



DEPARTEMEN MATEMATIKA ITS

Pertemuan 7

ENCAPSULATION & INHERITANCE

PRAKTIKUM ALGORITMA DAN
PEMROGRAMAN KOMPUTER 2

Genap 2024/2025

PRESENTED BY : Putri Ghaida Tsurayya (5002221053)
Mohammad Febryan Khamim (5002221072)





OUTLINE

Berikut ini adalah **outline** pembahasan pada pertemuan hari ini:

01 **ENCAPSULATION**

02 **INHERITANCE**

03 **MARI MENCoba**

04 **PENUGASAN**



ENKAPSULASI



ENCAPSULATION

Enkapsulasi adalah suatu proses **membungkus** data (variabel) dan method yang beroperasi pada data ke dalam suatu unit, yaitu kelas, untuk **menyembunyikan** detail implementasi internal dari sebuah kelas.



KONSEP ENKAPSULASI



Jika anda memiliki suatu atribut yang **tak terlihat / tak dapat diakses** dari luar objek, kita dapat mem-'bungkus'-nya dengan sebuah method yang memungkinkan untuk membaca atau mengakses nilai.



AKSES MODIFIER

Modifier	Class dan Interface	Method dan Variabel
Default (tak ada modifier) Friendly	Dikenali di paketnya	Diwarisi subclass di paket yang sama dengan superclassnya. Dapat diakses oleh method-method di class-class yang sepaket.
Public	Dikenali di manapun	Diwarisi oleh semua subclassnya. Dapat diakses dimanapun.
Protected	Tidak dapat diterapkan	Diwarisi oleh semua subclassnya. Dapat diakses oleh method-method di class-class yang sepaket.
Private	Tidak dapat diterapkan	Tidak diwarisi oleh subclassnya Tidak dapat diakses oleh class lain.



AKSES MODIFIER

Modifier	Dalam Kelas Sendiri	Dalam Paket yang Sama	Subclass di Paket Lain	Di Luar Kelas/Paket
private	✓	✗	✗	✗
(default)	✓	✓	✗	✗
protected	✓	✓	✓	✗
public	✓	✓	✓	✓



MANFAAT ENKAPSULASI

- **Data hiding** : Mekanisme penyembunyian data, misalnya data member, dengan menyembunyikan detail implementasi
- **Data integrity** : Hanya nilai aman yang dapat diperintahkan pada satu atribut objek lewat method setter
- **Reusability** : Code yang terenkapsulasi lebih fleksibel dan digunakan ulang untuk modifikasi pada masa mendatang.
- **Security** : Data yang sensitif dapat dilindungi.



KELEMAHAN ENKAPSULASI

- Dapat meningkatkan kompleksitas program
- Membuat cara kerja program menjadi lebih susah dipahami
- Membatasi fleksibilitas program



CARA ENKAPSULASI

- Deklarasikan variabel sebuah class dengan modifier private
- Tambahkan **setter** dan **getter** method untuk memodifikasi dan melihat nilai suatu variabel



SETTER & GETTER

Setter	Getter
Setter digunakan untuk mengatur atau mengubah nilai dari atribut objek	Getter digunakan untuk mendapatkan nilai dari atribut objek
Setter memiliki parameter	Getter tidak memiliki parameter
Nama diikuti dengan kata “set”	Nama diikuti dengan kata “get”
Setter dapat memiliki metode yang hanya berfungsi sebagai <i>write-only</i> untuk mengeset nilai atribut tanpa memberikan kemampuan untuk membaca nilainya.	Getter sering digunakan untuk memberikan akses baca (<i>read-only</i>) ke atribut.



CONTOH ENKAPSULASI

```
1 // Java program demonstrating Encapsulation
2 class Programmer {
3     private String name;
4
5     // Getter and Setter for name
6     public String getName() { return name; }
7     public void setName(String name) { this.name = name; }
8 }
9
10 public class Geeks {
11     public static void main(String[] args) {
12         Programmer p = new Programmer();
13         p.setName("Geek");
14         System.out.println("Name=>" + p.getName());
15     }
16 }
```



MARI MENCUPRA



INHERITANCE



INHERITANCE

Inheritance adalah konsep yang memungkinkan sebuah **class** dapat mewarisi **atribut** dan **method** dari kelas lain. Atau dengan kata lain, kita dapat membuat **class** baru berdasarkan **class** yang sudah ada, sehingga **code program** yang ditulis di kelas induk (**superclass**) dapat digunakan kembali di kelas turunan (**subclass**).

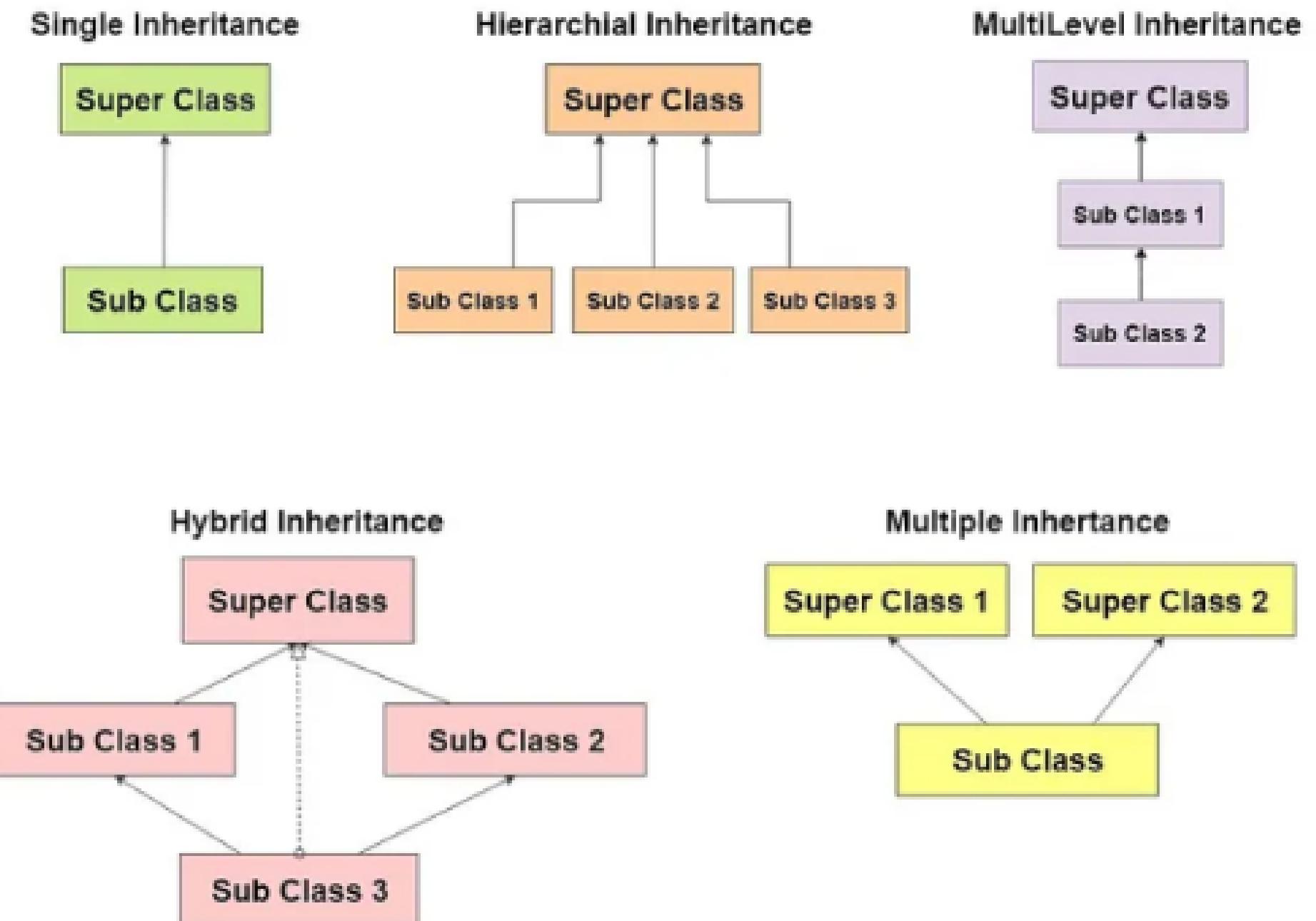


TERMINOLOGI INHERITANCE

- **Class.** *Template* atau cetak biru untuk membuat objek.
- **Superclass.** Kelas yang atribut dan methodnya akan diwariskan.
- **Subclass.** Kelas yang mewarisi atribut dan method (tetapi dapat menambahkan atribut atau method baru)
- **Extends.** Kata kunci untuk mendeklarasikan pewarisan dalam Java.



JENIS-JENIS INHERITANCE





JENIS-JENIS INHERITANCE

- **Single Inheritance.** Sebuah kelas turunan yang mewarisi satu kelas induk
- **Multilevel inheritance.** Kelas turunan mewarisi kelas induk dan kelas induk tersebut juga turunan dari kelas lain.
- **Hierarchical Inheritance.** Beberapa kelas turunan mewarisi satu kelas induk



KONSEP INHERITANCE

- Keyword **Extends** digunakan untuk melakukan inheritance dalam Java. Keyword ini memungkinkan **subclass** menurunkan atribut dan method dari **superclass**.
- **Subclass** dapat : menggunakan kembali fungsionalitas **superclass**, menambah fungsionalitas baru, mengubah perilaku lewat **override**.



APA ITU OVERRIDE?

- **Method overriding** terjadi ketika **subclass** mendefinisikan ulang method yang sudah ada di **superclass** dengan nama, parameter, dan tipe return yang sama.
- **Tujuan** : Memberikan implementasi spesifik / perilaku berbeda pada method yang diwarisi dari superclass (sesuai kebutuhan subclass).



CONTOH OVERRIDE

```
// Kelas induk (superclass)
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

// Kelas turunan (subclass) yang mengoverride method sound()
class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("Cat meows");
    }
}
```



ATURAN OVERRIDE

- Nama method **sama**
- Parameter **harus sama persis** (jumlah dan tipe)
- Method **static** dan **private** tak dapat di-**override**
- Anotasi **@Override** hanya membantu mendekripsi kesalahan saat overriding



MANFAAT INHERITANCE

- **Penggunaan Ulang Kode.** Memungkinkan pemrogram menggunakan ulang kode dan mengurangi jumlah kode yang harus ditulis
- **Abstraction.** Memungkinkan pembuatan kelas Abstract yang mendefinisikan karakter umum dari sekelompok class
- **Polymorphism.** Kemampuan agar sebuah objek dapat menjadi beberapa bentuk



KELEMAHAN INHERITANCE

- **Kompleksitas.** Membuat program menjadi lebih kompleks dan lebih sulit dipahami
- **Strong Dependencies.** Membuat kelas anak sangat bergantung pada kelas orang tua. Apabila terdapat perubahan pada kelas induk, maka kelas anak turut mengalami perubahan.
- **Mewarisi hal yang tidak dibutuhkan.** Subclass mewarisi semua method dan atribut, bahkan yang tidak dibutuhkannya.



CONTOH INHERITANCE

```
// Superclass (kelas induk)
class Animal {
    void eat() {
        System.out.println("Animal is eating...");
    }
}

// Subclass (kelas turunan)
class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking...");
    }
}
```

```
// Kelas utama untuk menjalankan program
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();

        d.eat(); // Method dari superclass
        d.bark(); // Method dari subclass
    }
}
```

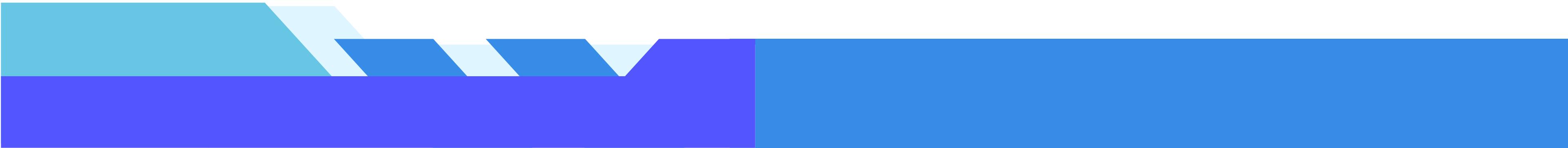


MARI MENCoba

<https://www.petanikode.com/java-oop-inheritance/>



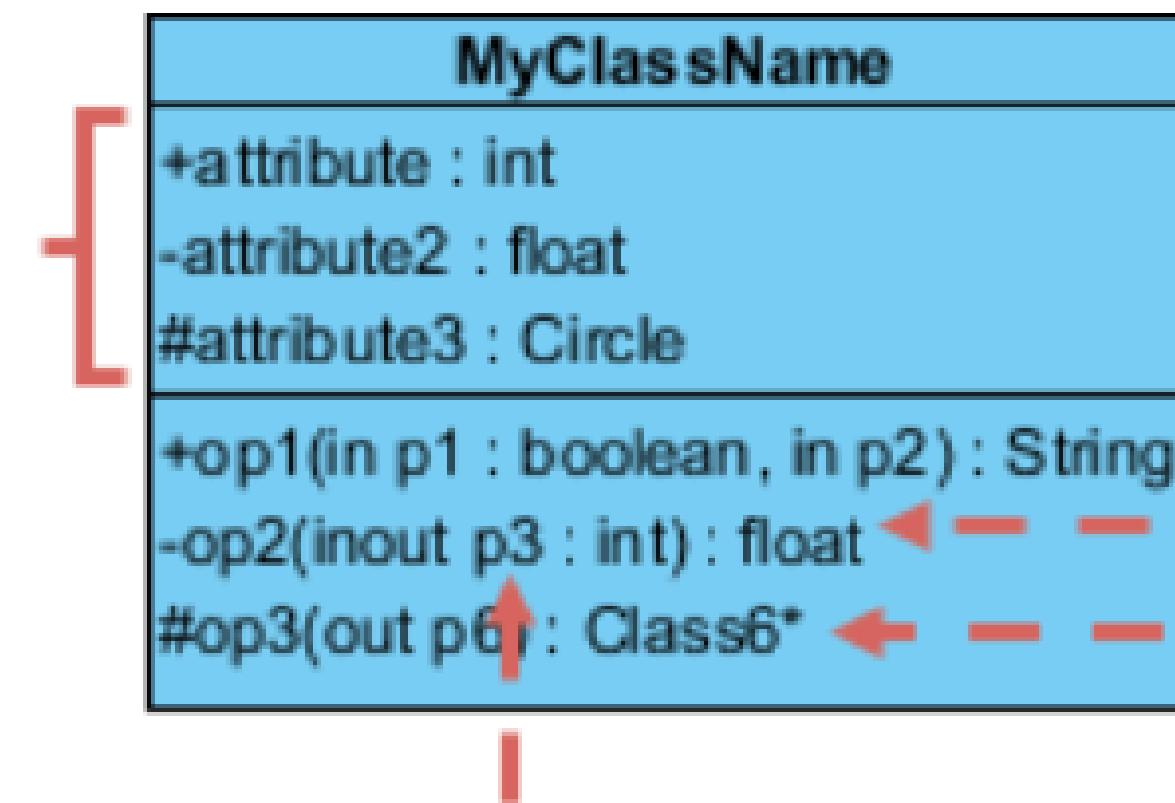
UML





NOTASI KELAS UML

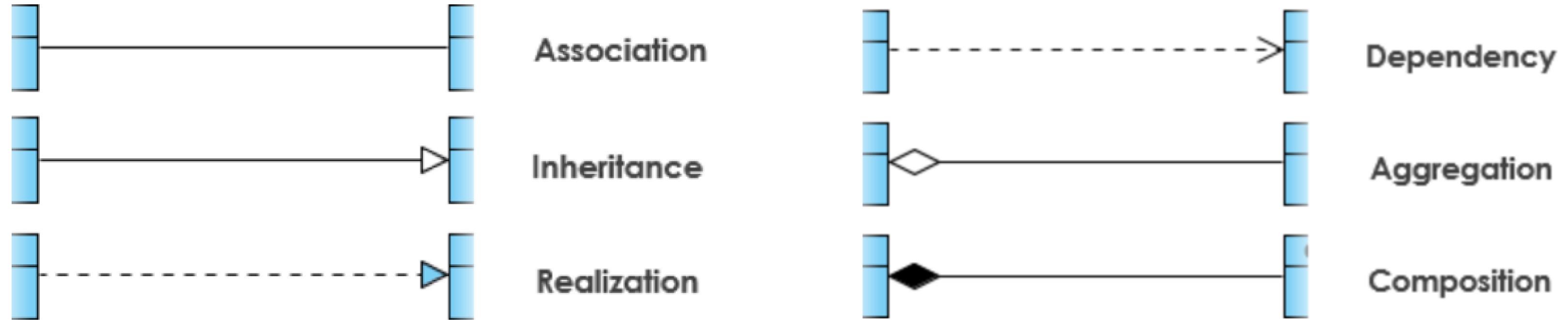
MyClassName has 3 attributes
and 3 operations



Parameter p3 of op2 is of type int



HUBUNGAN ANTAR KELAS





Association



Kelas yang saling mengenal tapi tidak saling memiliki. Bisa saling dipanggil lewat objek.

```
class Mahasiswa {
    String nama;

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

class Dosen {
    String nama;

    Dosen(String nama) {
        this.nama = nama;
    }

    void bimbingMahasiswa(Mahasiswa m) {
        System.out.println(nama + " membimbing " + m.nama);
    }
}
```



Inheritance

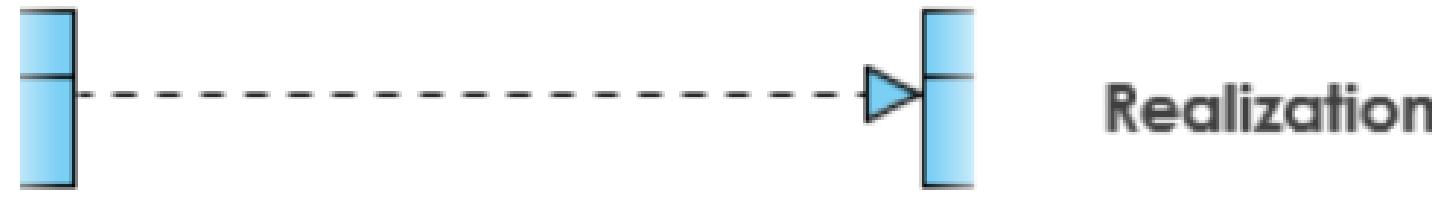


Hubungan antar kelas di mana satu kelas mewarisi atribut dan method dari kelas lain.

```
class Orang {  
    String nama;  
  
    void perkenalan() {  
        System.out.println("Halo, saya " + nama);  
    }  
  
}  
  
class Mahasiswa extends Orang {  
    String nim;  
  
    void infoMahasiswa() {  
        System.out.println("NIM: " + nim);  
    }  
}
```



Realization

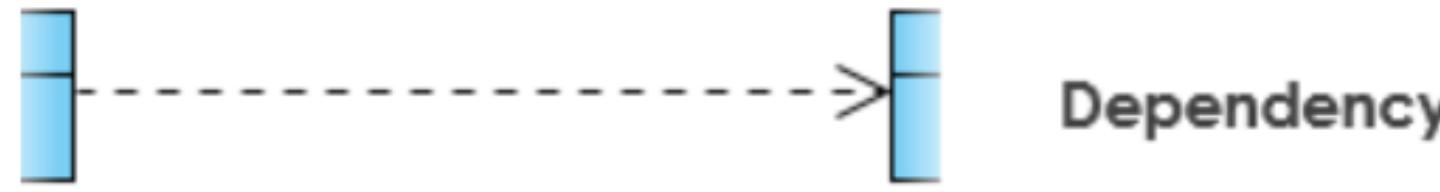


Hubungan antara interface dan kelas konkret yang mengimplementasikannya.

```
interface Printer {  
    void print();  
}  
  
class PrinterCanon implements Printer {  
    public void print() {  
        System.out.println("Mencetak dengan Printer Canon...");  
    }  
}
```



Dependency



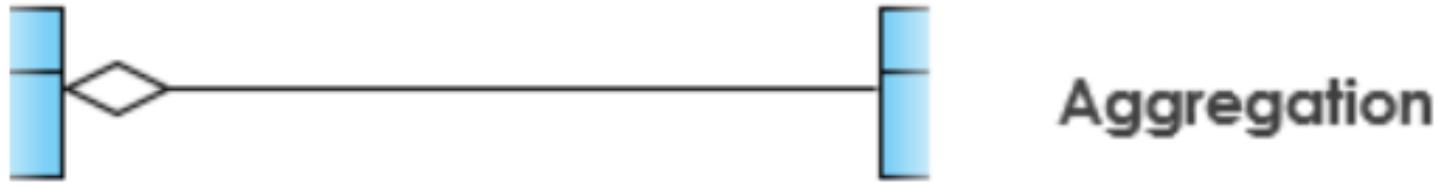
Hubungan sementara, hanya menggunakan objek kelas lain secara sesaat.

```
import java.util.Scanner;

class InputHelper {
    void mintaInput() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Masukkan nama: ");
        String nama = scanner.nextLine();
        System.out.println("Halo, " + nama);
    }
}
```



Aggregation



Hubungan di mana satu kelas menyimpan referensi ke objek kelas lain, tapi tetap independen.

```
class Dosen {  
    String nama;  
  
    Dosen(String nama) {  
        this.nama = nama;  
    }  
}  
  
class Departemen {  
    String nama;  
    ArrayList<Dosen> daftarDosen = new ArrayList<>();  
  
    void tambahDosen(Dosen d) {  
        daftarDosen.add(d);  
    }  
  
    void tampilkanDosen() {  
        for (Dosen d : daftarDosen) {  
            System.out.println("Dosen di Departemen " + nama + ": " + d.nama);  
        }  
    }  
}
```



Composition



Hubungan di mana satu kelas sepenuhnya memiliki objek dari kelas lain.

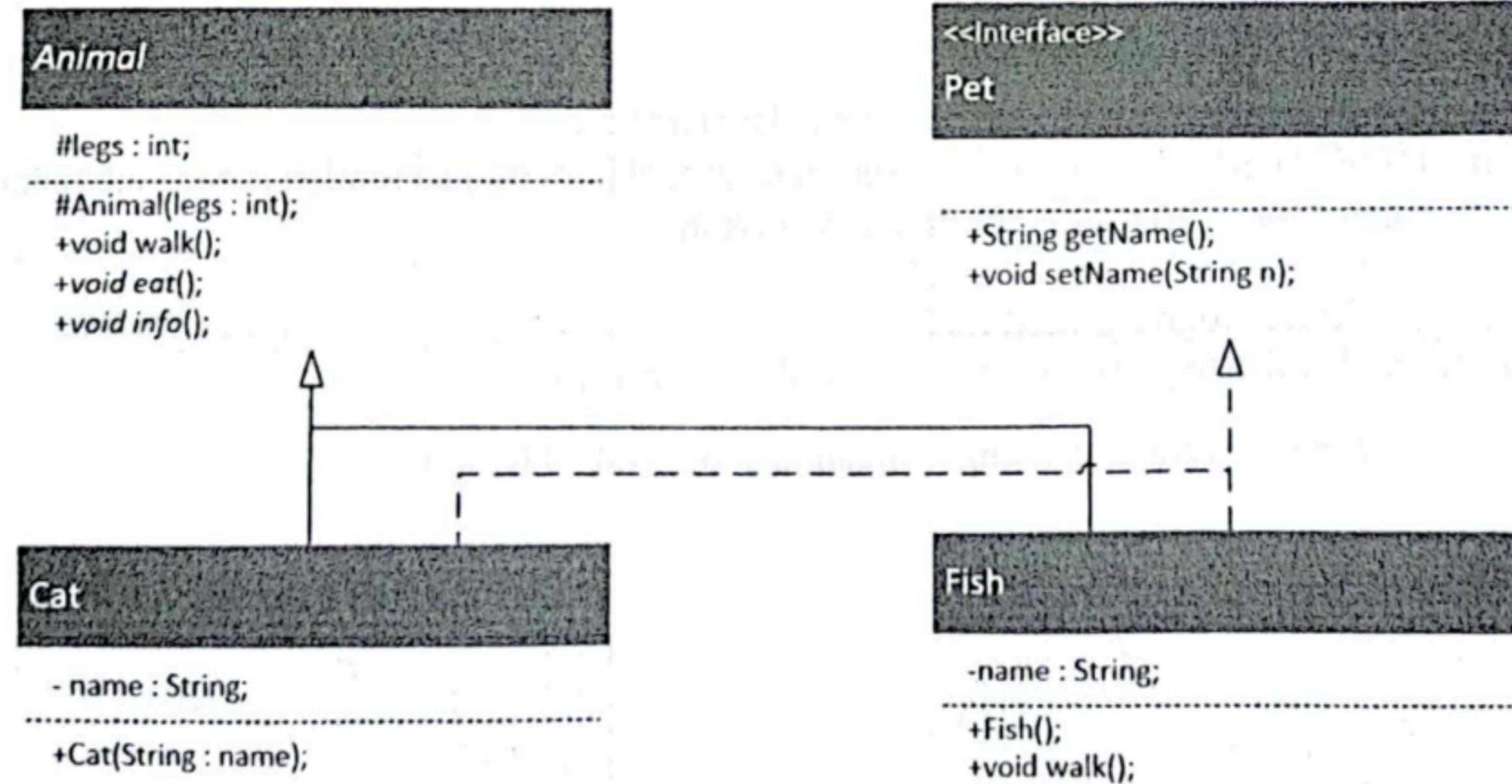
```
class Ruang {  
    String namaRuang;  
  
    Ruang(String nama) {  
        this.namaRuang = nama;  
    }  
}  
  
class Rumah {  
    Ruang ruang; // Rumah "memiliki" ruang  
  
    Rumah(String namaRuang) {  
        this.ruang = new Ruang(namaRuang); // Ruang dibuat dalam Rumah  
    }  
  
    void tampilanRuang() {  
        System.out.println("Ruang di rumah: " + ruang.namaRuang);  
    }  
}
```



MARI MENCUPRA



4. Buat implementasi (coding) dari diagram UML berikut :





KETERANGAN :

- Buat class Animal
 - Constructor Animal(int : legs) → menginisiasi legs
 - Method walk() → Mencetak kata “Berjalan”
 - Method abstract eat() → mencetak kata “Daging” di class Cat dan kata “Ganggang ” di class Fish
 - Methode abstract info() → mencetak jumlah legs; walk() dan eat()
- Buat Interface Pet
- Buat class Cat
 - Constructor Cat(String : name) akan menginisiasi nama dan legs = 4;
 - Buat method lainnya agar sesuai dengan class diagram
- Buat class Fish
 - Constructor Fish() akan menginisiasi legs = 0;



PENUGASAN



PENUGASAN

Penugasan **pertemuan** ini dapat diakses melalui pranala berikut:

[Intip.in/Pertemuan7Alprokom2](https://intip.in/Pertemuan7Alprokom2)



THANK YOU

Terima kasih banyak atas kehadirannya pada praktikum pertemuan hari ini.
Semoga mendapatkan ilmu yang bermanfaat dan diberikan semangat
dalam mengerjakan penugasan!