

# Daftar ISI

[Daftar ISI 2](#_heading=h.gjdgxs)

[1.](#_heading=h.30j0zll) OOP 4

[a.](#_heading=h.1fob9te) Apa itu Class? 5

[b.](#_heading=h.3znysh7) Apa itu Attribut ? 7

[c.](#_heading=h.2et92p0) Apa itu Construktor ? 8

[d.](#_heading=h.tyjcwt) Apa itu method/fungsi ? 9

[e.](#_heading=h.3dy6vkm) Apa itu setter getter ? 10

[f.](#_heading=h.1t3h5sf) Apa itu Object? 12

[g.](#_heading=h.4d34og8) Kode Program Lengkap 13

[2.](#_heading=h.2s8eyo1) Cara Kerja OOP (*Object-Oriented Programming*) 17

[3.](#_heading=h.17dp8vu) Manfaat OOP 18

[Daftar Pustaka](#_heading=h.2u6wntf) 19

# OOP

**Apa itu OOP ?**

OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek.

Ini digunakan untuk menyusun program *software* (perangkat lunak) menjadi potongan-potongan sederhana dari cetak biru kode (biasanya disebut *class*) yang digunakan untuk membuat *instance* objek secara individu.

Karena OOP merupakan paradigma pemrograman, ada banyak bahasa pemrograman yang berorientasi objek termasuk seperti C++, Java, Python dan Ruby serta lainnya.

**Apa tujuan OOP ?**

Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Tujuan OOP juga untuk mempermudah pengembangan program.

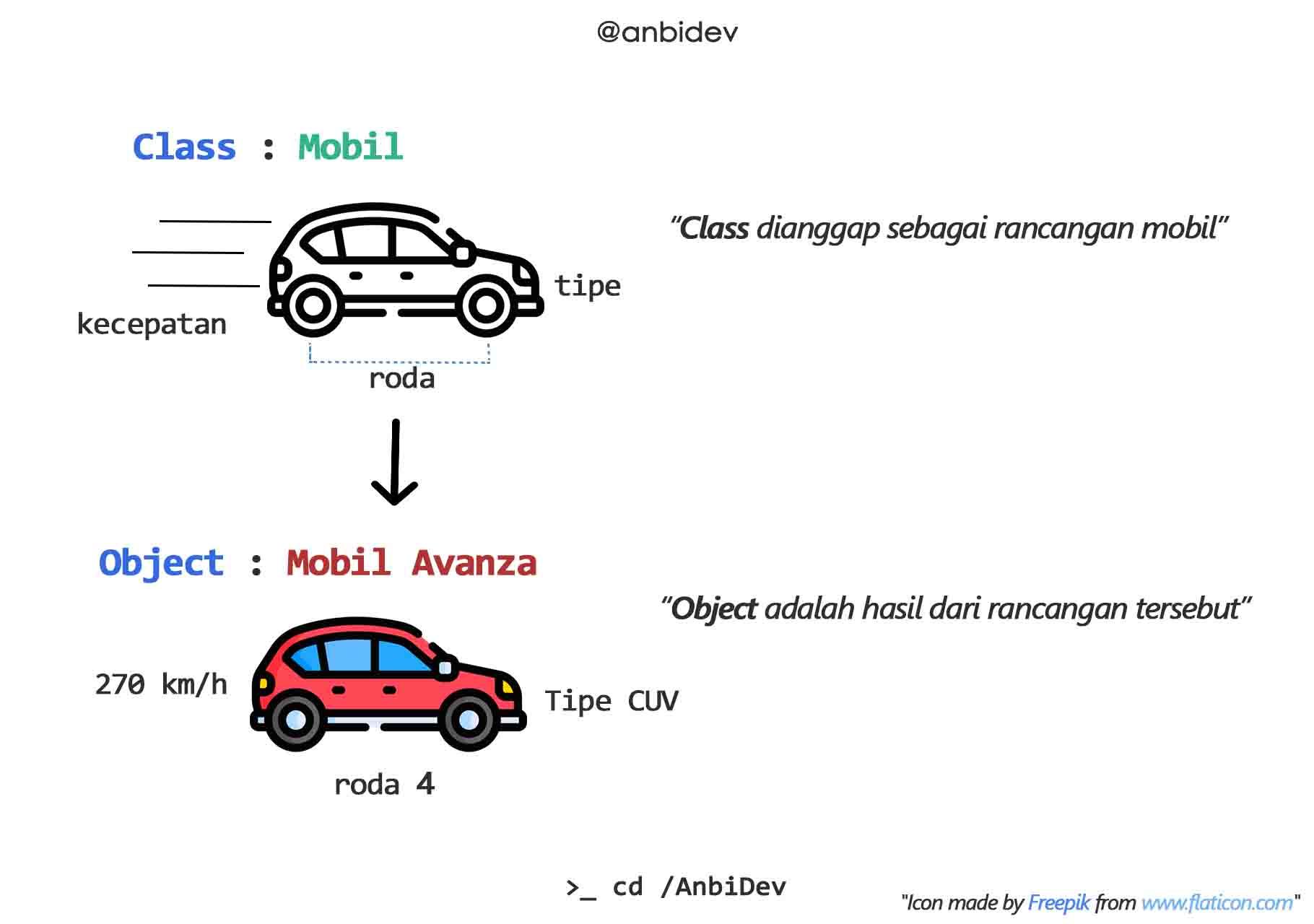
Seorang *programmer* merancang program perangkat lunak dengan mengatur informasi dan perilaku yang terkait bersama-sama ke dalam templat yang disebut kelas.

Kemudian objek individual dibuatkan atau dibuat dari *template*  kelas, benda-benda ini biasanya mewakili benda dunia nyata.

Seluruh program perangkat lunak dijalankan dengan menyuruh banyak objek berinteraksi dengan objek untuk membuat program yang lebih besar.

*Object-Oriented Programming* (OOP) adalah model bahasa pemrograman di mana program disusun di sekitar data, atau objek, bukan fungsi dan logika.

Objek dapat didefinisikan sebagai bidang data yang memiliki atribut dan perilaku yang *unique* (unik).



Contoh Analogi OOP

Contohnya seperti ini, anggap kita punya Class **mobil**. Ia memiliki attribute

* **roda**,
* **kecepatan** dan
* **tipe**.

Serta didalam mobil dia juga memiliki method :

* **melaju**(),
* **klakson**() dan
* **berbelok**().

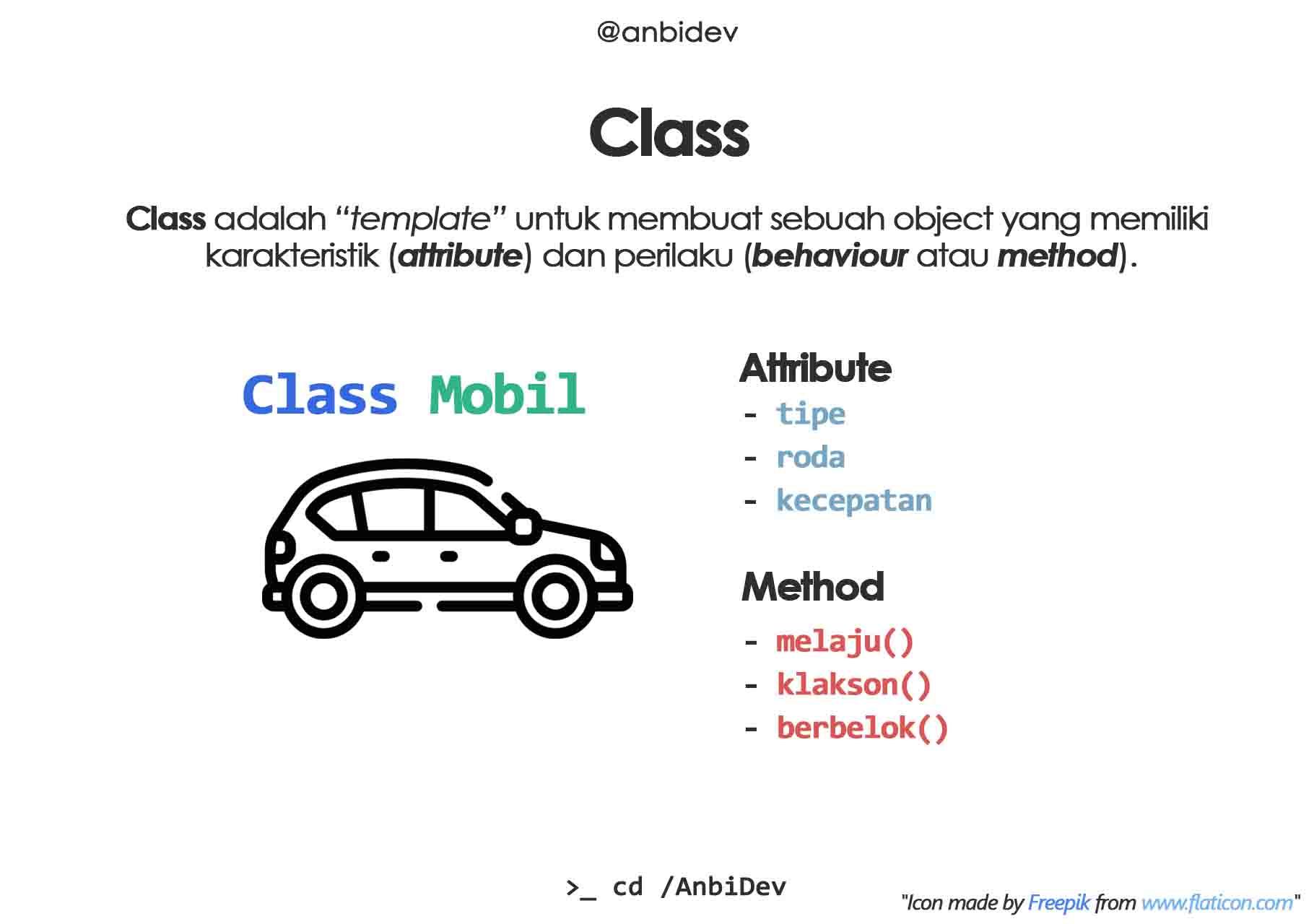
Jika Class dalam perumpamaannya adalah blueprint dari suatu kategori yang umum, maka object diumpamakan sebagai hal yang lebih spesifik. Jadi, class **mobil** tadi bisa membuat banyak objek yang lebih spesifik, misal mobilAvanza atau mobilFerari.

OOP karena merupakan objek, maka objek ini memiliki **class, atribut, method**. Mari kita simak apa class, atribut, method seperti dibawah ini.

## Apa itu Class?

**Class** adalah ‘cetakan’, ‘template’ atau ‘blueprint’ untuk membuat sebuah **object**. Class hanya sebagai kerangka dasar dari object. Sehingga nanti cetakan (class) tersebut dapat membuat berbagai macam objek untuk kebutuhan yang berbeda - beda.

Class diibaratkan sebagai kategori yang umum, seperti yang dijelaskan diatas. Anggaplah kita membuat Class bernama "mobil", didalam class tersebut terdapat karakteristik (attribute) dan perilaku (method) yang hanya dimiliki oleh **mobil**.



Gambar Analogi OOP class

Kita tahu bahwa mobil memiliki roda, tipe atau kecepatan mobil. Hal ini bisa kita sebut dengan attribute. Lalu kita juga tahu bahwa mobil juga memiliki fungsi untuk jalan melaju, klakson dan berbelok.

Dalam ilustrasi diatas, kita bisa menyimpulkan seperti apa bentuk class **mobil**. Dan selanjutnya, bagaimana bentuk class tersebut, jika ditranslasikan kedalam bentuk code java.

| public class Mobil {   String type;  int roda;  Double kecepatan;   public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }   public void doMelaju(String kecepatan){  System.*out*.println("Melaju dengan kecepatan : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  } } |
| --- |

## Apa itu Attribut ?

**Attribute** atau **properties** merupakan karakteristik dari sebuah

class. Attribute ini berupa suatu variabel didalam sebuah class. **Variabel** yang didefinisikan sebagai sebuah attribute juga disebut dengan variabel Global.

**Bagaimana Cara Deklarasi Attribut ?**

Cara mendeklarasikan sebuah attribute sama dengan cara mendeklarasikan sebuah variabel. Dalam java, variabel yang digunakan sebagai attribute biasanya didefinisikan dalam fungsi constructur (type data\_). Variabel **attribute** pada class disebut juga dengan instance (turunan) variable.

**Kenapa harus pakai constructor?**

karena instance variabel hanya bisa didefinisikan hanya dari sebuah method.

Tadi kita sudah sedikit menyinggung tentang *attribute* yang dimiliki oleh mobil (tipe, roda dan kecepatan). Dalam kode yang dibuat diawal.

| String type; int roda; Double kecepatan; |
| --- |

Pada kode diatas kita mendeklarasikan *attribute* dengan nama variabel “**type**,**roda**” bertype String dan variabel “**kecepatan**” bertype Double.

## Apa itu Construktor ?

Constructor merupakan suatu method yang akan memberikan nilai awal pada saat suatu objek dibuat. Pada saat program dijalankan, constructor akan langsung memberikan nilai awal pada saat perintah new, membuat suatu objek. Pada saat kita bekerja dengan constructor, hal mendasar yang perlu diperhatikan, **yaitu** :

* Nama Constructor sama dengan nama Class.
* Tidak ada return type yang diberikan kedalam Constructor Signature.
* Tidak ada return stetement, didalam tubuh constructor.

Jika kita lihat pada kode class **mobil** diatas, maka kodingan yang menunjukkan construktor adalah sebagai berikut :

| public class Mobil {   String type;  int roda;  Double kecepatan;   public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }  …...  } |
| --- |

Dapat kita lihat, nama kelas dan nama construktor nya sama , yaitu mobil. Dan tidak memiliki return nilai.

## Apa itu method/fungsi ?

Method adalah fungsi yang didefinisikan dalam suatu **class**. Biasanya method memiliki hubungan dalam behaviour atau perilaku kelas tersebut. Dalam kasus class **mobil**, dapat mendefinisikan 3 method yaitu :

* **melaju,**
* **klakson**dan
* **berbelok.**

Contoh penulisan method dalam java seperti dibawah ini :

| public class Mobil { ….  public void doMelaju(String kecepatan){  System.*out*.println("Melaju dengan kecepatan : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  }  …. } |
| --- |

Method diatas adalah jenis method **void,** artinya method yang tidak mengembalikan nilai. Adapun **jenis** metho pada ada java aad 2, yaitu :

* **void Method**
* **return Method**

**void** **method** adalah method yang tidak mengembalikan suatu nilai. Dilihat sekilas, void method dapat diidentifikasi dengan adanya kata kunci “void” di depan nama method. Contoh nya terlihat pada tabel kode program diatas yaitu **doMelaju, doKlakson, doBelok.**

**Terus, apa itu return method ?**

return method adalah method yang mengembalikan nilai. Method ini bisa diidentifikasi dengan adanya data type di depan nama method-nya dan kata kunci return di dalam method-nya. Contoh:

| //Return Method public double hitungKecepatan(Double kecepatan, int waktuDetik){  return kecepatan/waktuDetik; } |
| --- |

Method hitungKecepatan merupakan return method dengan type data double, ditandai dengan adanya return di body methodnya.

## Apa itu setter getter ?

Method setter dan getter adalah dua method yang tugasnya untuk mengambil dan mengisi data ke dalam objek.

Dalam OOP kita sering mendengar istilah *encapsulation* (pembungkusan), di mana data dibungkus dengan modifier **private** agar tidak bisa diakses secara langsung dari luar class. Nanti akan kita bahas apa itu *encapsulation*. Contoh method setter dan getter jika mengacu pada class mobil pada variabel nya adalah sebagai berikut :

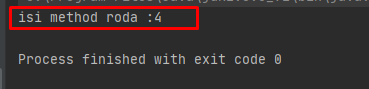
| //Setter and Getter public String getType() {  return type; }  public void setType(String type) {  this.type = type; }  public int getRoda() {  return roda; }  public void setRoda(int roda) {  this.roda = roda; }  public Double getKecepatan() {  return kecepatan; }  public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan; } |
| --- |

Pada potongan kode diatas, setiap method ditandai dengan adanya keyword **set** dan **get**. Misalnya setRoda dan getRoda. Artinya setRoda(“roda 1”), variabel roda akan berisi nilai “roda 1”. Jika String ambilRoda= getRoda(); , artinya varibale ambilRoda akan berisi nilai dari method getRoda();

**Contoh**

| public class MainMobil {  public static void main(String[] args){  Mobil mobil= new Mobil();  mobil.setRoda(4);  System.*out*.println("isi method roda :"+mobil.getRoda());  } } |
| --- |

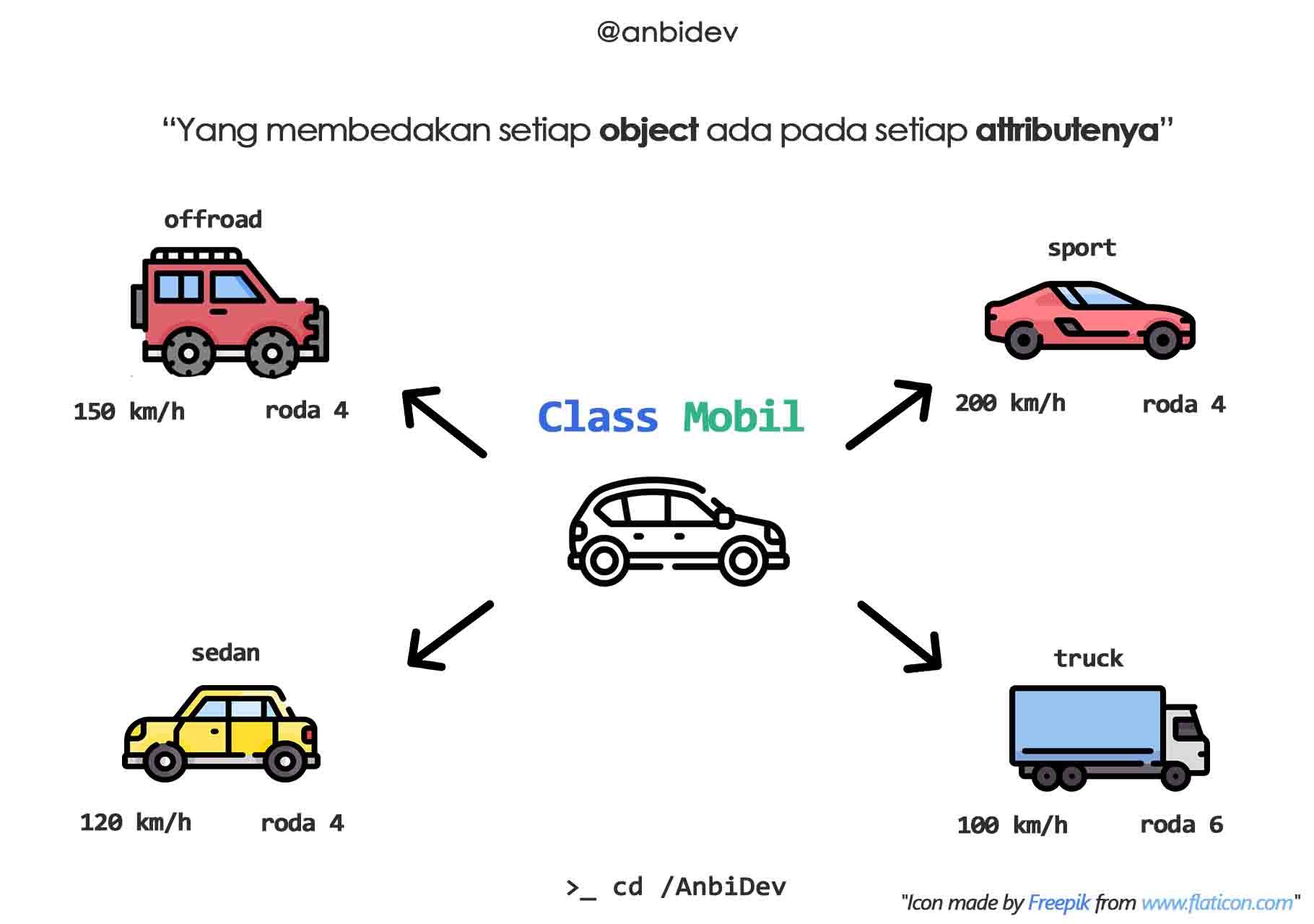
Maka outputnya adalah :



Gambar Output Method Roda

## Apa itu Object?

Object adalah instance atau representasi dari sebuah class. Jika class adalah sebuah cetakan, maka object adalah hasil dari cetakan tersebut. Kita dapat membuat berbagai object dengan class mobil yang tadi kita buat.

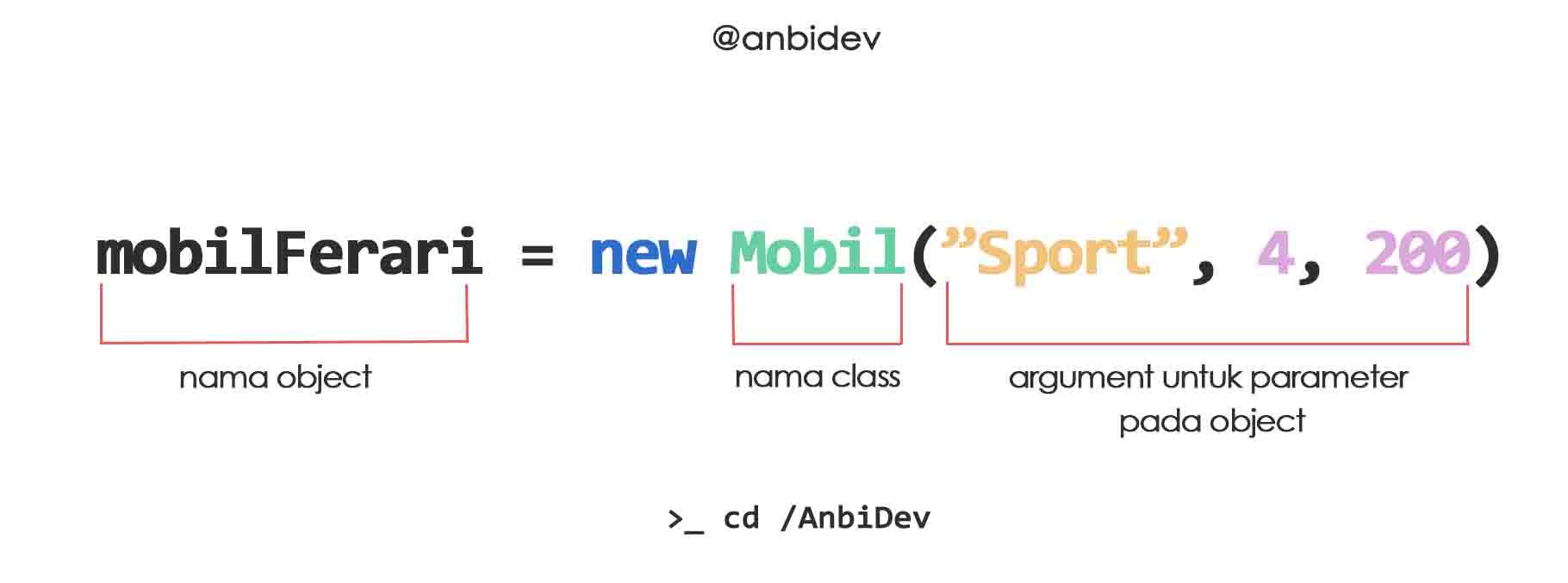


Gambar Objek Class Mobil

Cara mendeklarasikan object dari sebuah class pada java adalah dengan memanggil nama class beserta dengan parameter yang diberikan pada fungsi *constructor.*

**Cara memanggil Objek kelas ?**

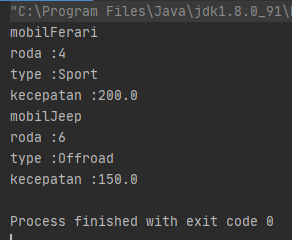
| Mobil mobilFerari = new Mobil("Sport", 4, 200.0); Mobil mobilJeep = new Mobil("Offroad", 6, 150.0); |
| --- |



Gambar Struktur Deklarasi Objek

Pada kode diatas membuat **2 object** pertama adalah object **mobilFerari** dan **mobilOffroad**. Kedua object tersebut dibuat dari class yang sama (Mobil) dan memiliki perilaku (*behaviour*) yang sama tapi dengan karakteristik (*attribute*) yang berbeda.

Jika di run, maka outputnya adalah :



Gambar Output Objek Class Mobil

## Kode Program Lengkap

Class Mobil

| package com.example.demo.model; //Class public class Mobil {   //Variabel  public String type;  public int roda;  public Double kecepatan;   //Construktor  public Mobil(){};    public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }   //Method VOID  public void doMelaju(String kecepatan){  System.*out*.println("Melaju dengan kecepatan : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  }   //Return Method  public double hitungKecepatan(Double kecepatan, int waktuDetik){  return kecepatan/waktuDetik;  }   //Setter and Getter  public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public int getRoda() {  return roda;  }   public void setRoda(int roda) {  this.roda = roda;  }   public Double getKecepatan() {  return kecepatan;  }   public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan;  } } |
| --- |

Class MainMobil

| public class MainMobil {  public static Mobil *objMobil* = new Mobil();  public static void main(String[] args){  Mobil mobilFerari = new Mobil("Sport", 4, 200.0);  Mobil mobilJeep = new Mobil("Offroad", 6, 150.0);  System.*out*.println("mobilFerari");  System.*out*.println("roda :"+mobilFerari.getRoda());  System.*out*.println("type :"+mobilFerari.getType());  System.*out*.println("kecepatan :"+mobilFerari.getKecepatan());  System.*out*.println("mobilJeep");  System.*out*.println("roda :"+mobilJeep.getRoda());  System.*out*.println("type :"+mobilJeep.getType());  System.*out*.println("kecepatan :"+mobilJeep.getKecepatan());   }  } |
| --- |

# Cara Kerja OOP (*Object-Oriented Programming*)

**Langkah pertama**

dalam OOP yaitu mengidentifikasi semua objek yang ingin dimanipulasi oleh *programmer* dan bagaimana mereka saling berhubungan.

Praktiknya sering dikenal sebagai [*data modelling* atau pemodelan data](https://rifqimulyawan.com/blog/pengertian-data-modelling/).

**Langkah kedua**

Setelah suatu objek diketahui, itu kemudian digeneralisasikan sebagai *object class* (kelas objek) yang mendefinisikan jenis data yang dikandungnya dan urutan logika apa pun yang dapat memanipulasinya.

**Langkah ketiga**

Setiap urutan logika yang berbeda dikenal sebagai *method* dan objek dapat berkomunikasi dengan antarmuka yang didefinisikan dengan baik yang disebut *message*.

Sederhananya, OOP berfokus pada objek yang ingin dimanipulasi *developers* atau pengembang daripada logika yang diperlukan untuk memanipulasi mereka.

Pendekatan pemrograman ini sangat cocok untuk program yang besar, kompleks, dan diperbarui atau dipelihara secara aktif.

Karena pengorganisasian program berorientasi objek, metode ini juga kondusif untuk pengembangan kolaboratif di mana proyek dapat dibagi menjadi kelompok-kelompok.

# Manfaat OOP

Pemrograman berorientasi objek, seperti namanya, *Object-Oriented Programming* atau OOP mengacu pada bahasa yang menggunakan objek dalam pemrograman.

Pemrograman berorientasi objek bertujuan untuk mengimplementasikan entitas dunia nyata seperti pewarisan, persembunyian, polimorfisme dan lainnya dalam pemrograman.

Tujuan utama OOP adalah untuk menyatukan data dan fungsi yang beroperasi pada mereka sehingga tidak ada bagian lain dari kode yang dapat mengakses data ini kecuali fungsi itu.

Salah satu manfaat besar dan utama OOP (*Object-Oriented Programming*) adalah skalabilitas, dengan objek dan definisi tidak memiliki batasan yang terbatas.

Selain itu, pemisahan data dari metode mencegah masalah umum yang ditemukan dalam bahasa perangkat lunak linier yang lebih lama.

Jika bug ([baca pengertian bug disini](https://rifqimulyawan.com/blog/pengertian-bug/)) muncul dalam kode linier, bug itu dapat diterjemahkan melalui sistem dan membuat banyak kesalahan yang sulit dilacak.

Sebaliknya, program OOP, dengan pemisahan metode dan data, tidak rentan terhadap kesalahan yang dapat berkembang biak.

Selain itu, adapun beberapa keuntungan dan manfaat dari *Object-Oriented Programming* (OOP) yang harus kalian ketahui adalah sebagai berikut:

* OOP memodelkan hal-hal kompleks sebagai struktur sederhana yang dapat direproduksi.
* Kode OOP dapat digunakan kembali.
* OOP mencegah *duplication* atau duplikasi kode.
* OOP membuat memperbaiki bug lebih mudah.
* Memperbaiki kesalahan yang terkandung dalam kelas yang terstruktur dengan baik lebih mudah daripada menemukan kesalahan di banyak tempat dalam kode.
* OOP melindungi informasi melalui *encapsulation* (enkapsulasi). Data objek hanya dapat diakses melalui properti dan metode publik.
* OOP menggunakan abstraksi untuk membuat kompleksitas dapat dicerna.
* OOP mudah diajak bekerja sama dalam tim pengembang, *class* dapat diberikan kepada pengembang yang berbeda, yang memungkinkan banyak orang untuk membuat kode proyek secara bersamaan.

# Daftar Pustaka

https://www.anbidev.com/python-oop-intro/

<https://idcloudhost.com/panduan/mengenal-pengertian-dan-konsep-oop/>

<https://www.anbidev.com/python-oop-intro/>