dengan awalan “`set`” diikuti nama atribut (misalnya `setNama` untuk mengubah nilai variabel `nama`). Di dalam `setter`, kita bisa menambahkan logika validasi—misalnya memastikan bahwa nilai yang diberikan valid—sebelum menetapkan nilai baru pada atribut tersebut.

Berikut contoh implementasi enkapsulasi dengan `getter` dan `setter`:



```

public class Mahasiswa {
    // Atribut privat: tidak dapat diakses langsung dari luar kelas
    private String nama;
    private int umur;

    // Konstruktor untuk menginisialisasi objek Mahasiswa
    public Mahasiswa(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
    }

    // Getter untuk mengambil nilai atribut 'nama'
    public String getNama() {
        return nama;
    }

    // Setter untuk mengubah nilai atribut 'nama'
    public void setNama(String nama) {
        this.nama = nama;
    }

    // Getter untuk mengambil nilai atribut 'umur'
    public int getUmur() {
        return umur;
    }

    // Setter untuk mengubah nilai atribut 'umur' dengan validasi
    public void setUmur(int umur) {
        if (umur > 0) { // Validasi: umur harus lebih besar dari 0
            this.umur = umur;
        } else {
            System.out.println("Nilai umur tidak valid.");
        }
    }

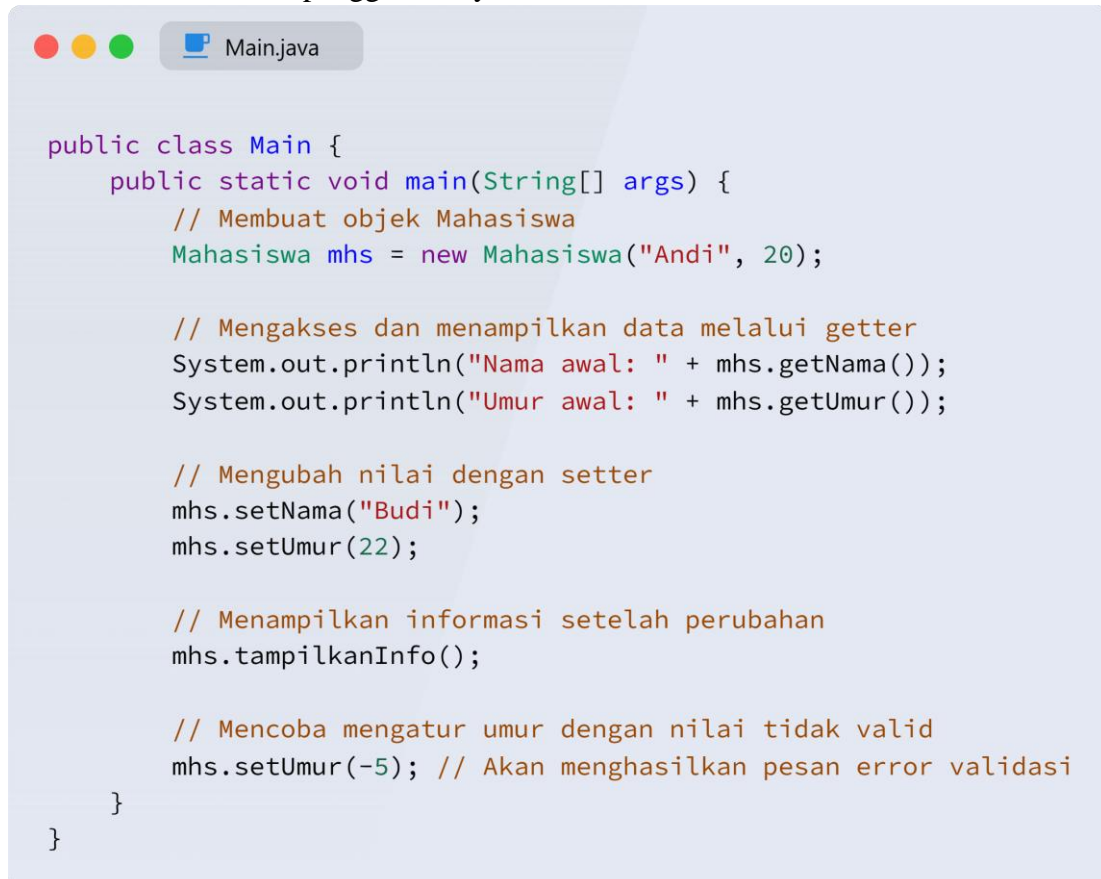
    // Metode untuk menampilkan informasi mahasiswa
    public void tampilkanInfo() {
        System.out.println("Nama: " + nama + ", Umur: " + umur);
    }
}

```

- **Atribut Privat:** Variabel nama dan umur dideklarasikan sebagai private sehingga hanya dapat diakses dari dalam kelas itu sendiri. Ini mencegah modifikasi langsung yang bisa merusak integritas data.
- **Konstruktor:** Digunakan untuk menginisialisasi nilai awal atribut ketika objek Mahasiswa dibuat.
- **Getter dan Setter:**

- `getNama()` dan `getUmur()` memberikan akses baca terhadap atribut.
- `setNama()` dan `setUmur()` memberikan akses ubah dengan tambahan validasi. Misalnya, dalam `setUmur()`, hanya nilai umur yang positif yang diterima, sehingga data tidak akan diubah dengan nilai yang tidak valid.
- **Tampilan Informasi:** Metode `tampilkanInfo()` menampilkan nilai dari atribut, yang kemudian bisa dilihat setelah proses modifikasi menggunakan setter.

Berikut contoh penggunaannya:



```

public class Main {
    public static void main(String[] args) {
        // Membuat objek Mahasiswa
        Mahasiswa mhs = new Mahasiswa("Andi", 20);

        // Mengakses dan menampilkan data melalui getter
        System.out.println("Nama awal: " + mhs.getNama());
        System.out.println("Umur awal: " + mhs.getUmur());

        // Mengubah nilai dengan setter
        mhs.setNama("Budi");
        mhs.setUmur(22);

        // Menampilkan informasi setelah perubahan
        mhs.tampilkanInfo();

        // Mencoba mengatur umur dengan nilai tidak valid
        mhs.setUmur(-5); // Akan menghasilkan pesan error validasi
    }
}

```

## 5.5 Array of Object

Dalam Java, **array of objects** adalah struktur data yang memungkinkan penyimpanan beberapa **objek** dari kelas yang sama dalam satu array. Setiap elemen dalam array ini merupakan referensi ke objek individual. Pendekatan ini berguna ketika kita perlu mengelola sekelompok objek secara efisien, seperti daftar siswa, produk, atau entitas lainnya.

Dalam contoh di bawah ini, kita akan mendemonstrasikan cara **membuat array objek** dan menginisialisasinya dengan nilai yang berbeda. Kemudian menampilkan detail setiap objek yang disimpan dalam array.

- Membuat Class Mahasiswa

```
Mahasiswa.java

class Mahasiswa {
    String nama;
    String nim;

    Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }

    void display() {
        System.out.println("Nama: " + nama + " NIM: " + nim);
    }
}
```

- Membuat array object dari class Mahasiswa dan memanggil method display():

```
App.java

public class App {
    public static void main(String[] args) throws Exception {
        Mahasiswa[] daftarMahasiswa = {
            new Mahasiswa("Budi", "12345"),
            new Mahasiswa("Andi", "67890"),
            new Mahasiswa("Caca", "54321")
        };

        for (Mahasiswa mhs : daftarMahasiswa) {
            mhs.display();
        }
    }
}
```

Output:

```
Nama: Budi, NIM: 12345
Nama: Andi, NIM: 67890
Nama: Caca, NIM: 54321
```

#### a. Deklarasi Array Object

Kita dapat membuat array objek seperti array lainnya. Satu-satunya perbedaan adalah bahwa elemen array adalah referensi ke objek bukan ke tipe primitif seperti array biasa.

- Deklarasi: Untuk mendeklarasikan array objek, tentukan nama kelas diikuti dengan tanda kurung siku [ ]

```

"""
    Sintaks Dasar:
    NamaKelas[] namaArray = new NamaKelas[ukuran];;
"""

// Contoh:
Mahasiswa[] daftarMahasiswa = new Mahasiswa[3];

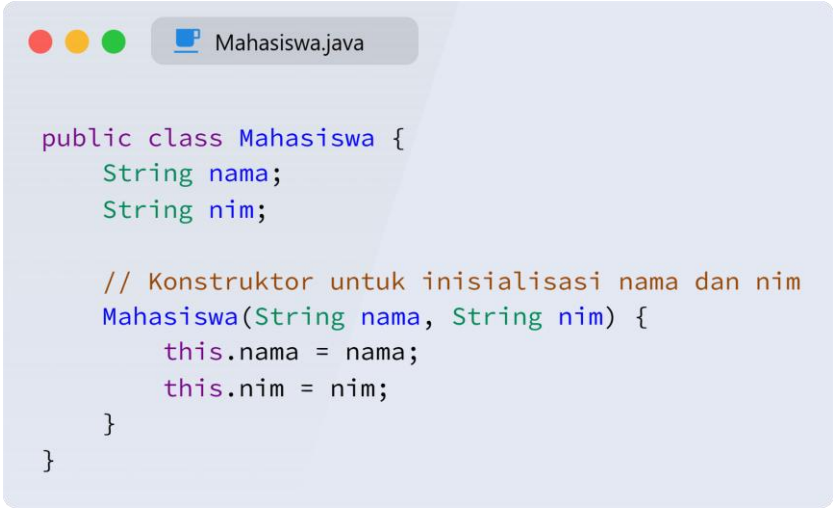
```

## b. Inisialisasi Array Object

- Menggunakan konstruktor

Contoh:

Pertama-tama pastikan konstruktor telah dibuat terlebih dahulu



```

public class Mahasiswa {
    String nama;
    String nim;

    // Konstruktor untuk inisialisasi nama dan nim
    Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }
}

```

Setelah konstruktor telah dibuat, kita dapat langsung menggunakannya untuk deklarasi dan inisialisasi array object yang akan kita buat.



```

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

        // Deklarasi array object mahasiswa berukuran 3
        Mahasiswa[] daftarMahasiswa = new Mahasiswa[3];

        // Inisialisasi object mahasiswa
        daftarMahasiswa[0] = new Mahasiswa("Budi", "123");
        daftarMahasiswa[1] = new Mahasiswa("Andi", "456");
        daftarMahasiswa[2] = new Mahasiswa("Caca", "789");
    }
}

```

- Menggunakan Setter Methods

Kita dapat membuat objek terlebih dahulu dan kemudian menetapkan nilai menggunakan **metode setter**.

Contoh:

Pastikan terlebih dahulu method setter telah dibuat,

```
Mahasiswa.java

public class Mahasiswa {
    private String nama;
    private String nim;

    public void setNama(String nama) {
        this.nama = nama;
    }

    public String getNama() {
        return nama;
    }

    public void setNim(String nim) {
        this.nim = nim;
    }

    public String getNim() {
        return nim;
    }
}
```

Setelah setter telah dibuat, kita dapat langsung menggunakannya untuk deklarasi dan inisialisasi array object yang akan kita buat.

```
App.java

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

        // Deklarasi array object mahasiswa berukuran 3
        Mahasiswa[] daftarMahasiswa = new Mahasiswa[3];

        // Inisialisasi object mahasiswa
        daftarMahasiswa[0].setNama("Budi");
        daftarMahasiswa[0].setNim("12345");

        daftarMahasiswa[1].setNama("Andi");
        daftarMahasiswa[1].setNim("67890");

        daftarMahasiswa[2].setNama("Caca");
        daftarMahasiswa[2].setNim("54321");

    }
}
```



Java juga memungkinkan inisialisasi array dengan objek secara langsung saat deklarasi:

```
App.java

Mahasiswa[] daftarMahasiswa = {
    new Mahasiswa("Andi", 20),
    new Mahasiswa("Budi", 21),
    new Mahasiswa("Citra", 22)
};
```

### c. Mengakses Array Object

Kita dapat mengakses elemen array object yang telah kita buat melalui getter method atau menggunakan looping untuk mengakses semua elemen secara langsung:

```
App.java

// Mengakses object mahasiswa
System.out.println("Nama: " + daftarMahasiswa[0].getNama() +
    ", NIM: " + daftarMahasiswa[0].getNim());
System.out.println("Nama: " + daftarMahasiswa[1].getNama() +
    ", NIM: " + daftarMahasiswa[1].getNim());
System.out.println("Nama: " + daftarMahasiswa[2].getNama() +
    ", NIM: " + daftarMahasiswa[2].getNim());
```

Menggunakan looping:

```
App.java

for (Mahasiswa mhs : daftarMahasiswa) {
    System.out.println("Nama: " + mhs.getNama() +
        ", NIM: " + mhs.getNim());
}
```

### d. Memodifikasi Array Object

```
App.java

daftarMahasiswa[2].setNama("Joko");

System.out.println("\nSesudah di modif:");
for (Mahasiswa mhs : daftarMahasiswa) {
    System.out.println("Nama: " + mhs.getNama() + ", NIM: " + mhs.getNim());
}
```

## Latihan

Kerjakan soal dibawah ini!

Latihan	
Durasi Pengerjaan	30 menit
Start	16:00
End	16:30
Soal	<p>Buatlah program Java untuk <b>manajemen data mahasiswa</b> yang menerapkan konsep <b>OOP</b> dengan beberapa fitur berikut:</p> <p><b>Spesifikasi Program</b></p> <p><b>1. Kelas Student (Paket: com.university.model)</b></p> <ul style="list-style-type: none"><li>• Atribut:<ul style="list-style-type: none"><li>◦ private String studentId</li><li>◦ private String name</li><li>◦ protected int age</li><li>◦ public double gpa</li></ul></li><li>• Konstruktor untuk menginisialisasi semua atribut.</li><li>• Metode <b>getter dan setter</b> untuk atribut studentId, name, age, dan gpa.</li></ul> <p><b>2. Kelas MainApp (Paket: com.university.main)</b></p> <ul style="list-style-type: none"><li>• Buat array objek berukuran 5 dari kelas Student dengan data dummy.</li><li>• Gunakan perulangan untuk mencetak semua data mahasiswa.</li></ul> <p><b>Contoh Output:</b></p> <p>=== Data Mahasiswa === ID: S001, Nama: Alice, Umur: 20, GPA: 3.8 ID: S002, Nama: Bob, Umur: 22, GPA: 3.6 ID: S003, Nama: Charlie, Umur: 21, GPA: 3.9</p>
Link Pengumpulan	

## Posttest

Kerjakan soal dibawah ini!

Pengerjaan Posttest	
Durasi Pengerjaan	30 menit
Start	16:10
End	16:40
Soal	<p>Berikut kode Java yang memiliki <b>kesalahan</b>. Perbaiki agar bisa berjalan dengan benar!</p> <pre>class Person {     private String name;     private int age;</pre>

	<pre> public Person(String n, int a) {     name = n;     age = a; }  public void display() {     System.out.println("Name: " + name + ", Age: " + age); }  class Employee extends Person {     private double salary;      public Employee(String n, int a, double s) {         salary = s;     }      public void showSalary() {         System.out.println("Salary: " + salary);     } }  public class MainApp {     public static void main(String[] args) {         Employee emp = new Employee("Alice", 25, 5.000);         emp.display();     } } </pre> <p><b>Tugas:</b></p> <ol style="list-style-type: none"> <li>1. Identifikasi dan perbaiki kesalahan yang anda temukan.</li> <li>2. Pastikan Employee bisa memanggil konstruktor Person.</li> <li>3. Pastikan objek Employee bisa mencetak nama, umur, dan gaji.</li> </ol>
Link Pengumpulan	

### **Take Home Task**

Kerjakan soal dibawah ini!

Tugas	
Durasi Pengerjaan	6 hari
Start	Kamis
End	Rabu, 2 April 2025, pukul 23:59

Template laporan	<a href="#">contoh template laporan tht</a>
Soal	<p><b>Sistem Manajemen Perpustakaan</b></p> <p>Anda diminta untuk membuat Sistem Manajemen Perpustakaan menggunakan konsep Object-Oriented Programming (OOP) di Java. Sistem ini harus mencakup:</p> <ul style="list-style-type: none"> <li>• Gunakan variabel lokal, instance, static, dan parameter dalam berbagai bagian program.</li> <li>• Buat package bernama librarysystem.</li> <li>• Di dalam package tersebut, buat beberapa kelas yang diperlukan.</li> <li>• Gunakan public, private, protected, dan default dengan tepat dalam setiap kelas.</li> <li>• Pastikan semua atribut bersifat private dan hanya bisa diakses melalui getter dan setter.</li> <li>• Gunakan array of object untuk menyimpan data buku yang tersedia di perpustakaan.</li> </ul> <p>Tugas Anda adalah membuat program Sistem Manajemen Perpustakaan yang memiliki fitur sebagai berikut:</p> <p><b>1. Kelas dan Package</b> Buat package librarysystem yang berisi:</p> <ul style="list-style-type: none"> <li>• Buku (Kelas yang merepresentasikan buku di perpustakaan).</li> <li>• Perpustakaan (Kelas utama yang mengelola koleksi buku).</li> <li>• User (Merepresentasikan pengguna perpustakaan).</li> </ul> <p><b>2. Kelas Buku (Buku.java)</b> Atribut:</p> <ul style="list-style-type: none"> <li>• judul (private)</li> <li>• penulis (private)</li> <li>• tahunTerbit (private)</li> <li>• statusDipinjam (private, boolean)</li> <li>• jumlahBukuTersedia (static)</li> <li>• Gunakan getter dan setter untuk mengakses atribut.</li> </ul> <p>Tambahkan metode:</p> <ul style="list-style-type: none"> <li>• pinjamBuku() → Mengubah statusDipinjam menjadi true.</li> <li>• kembalikanBuku() → Mengubah statusDipinjam menjadi false.</li> </ul> <p><b>3. Kelas Perpustakaan (Perpustakaan.java)</b> Atribut:</p> <ul style="list-style-type: none"> <li>• koleksiBuku (Array of Object) untuk menyimpan daftar buku.</li> </ul>

	<p>Metode:</p> <ul style="list-style-type: none"> <li>• tambahBuku(Buku buku) → Menambahkan buku ke daftar.</li> <li>• tampilkanBuku() → Menampilkan seluruh koleksi buku.</li> </ul> <p><b>4. Kelas User (User.java)</b></p> <p>Atribut:</p> <ul style="list-style-type: none"> <li>• nama (private)</li> <li>• idUser (private)</li> </ul> <p>Metode:</p> <ul style="list-style-type: none"> <li>• pinjamBuku(Buku buku) → Meminjam buku jika tersedia.</li> <li>• kembalikanBuku(Buku buku) → Mengembalikan buku ke perpustakaan.</li> </ul> <p><b>5. Kelas Main (Main.java)</b></p> <p>Berada di luar package librarysystem.</p> <p>Mengimpor package dan menjalankan simulasi:</p> <ul style="list-style-type: none"> <li>• Tambahkan beberapa buku ke perpustakaan.</li> <li>• Tampilkan daftar buku.</li> </ul> <p>Buat pengguna yang meminjam dan mengembalikan buku.</p>
Link Pengumpulan	