# Data Structure and Object Oriented-Programming

*Hafara Firdausi, M.Kom.*

*Departemen Teknologi Informasi*
*Fakultas Teknologi Elektro dan Informatika Cerdas*
*Institut Teknologi Sepuluh Nopember*

## 01. Introduction to DS-OOP

*Pengenalan SD-PBO*

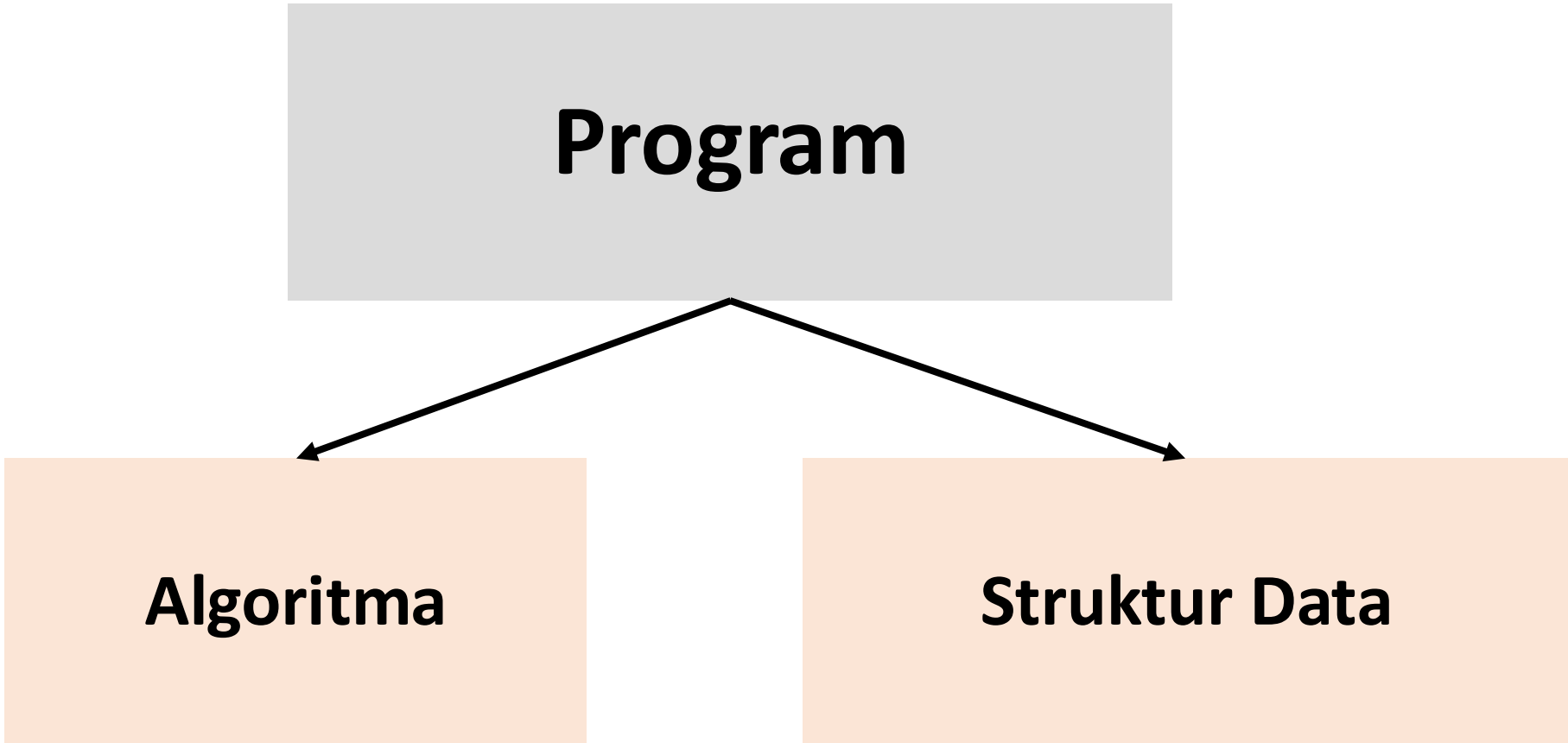
> "Get your data structures correct first, and the rest of the program will write itself."
>
> *David Jones*

SMART DATA STRUCTURES AND DUMB CODE WORKS A LOT BETTER THAN THE OTHER WAY AROUND.

- ERIC S. RAYMOND -

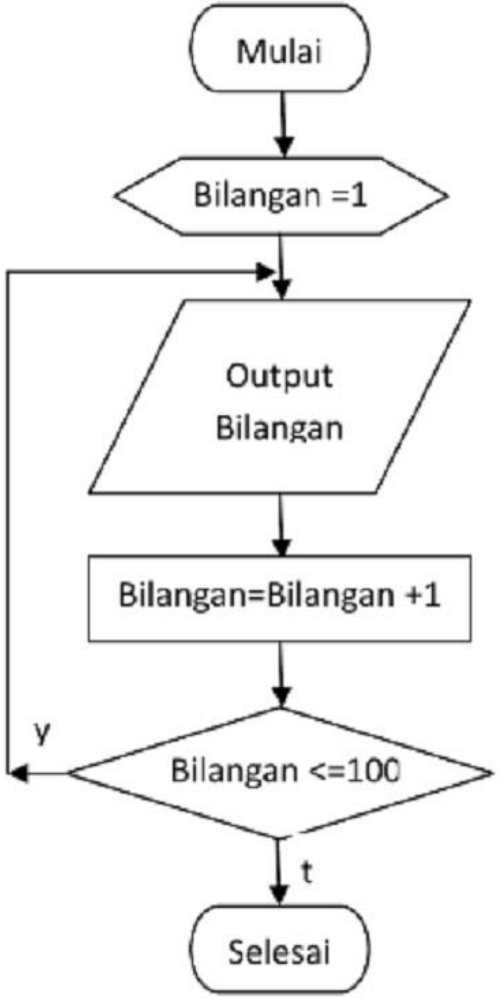
LIBQUOTES.COM

# *Program Komputer*



# Algoritma

**Deskripsi langkah-langkah penyelesaian masalah** yang tersusun secara logis.




Ciri algoritma yang baik menurut Donald E.Knuth:

- **Input**: ada minimal 0 input atau lebih
- **Ouput**: ada minimal 1 output atau lebih
- **Definite**: ada kejelasan apa yang dilakukan
- **Efective**: langkah yang dikerjakan harus efektif
- **Terminate**: langkah harus dapat berhenti / stop secara jelas

# *Pengantar Struktur Data*

# *Struktur Data*

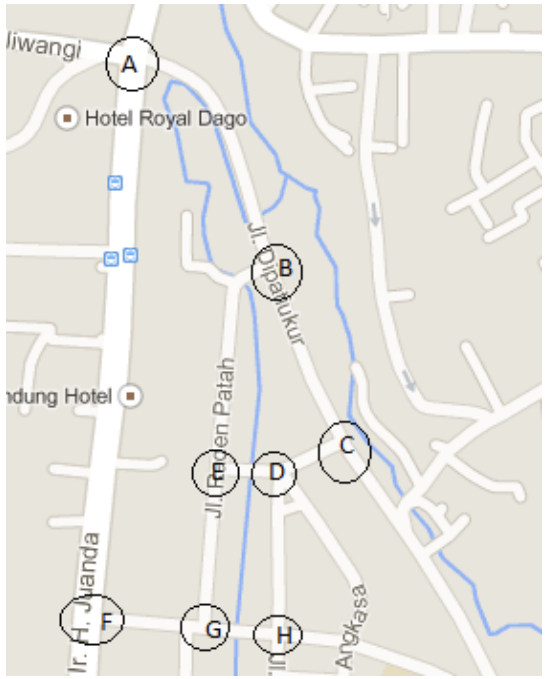
Struktur data adalah **cara menyimpan atau merepresentasikan data** di dalam komputer agar bisa dipakai secara **efisien.**

**Pemakaian struktur data yang tepat** di dalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.

# Contoh Implementasi

- Sebuah program pencarian rute terdekat di suatu kota mempunyai peta seperti di bawah ini.

Dengan jarak (meter):
A → B = 330
A → F = 690
B → C = 250
B → E = 280
C → D = 86
D → E = 80
D → H = 203
E → G = 200
F → G = 140
G → H = 95
Dengan asumsi semua jalan 2 arah




**Bagaimana cara menyimpan data peta tersebut di memori?**

# Solusi 1

Menggunakan Array 2 Dimensi

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | 0 | 330 | 0 | 0 | 0 | 690 | 0 | 0 |
| B | 330 | 0 | 250 | 0 | 280 | 0 | 0 | 0 |
| C | 0 | 250 | 0 | 86 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 86 | 0 | 80 | 0 | 0 | 203 |
| E | 0 | 280 | 0 | 80 | 0 | 0 | 200 | 0 |
| F | 690 | 0 | 0 | 0 | 0 | 0 | 140 | 0 |
| G | 0 | 0 | 0 | 0 | 200 | 140 | 0 | 95 |
| H | 0 | 0 | 0 | 203 | 0 | 0 | 95 | 0 |

# Solusi 1

Menggunakan Array 2 Dimensi

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | 0 | 330 | 0 | 0 | 0 | 690 | 0 | 0 |
| B | 330 | 0 | 250 | 0 | 280 | 0 | 0 | 0 |
| C | 0 | 250 | 0 | 86 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 86 | 0 | 80 | 0 | 0 | 203 |
| E | 0 | 280 | 0 | 80 | 0 | 0 | 200 | 0 |
| F | 690 | 0 | 0 | 0 | 0 | 0 | 140 | 0 |
| G | 0 | 0 | 0 | 0 | 200 | 140 | 0 | 95 |
| H | 0 | 0 | 0 | 203 | 0 | 0 | 95 | 0 |

**Apakah ini solusi terbaik?**

# Solusi 1

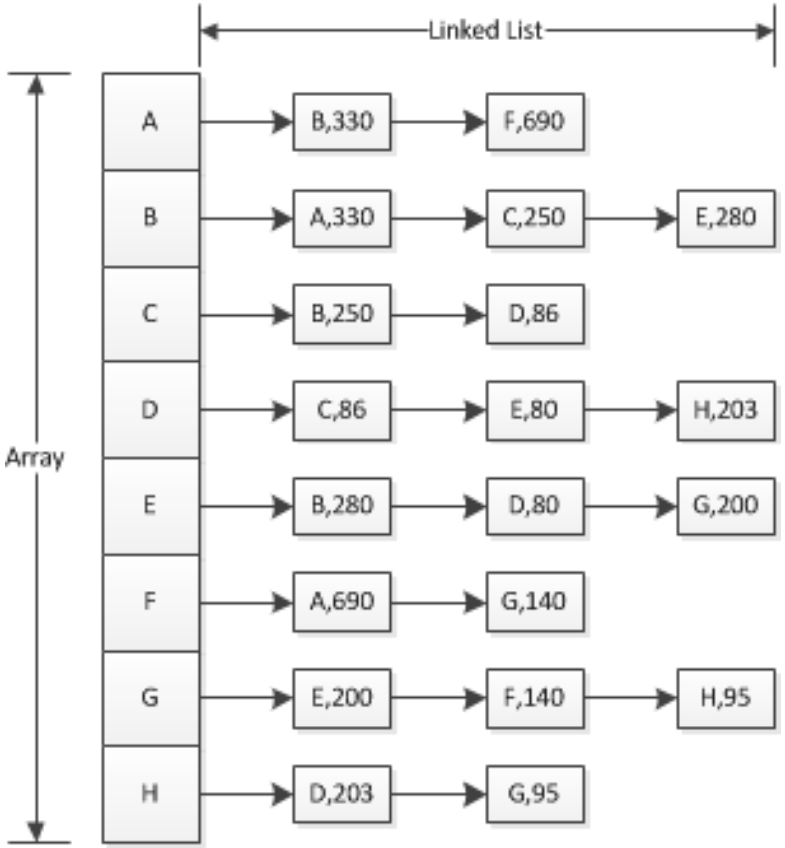
Menggunakan Array 2 Dimensi

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | 0 | 330 | 0 | 0 | 0 | 690 | 0 | 0 |
| B | 330 | 0 | 250 | 0 | 280 | 0 | 0 | 0 |
| C | 0 | 250 | 0 | 86 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 86 | 0 | 80 | 0 | 0 | 203 |
| E | 0 | 280 | 0 | 80 | 0 | 0 | 200 | 0 |
| F | 690 | 0 | 0 | 0 | 0 | 0 | 140 | 0 |
| G | 0 | 0 | 0 | 0 | 200 | 140 | 0 | 95 |
| H | 0 | 0 | 0 | 203 | 0 | 0 | 95 | 0 |

- Butuh memory yang sangat besar apabila matrix membesar
- Banyak space yang terbuang hanya untuk menyimpan **0**

# Solusi 2

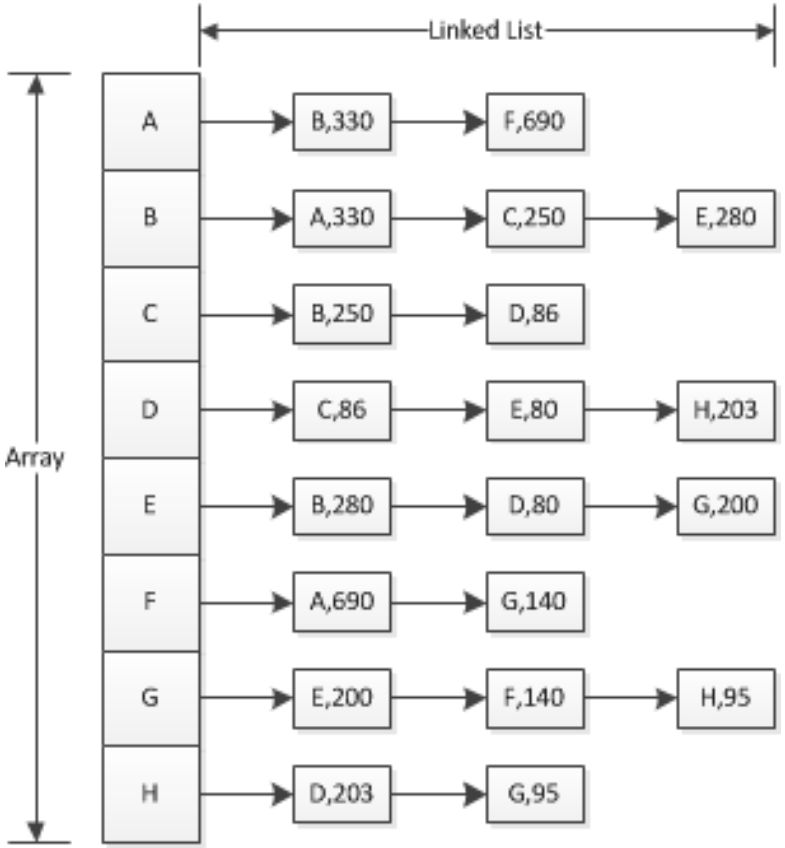
Menggunakan Array 1 Dimensi yang setiap elemennya menyimpan Linked-List




**Apakah ini solusi terbaik?**

# Solusi 2

Menggunakan Array 1 Dimensi yang setiap elemennya menyimpan Linked-List

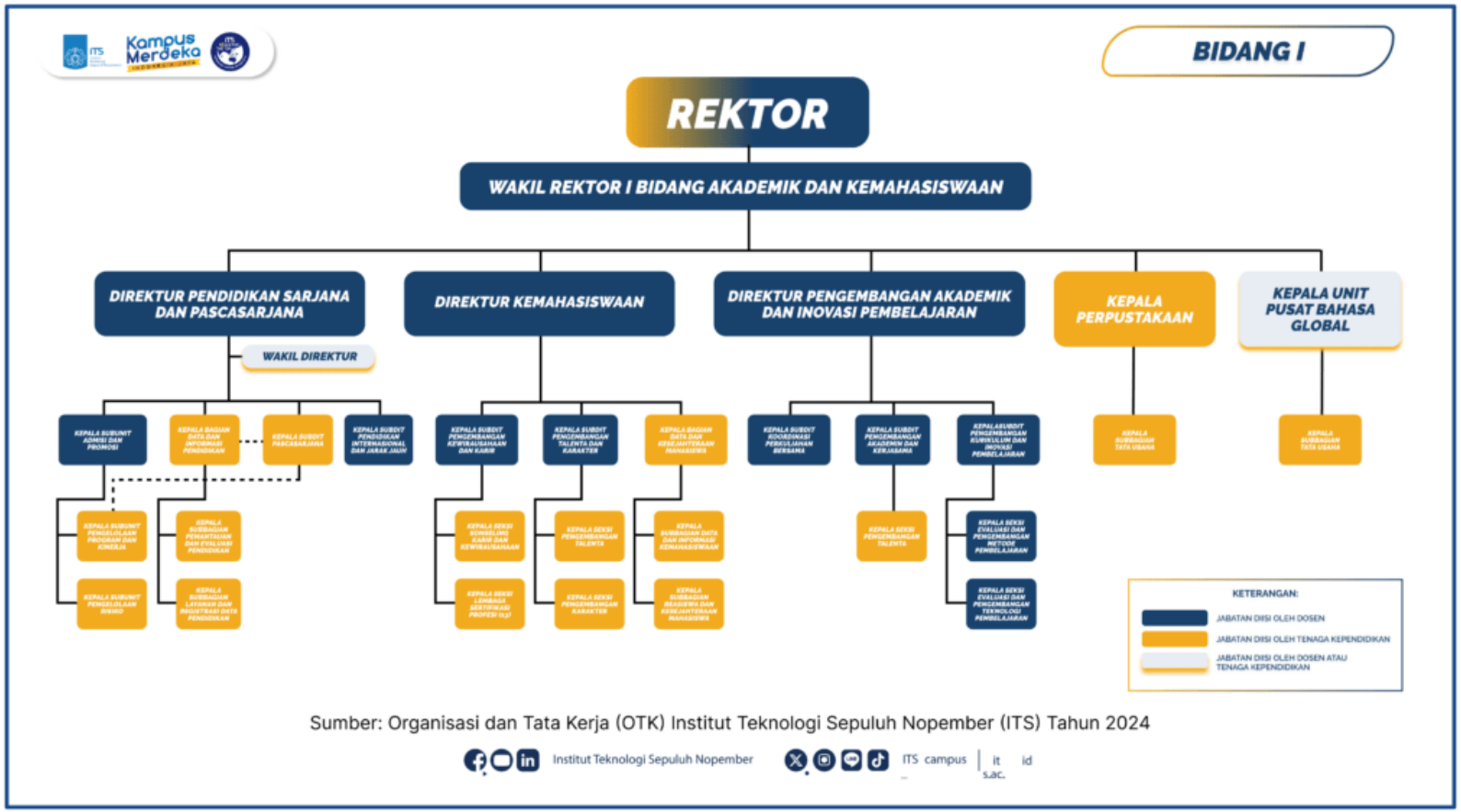



**Yap, dapat dipertimbangkan.**

Tetapi dengan struktur data ini, dibutuhkan pendekatan **algoritma khusus** dalam pembuatan strukturnya, penyimpanan, penambahan, penelusuran, penghapusan dan pencariannya.

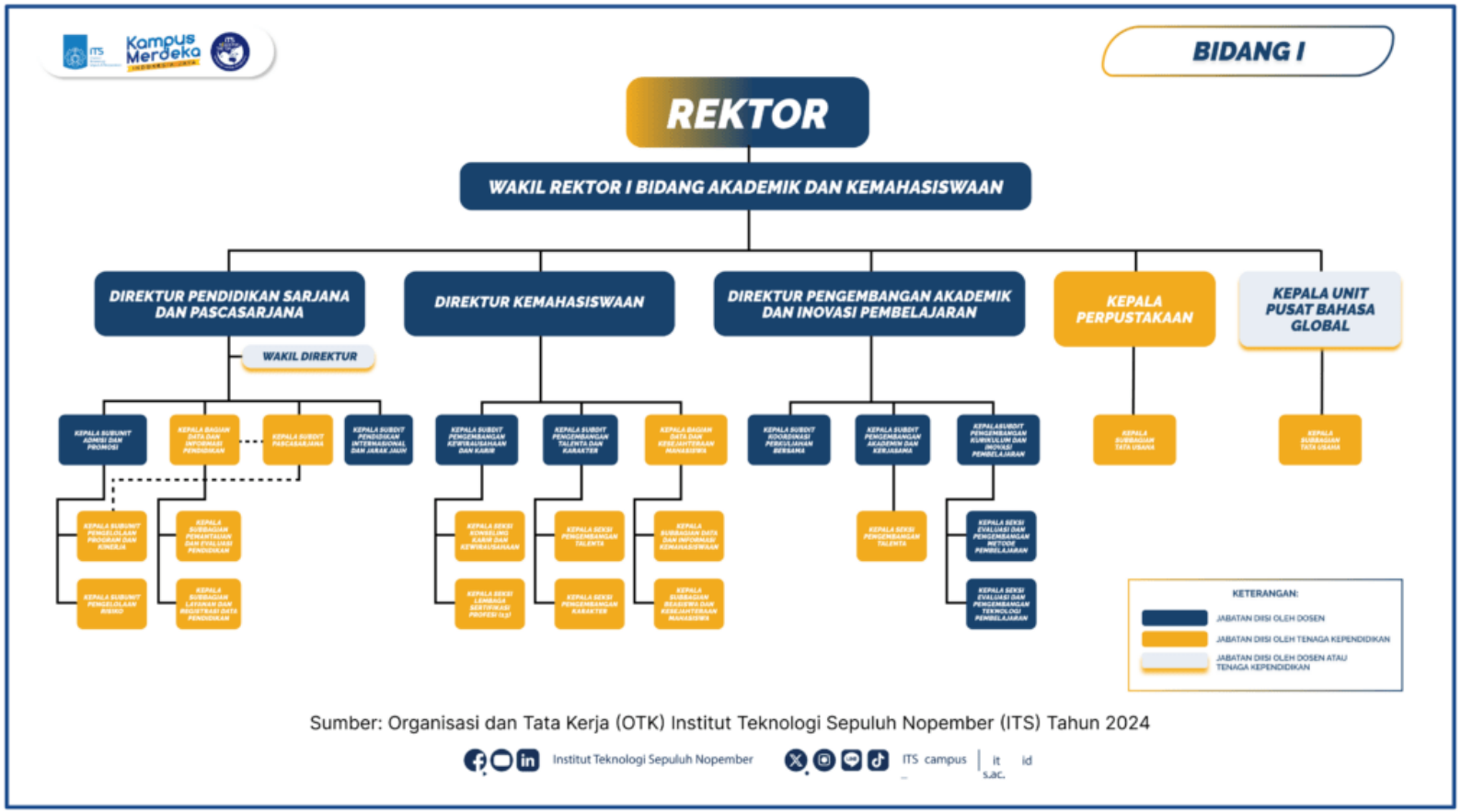
# Contoh Lain

Bagaimana cara menyimpan struktur keanggotaan di suatu organisasi?



# Contoh Lain

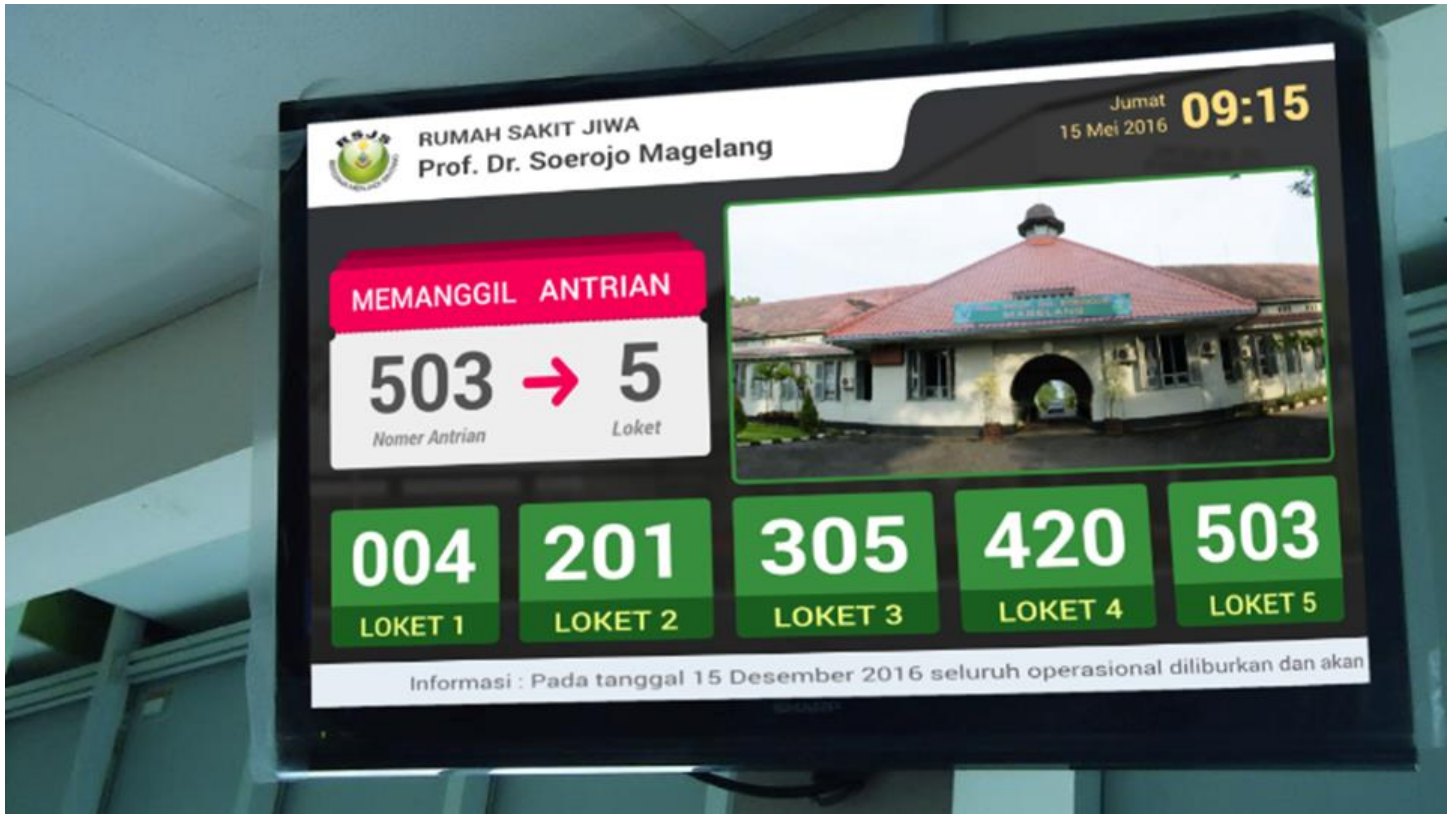
Bagaimana cara menyimpan struktur keanggotaan di suatu organisasi?




Menggunakan **Tree**

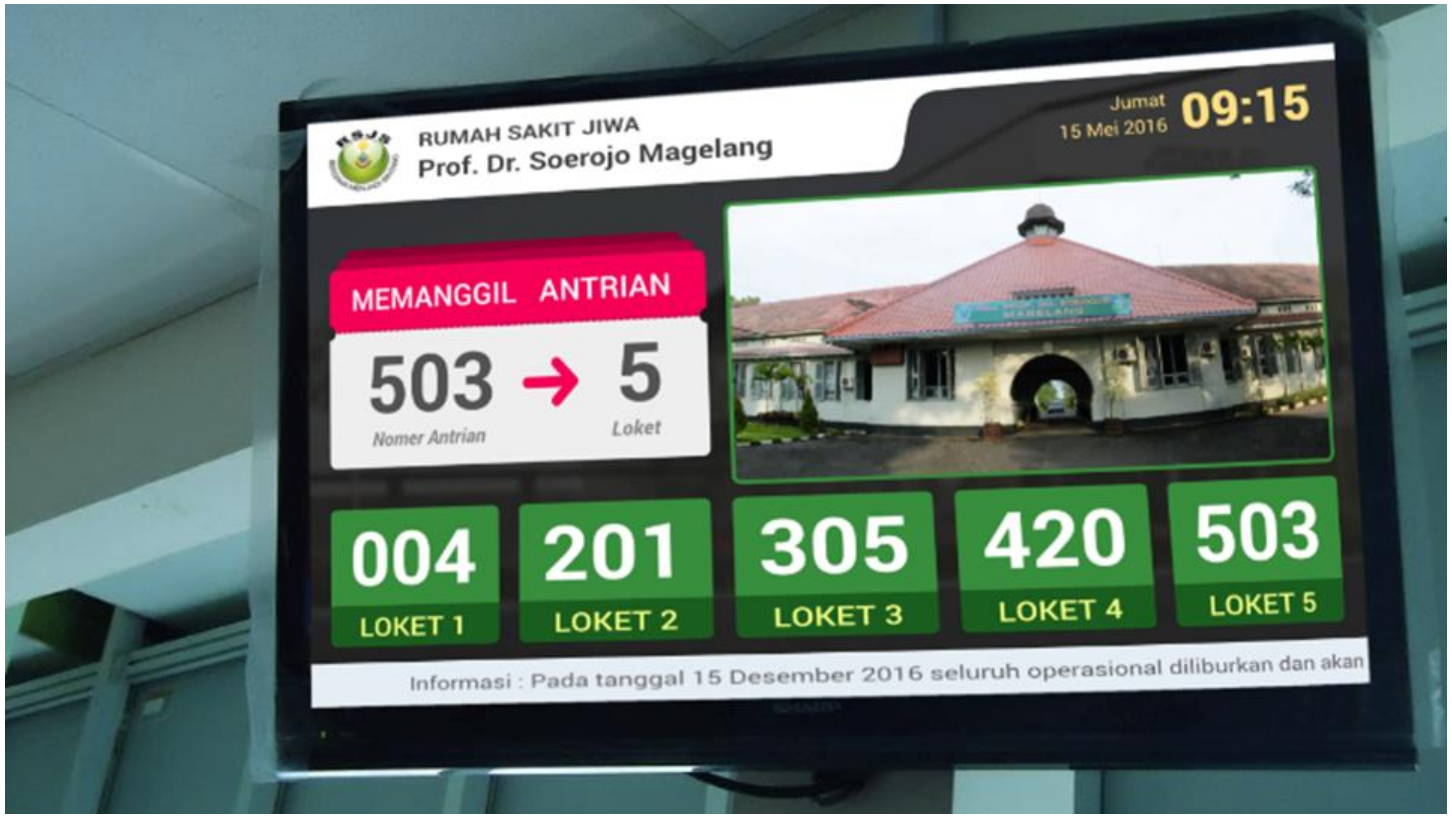
# Contoh Lain

Bagaimana cara menyimpan data antrian di sebuah Rumah Sakit?



# Contoh Lain

Bagaimana cara menyimpan data antrian di sebuah Rumah Sakit?




Menggunakan
**Queue**

# *Pengantar Pemrograman Berorientasi Objek*

# *Pemrograman Berorientasi Objek*

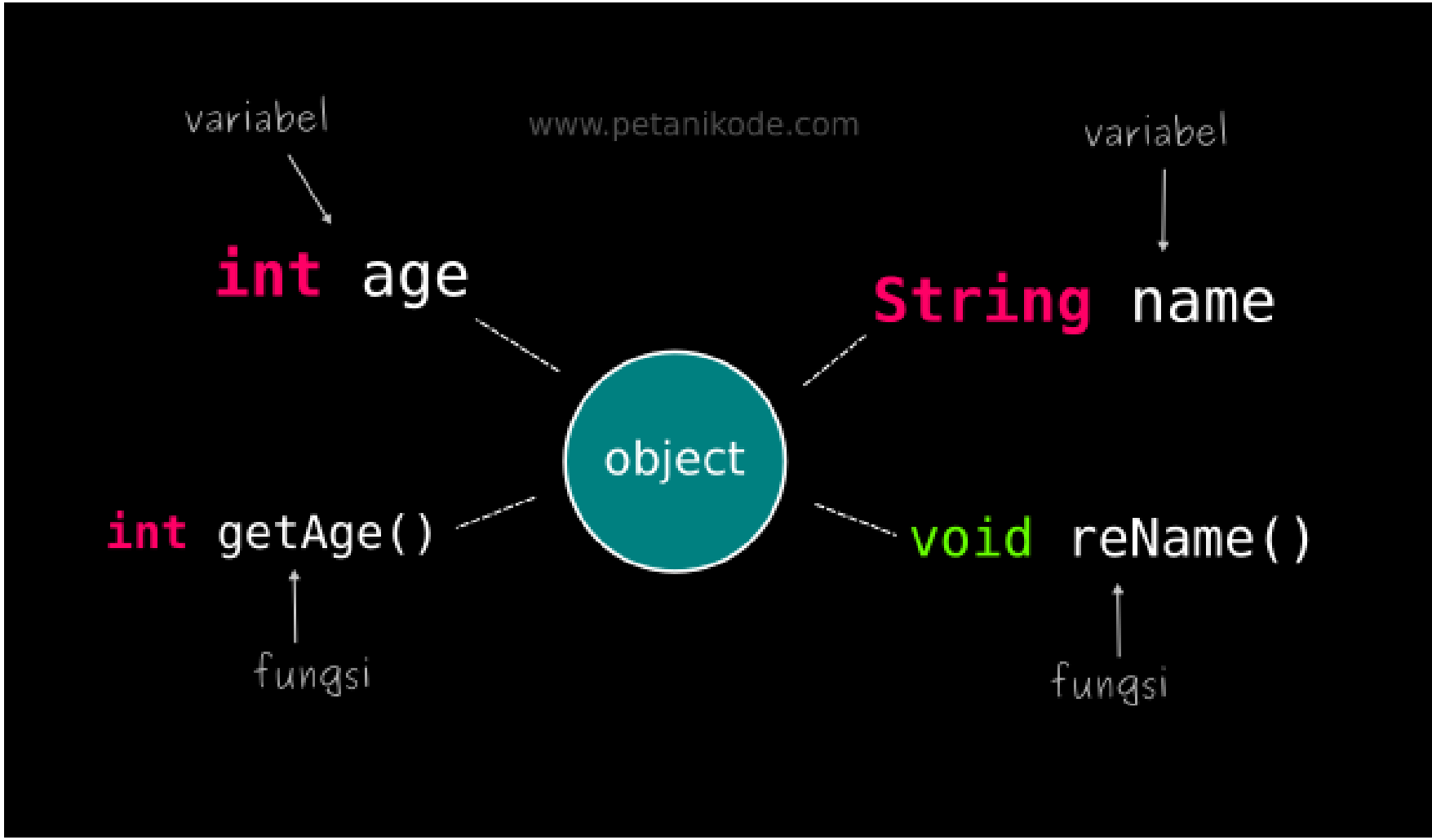
Paradigma pemrograman yang bertujuan untuk **memodelkan kasus-kasus nyata** dalam kehidupan **ke dalam konsep sebuah objek.**

Pada OOP, **fungsi dan variabel dibungkus dalam sebuah objek** yang dapat saling berinteraksi, sehingga membentuk sebuah program.

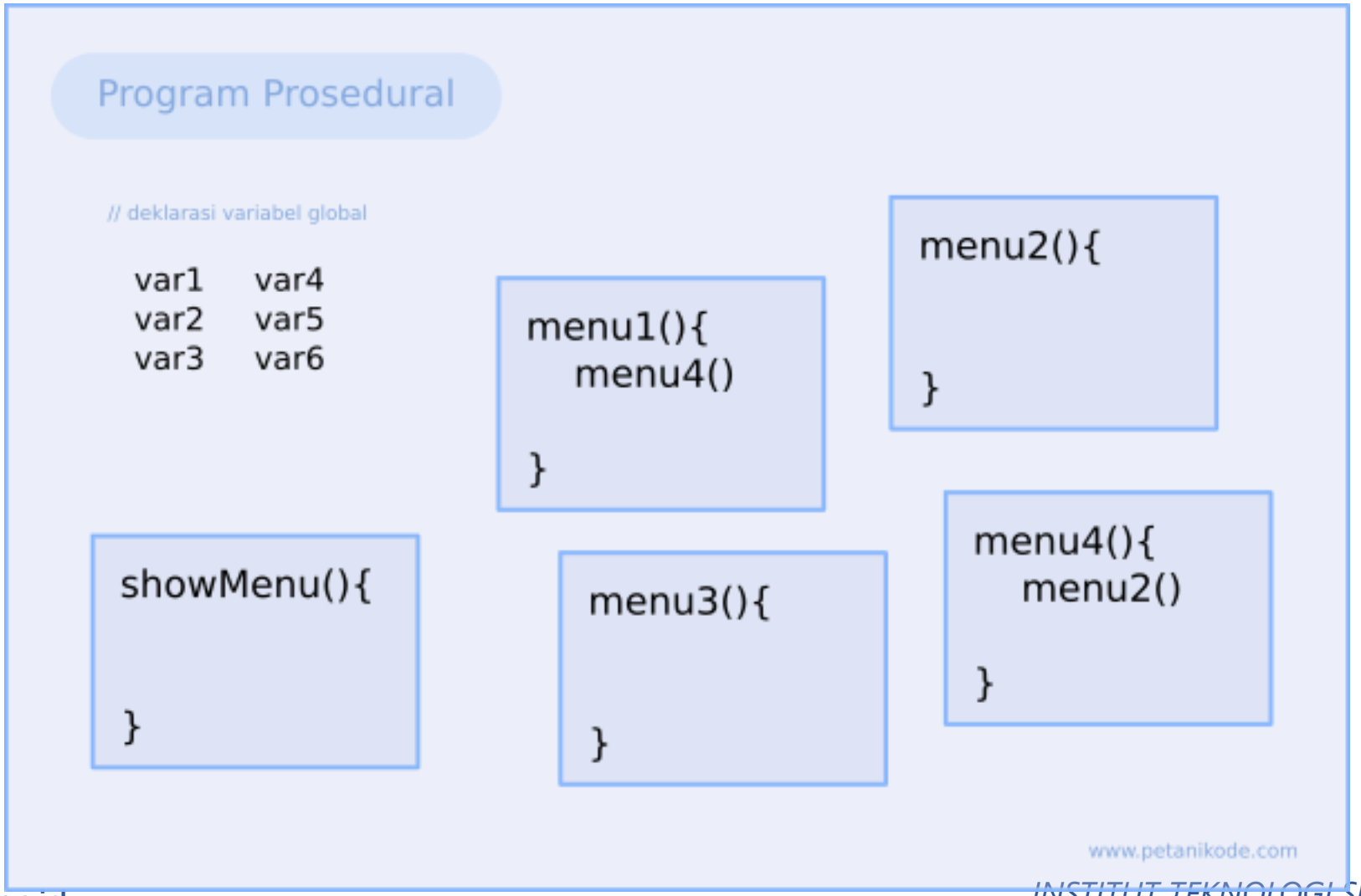
Kunci dalam pemrograman:
- **Sederhana**
- ***Reusable***

# *Pemrograman Berorientasi Objek*

**Fungsi dan variabel dibungkus dalam sebuah objek** yang dapat saling berinteraksi, sehingga membentuk sebuah program.



# *Mengapa menggunakan OOP?*




**Semakin besar dan kompleks program**, maka kode program jadi **sulit di maintenance.**

# Mengapa menggunakan OOP?

## Prosedural

```
var catMood = "happy";
var catEnergy = 80;
var makanan = "Fish";

catMove()
catPlay()
catJump()
catEat()
```

www.petanikode.com

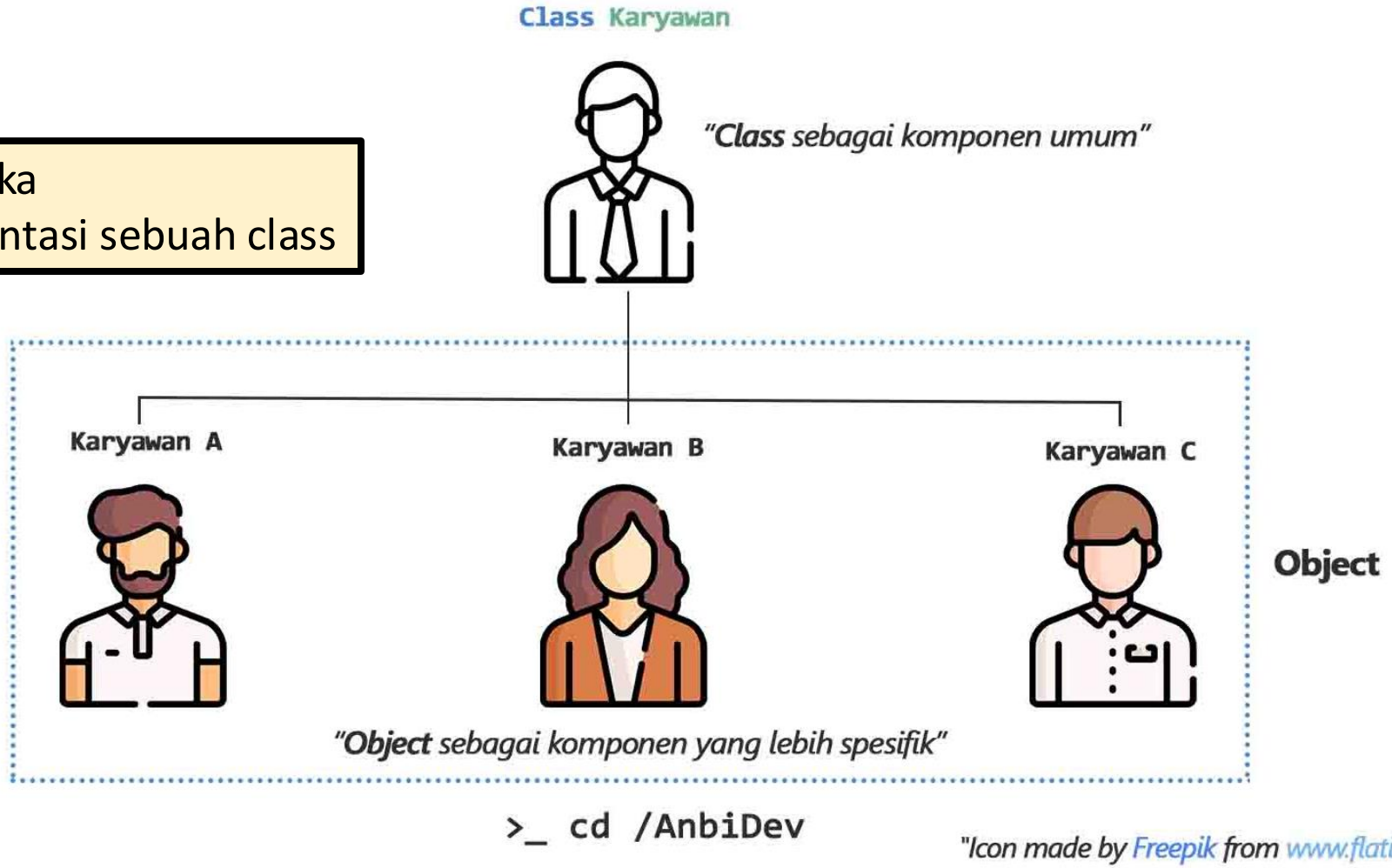
## OOP

```
class Kucing {

    var mood;
    var energi;

    lari()
    loncat()
    makan()
}
```

```
class Makanan {
    var nama;
    var rasa;

    hide()
}
```

# Object vs Class

- **Class** : Blueprint / kerangka
- **Object** : Instance / representasi sebuah class

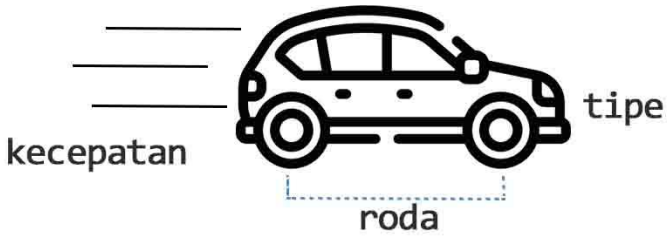



"Icon made by Freepik from www.flaticon.com"

# Object vs Class

@anbidev

**Class : Mobil**

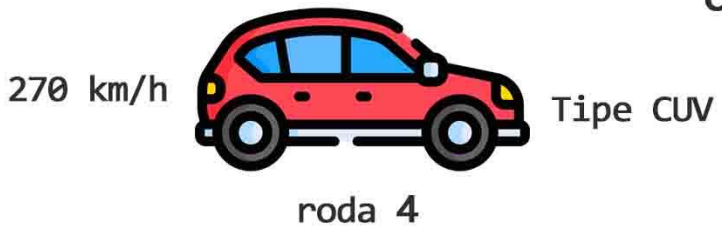



*"**Class** dianggap sebagai rancangan mobil"*

- **Class** : Blueprint / kerangka
- **Object** : Instance / representasi sebuah class

**Object : Mobil Avanza**

*"**Object** adalah hasil dari rancangan tersebut"*




>_ cd /AnbiDev

"Icon made by Freepik from www.flaticon.com"

# Class

**Class** adalah "*template*" untuk membuat sebuah object yang memiliki karakteristik (***attribute***) dan perilaku (***behaviour*** atau ***method***).

## Class Mobil

### Attribute

- tipe
- roda
- kecepatan

### Method

- melaju()
- klakson()
- berbelok()

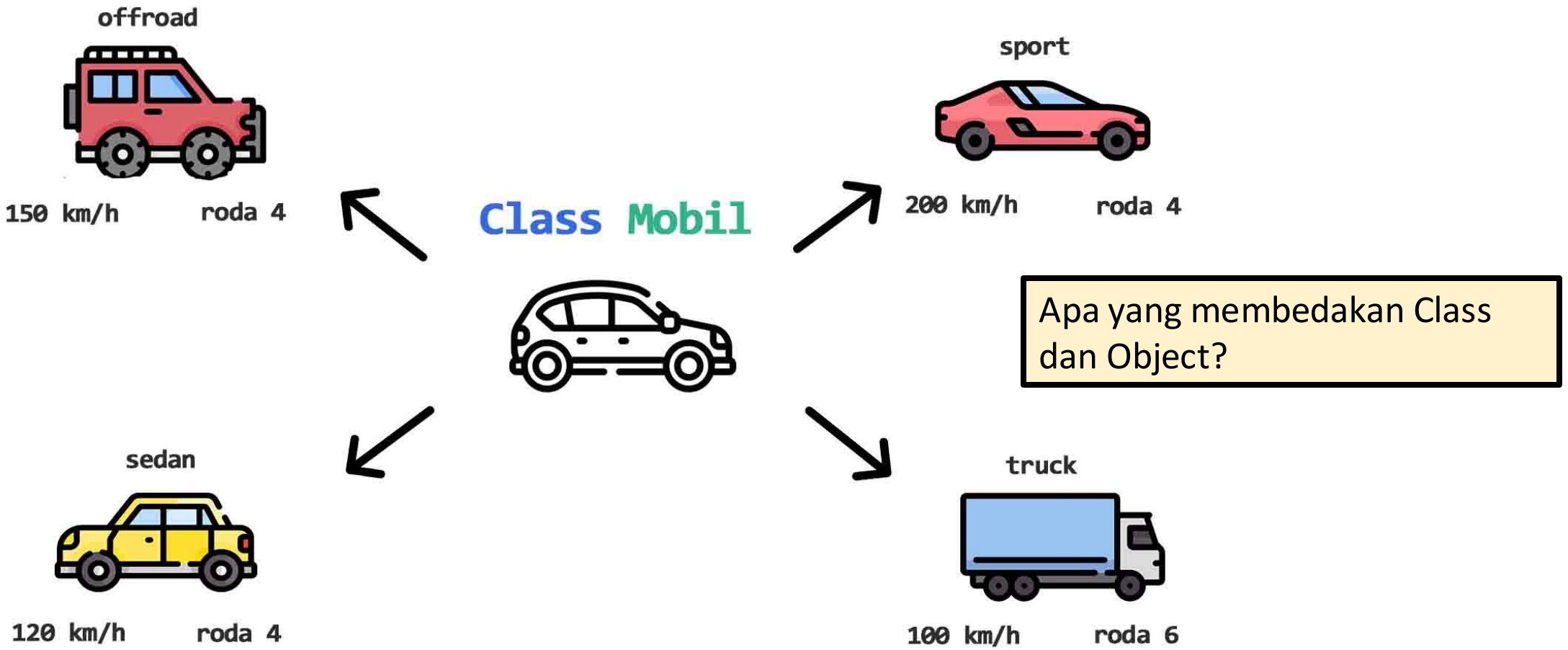
1. Atribut: Apa **ciri-ciri** objek itu?
2. Method: Apa **yang bisa dilakukan** oleh objek itu?

>_ cd /AnbiDev

"Icon made by Freepik from www.flaticon.com"

# Instansiasi

**Instansiasi** adalah proses pembuatan objek dari sebuah class




Apa yang membedakan Class dan Object?

>_ cd /AnbiDev

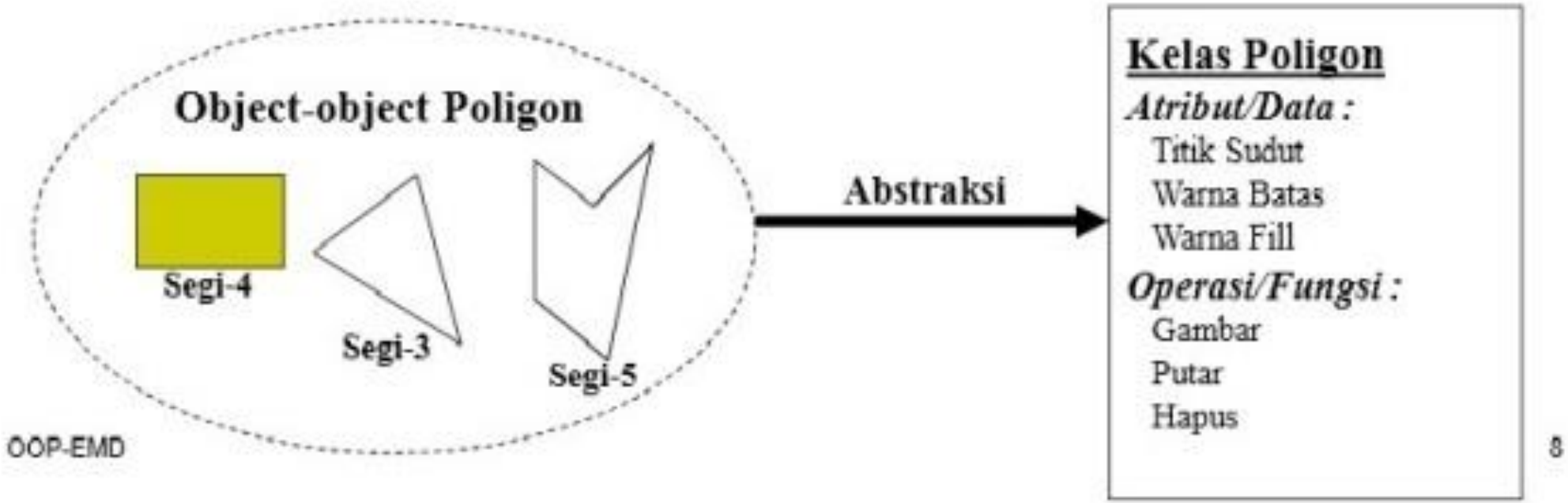
# *Konsep Dasar OOP*

- **Abstraksi** *(Abstraction)*
- **Pembungkusan** (*Encapsulation*)
- **Pewarisan** *(Inheritance)*
- **Polimorfisme** (*Polymorphysm*)

# Abstraction

Suatu **proses penyembunyian kerumitan (pengabstrakan)** yang terjadi dalam suatu objek sehingga pengguna objek tidak perlu mengetahui detail proses yang dilakukan.

- **Menemukan** hal-hal yang **essensial / general**
- **Mengabaikan** hal-hal yang **insidental**



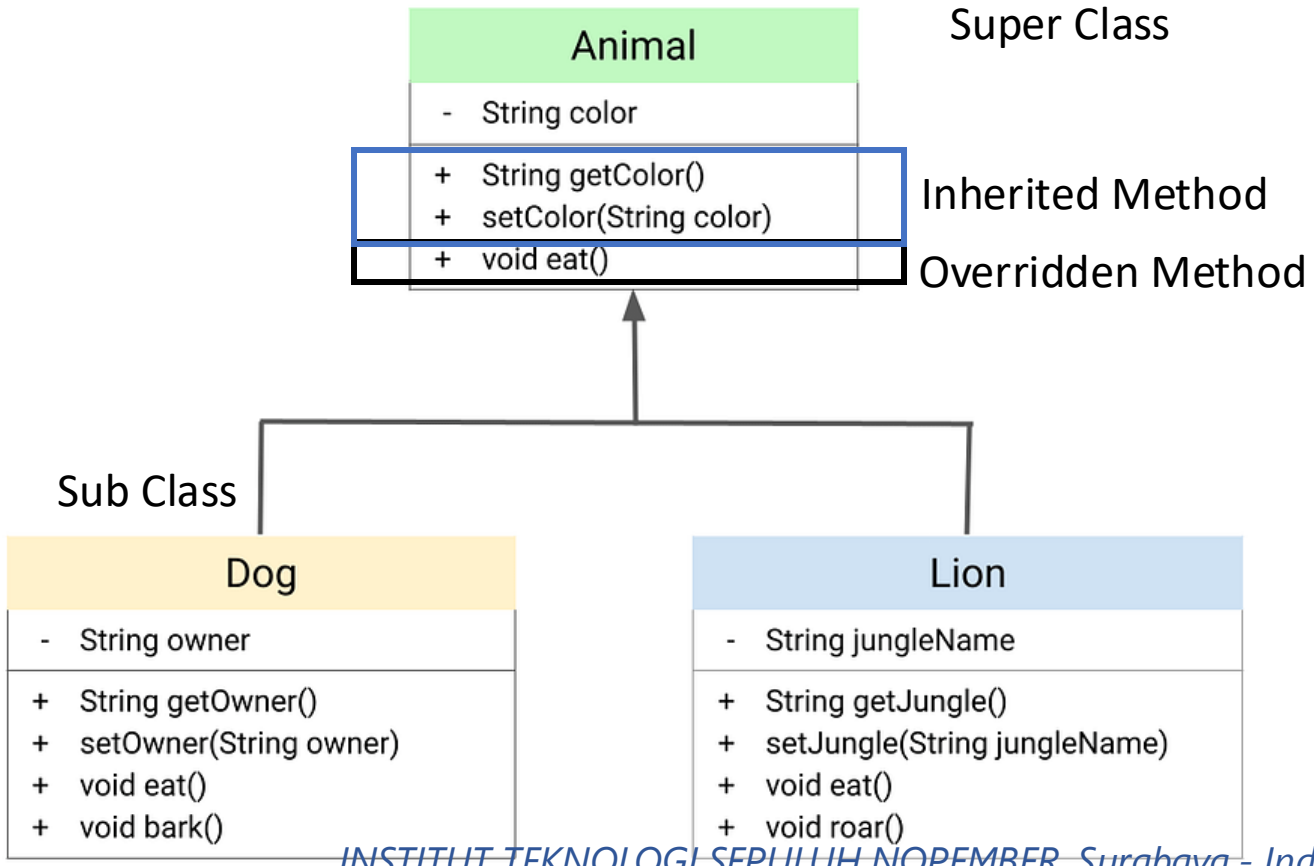
# *Encapsulation*

**Attribute dan method akan dibungkus menjadi objek** yang merupakan satu kesatuan sehingga dapat bekerjasama dalam melaksanakan tugas-tugas pemrograman tertentu.

# *Inheritance*

Sebuah objek dapat diturunkan menjadi objek baru lainnya, sehingga **objek baru tersebut dapat mewarisi sifat-sifat dari objek induknya.**

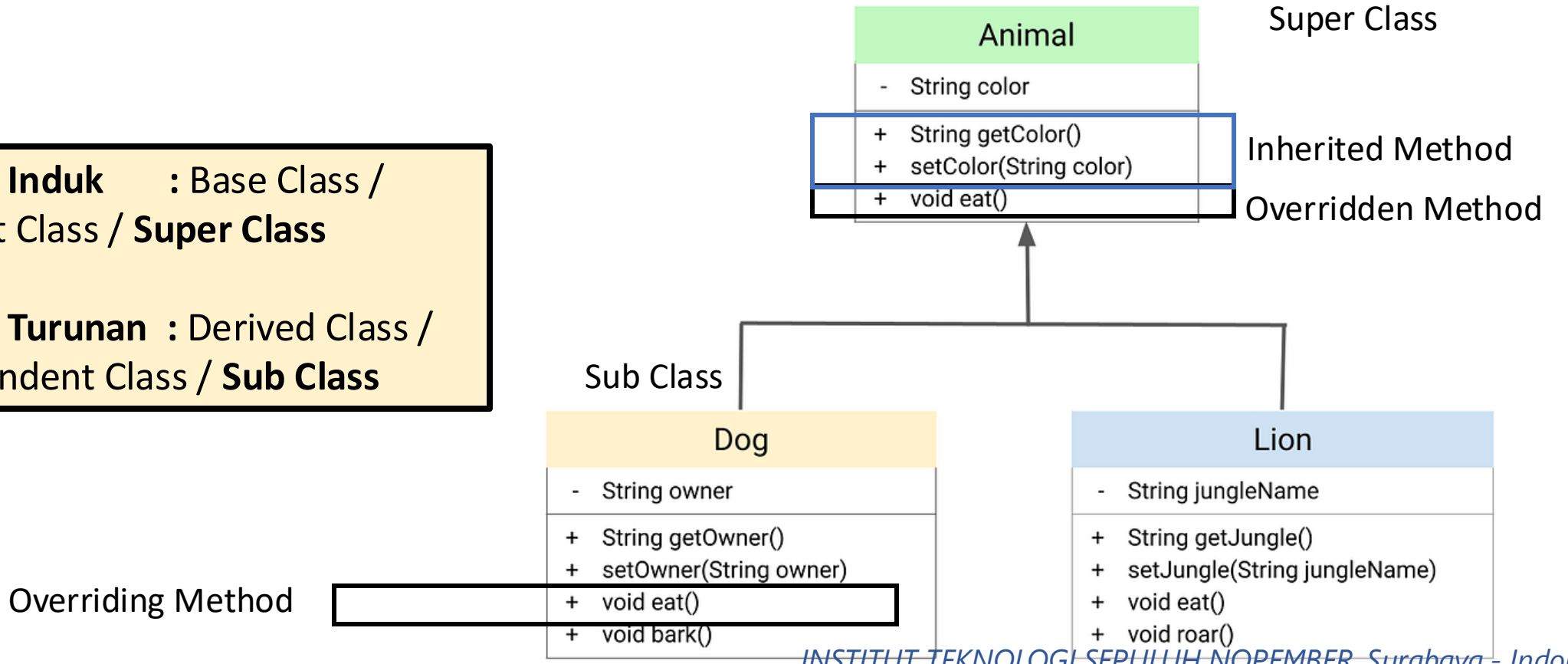
- **Objek Induk** : Base Class / Parent Class / **Super Class**
- **Objek Turunan** : Derived Class / Descendent Class / **Sub Class**



# *Inheritance*

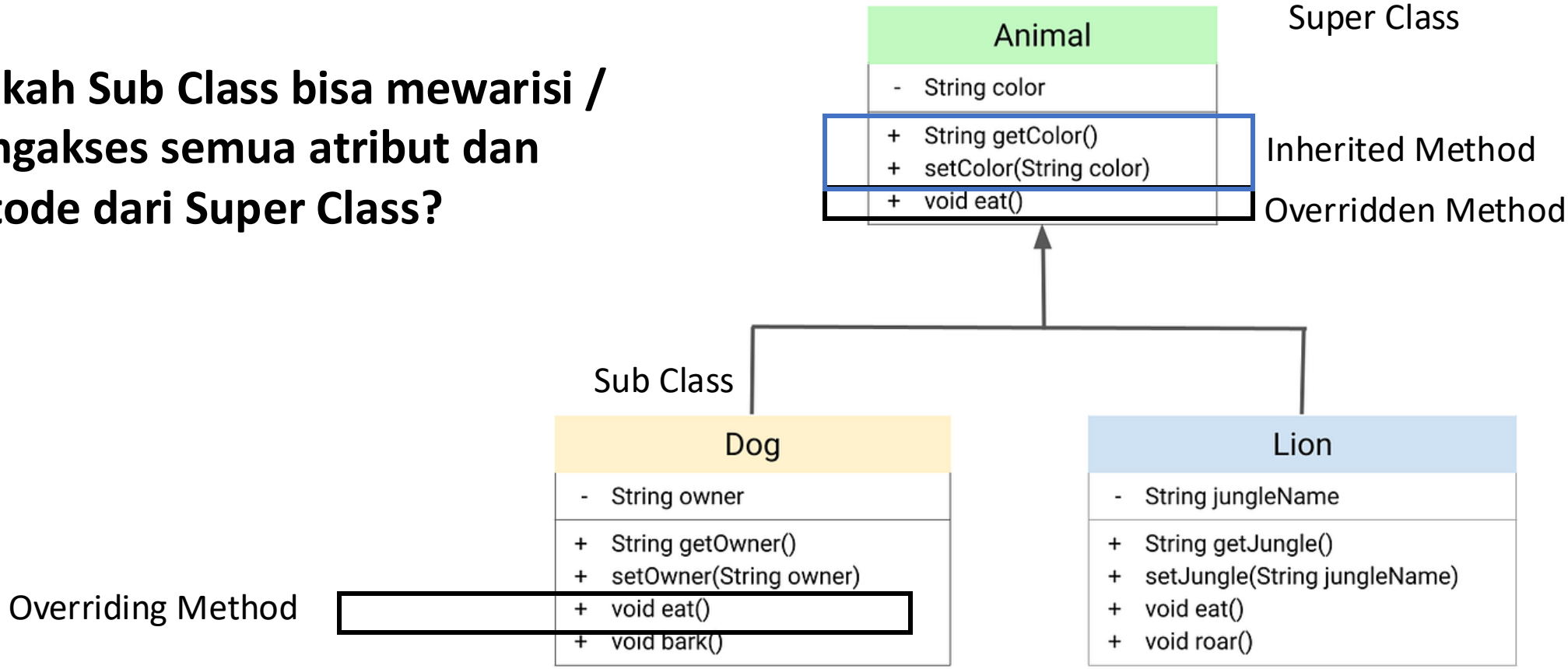
Sebuah objek dapat diturunkan menjadi objek baru lainnya, sehingga **objek baru tersebut dapat mewarisi sifat-sifat dari objek induknya.**

- **Objek Induk** : Base Class / Parent Class / **Super Class**
- **Objek Turunan** : Derived Class / Descendent Class / **Sub Class**



# Inheritance

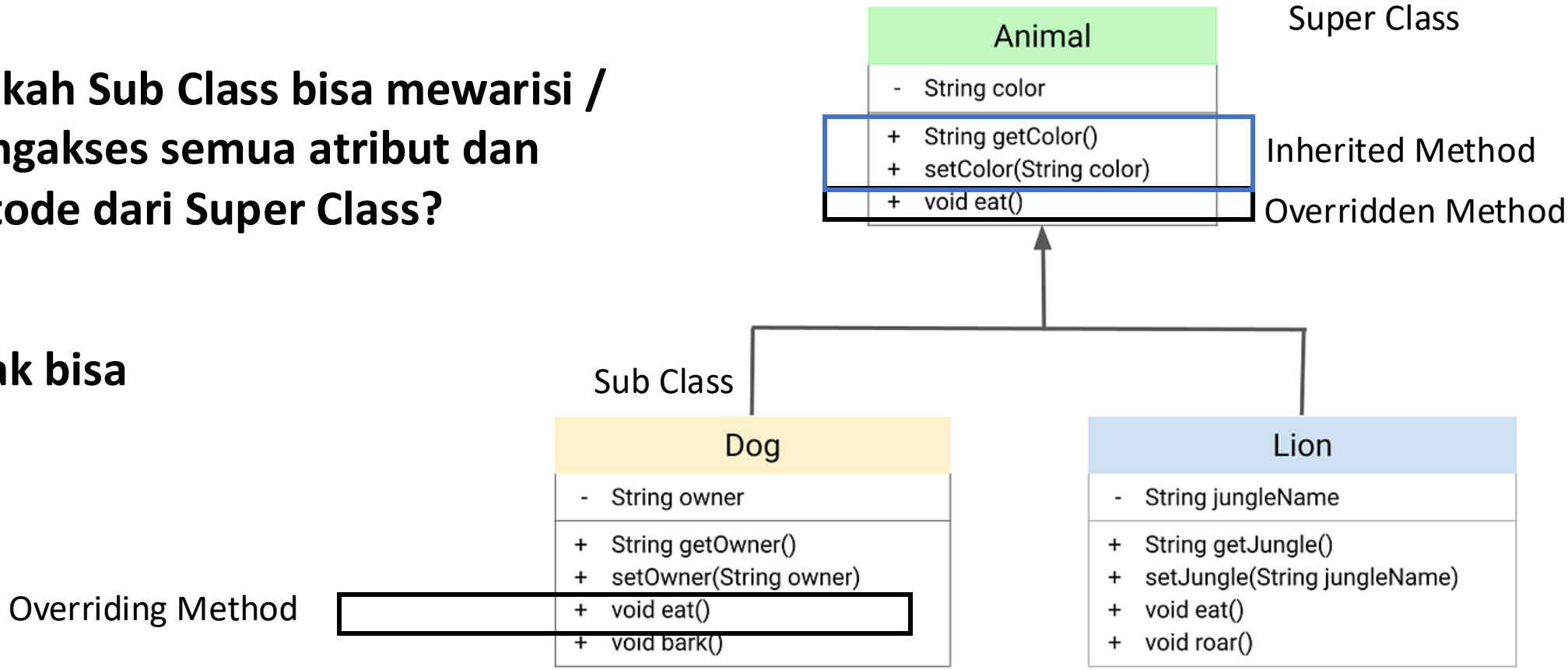
**Apakah Sub Class bisa mewarisi / mengakses semua atribut dan metode dari Super Class?**



# *Inheritance*

**Apakah Sub Class bisa mewarisi / mengakses semua atribut dan metode dari Super Class?**

**Tidak bisa**

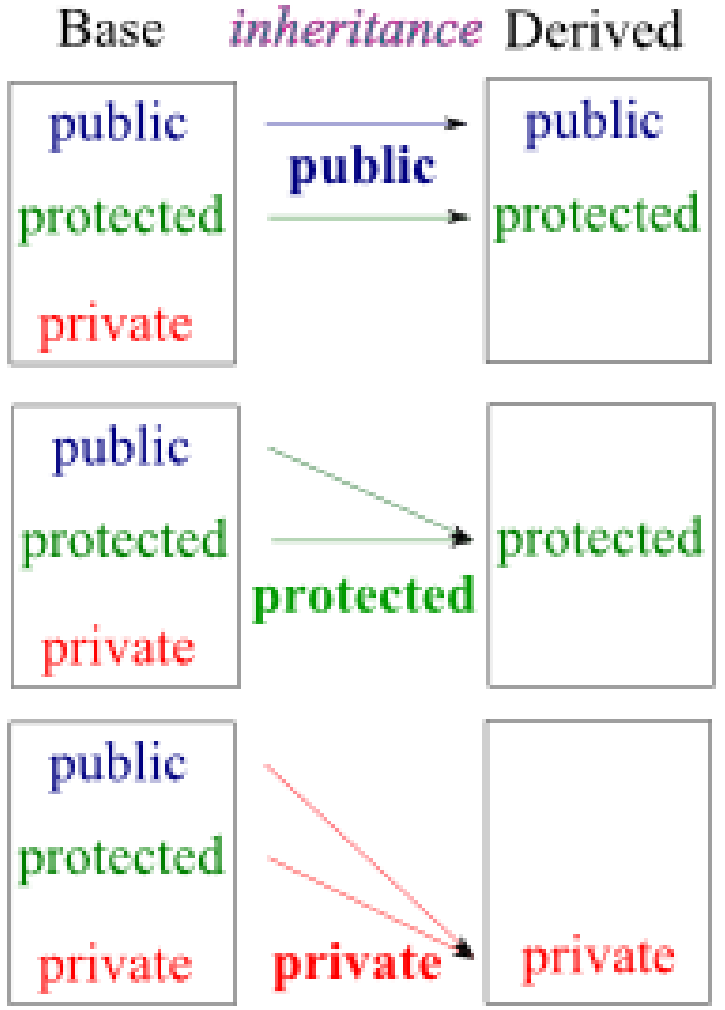


# Access Modifiers (Modifier Akses)

Untuk membatasi hak akses **attribute** dan **method:**

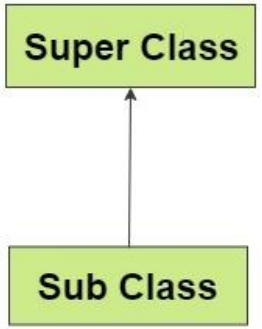
**Public, Protected** dan **Private**

| | Class Itself | Outside Class | Derived Class |
|---|---|---|---|
| Public | ✓ | ✓ | ✓ |
| Protected | ✓ | ✗ | ✓ |
| Private | ✓ | ✗ | ✗ |

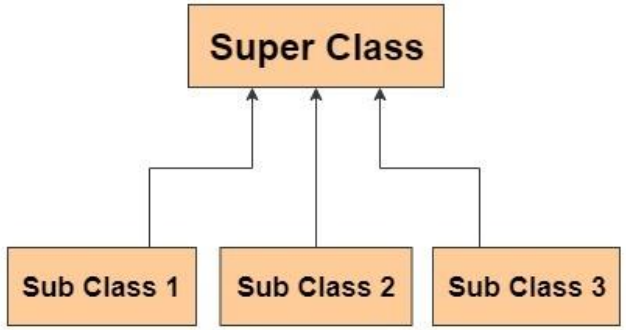


# Inheritance

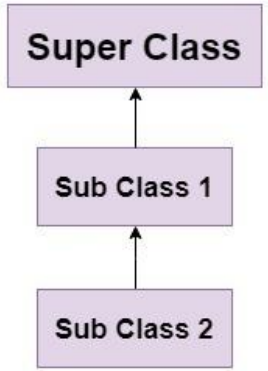
Single Inheritance



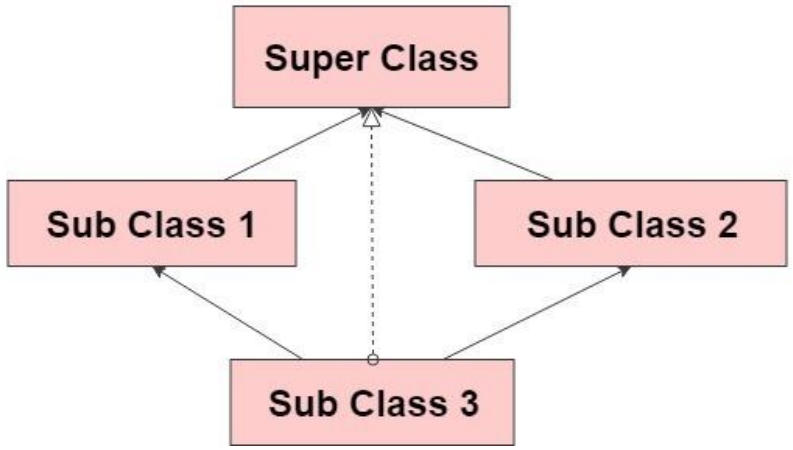

Hierarchial Inheritance



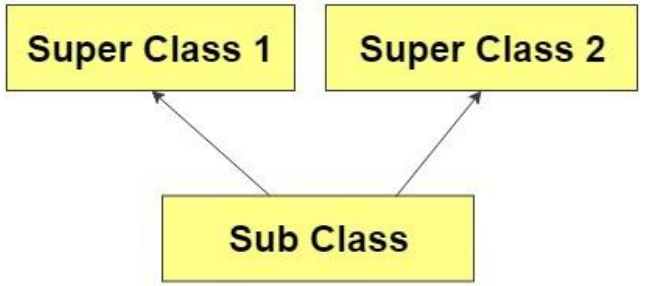

MultiLevel Inheritance



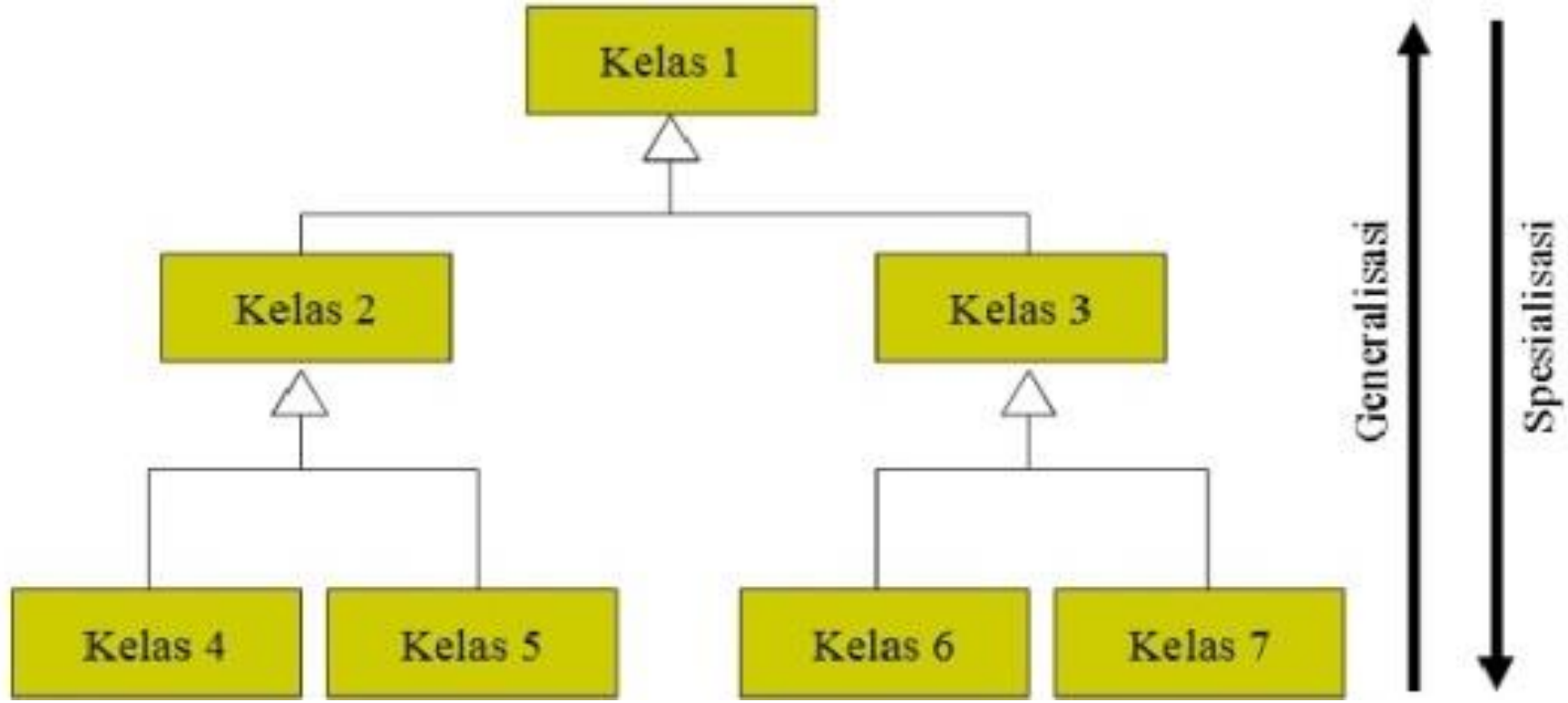

Hybrid Inheritance




Multiple Inhertance

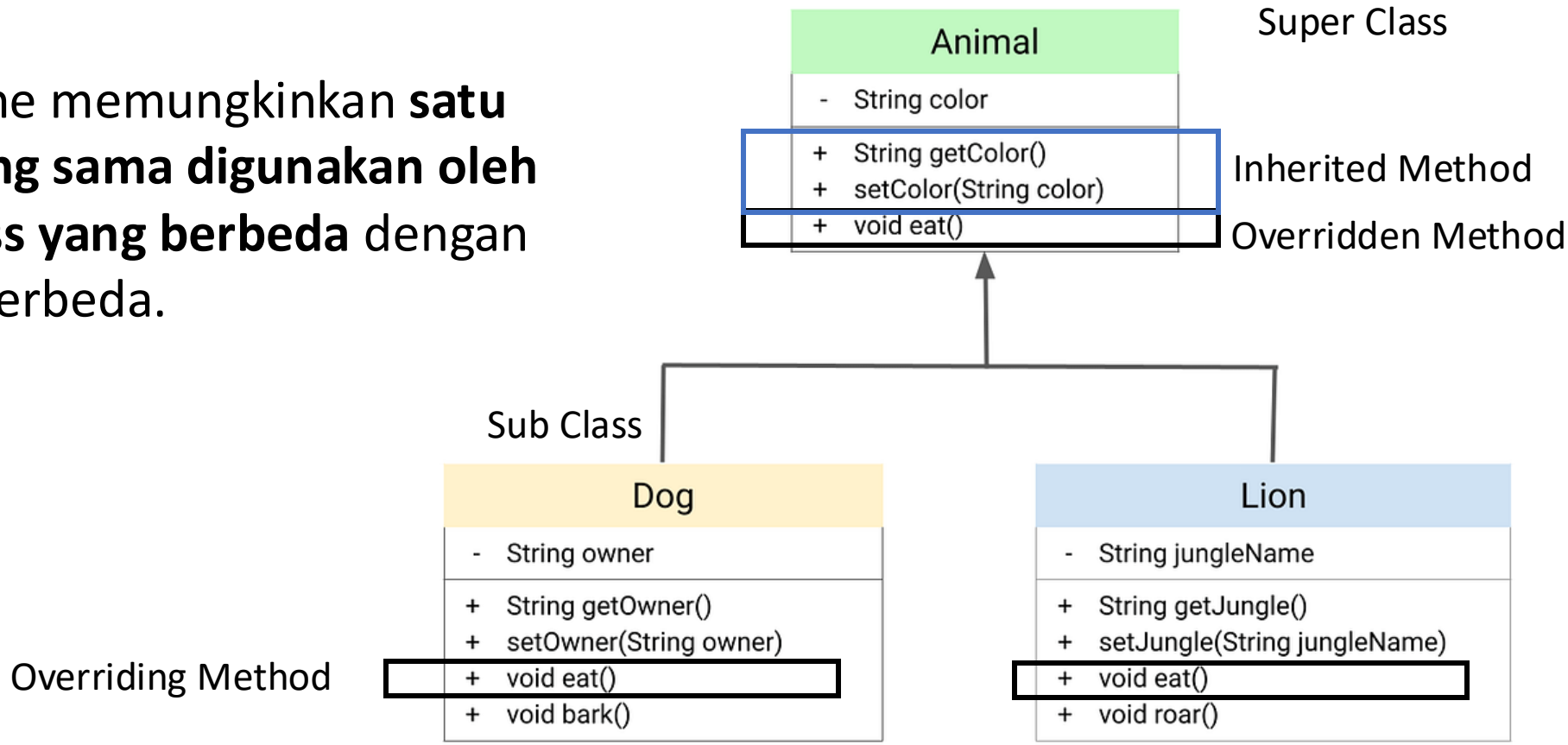


# Spesialisasi dan Generalisasi



# *Polymorphysm*

Polimorfisme memungkinkan **satu metode yang sama digunakan oleh banyak class yang berbeda** dengan cara yang berbeda.



## *Latihan 1*
## *Bahasa: C++*

Buatlah class **MHS** yang memiliki :

- Attribute:
  - string **MHSname** untuk menyimpan nama mahasiswa
- Method:
  - **void printname()** menampilkan nama mahasiswa
- Setelah itu, instansiasi objek **mhs1**
- Isi nama mahasiswa sebagai “Rahmat Budi”
- Tampilkan nama mahasiswa

**Apa kelemahan dari class yang dibuat tsb?**

```
#include <iostream>
using namespace std;

class MHS
{
   // Access specifier
   public:

   // Data Members
   string MHSname;

   // Member Functions()
   void printname()
   {
      cout << "MHS name is: " << MHSname;
   }
};

int main() {

   // Declare an object of class geeks
   MHS mhs1;

   // accessing data member
   mhs1.MHSname = "Rahmat Budi";

   // accessing member function
   mhs1.printname();
   return 0;
}
```

## *Latihan 2*
## *Bahasa: C++*

a. Modifikasilah class tsb sehingga akses ke atribut **HANYA** dapat dilakukan melalui method di dalam kelas. Apa yang pelu diubah / ditambahkan?
b. Tambahkan method baru sehingga pada waktu create objek baru dapat dilakukan pengisian nama mahasiswa.

   Misal digunakan dengan cara : **MHS mhs2("Muhammad Ali");**

c. Tampilkan nama kedua mahasiswa dengan **function printname()**
d. Tambahkan member function **string getNama()**, untuk mendapatkan nama mahasiswa.
e. Tampilkan nama kedua mahasiswa tanpa menggunakan function printname()