**Rangkuman Object-Oriented Programming (OOP)**

Kita sudah berada di penghujung materi Object-Oriented Programming (OOP). Sampai sejauh ini, Anda diharapkan paham untuk mengimplementasikan konsep OOP ke dalam setiap program yang Anda bangun. Mari kita rangkum secara saksama.

**Duck Typing**

Duck typing merupakan konsep yang menjelaskan bahwa sebuah tipe atau class dari sebuah object tidak lebih penting daripada methodyang menjadi perilakunya. Konsep ini berbunyi "**if it walks like a duck and it quacks like a duck, then it must be a duck**" yang artinya jika sesuatu berjalan seperti bebek dan bersuara seperti bebek, kemungkinan besar ia adalah bebek.

Python ingin memberikan keleluasaan terhadap para developernya untuk tidak perlu mencemaskan tipe atau kelas (class) dari sebuah objek (object), yang lebih penting adalah kemampuan melakukan operasinya (method).

**Class, Object, dan Method**

Object-oriented programming adalah paradigma pemrograman berorientasi pada pengorganisasian kode menjadi objek-objek yang memiliki atribut dan perilaku (method). Objek adalah turunan dari class dan kelas merupakan cetakan yang memungkinkan kita dapat membuat banyak objek berdasarkan cetakan tersebut.

Method adalah perilaku atau tindakan yang dapat dilakukan oleh objek atau kelas. Sebagaimana halnya maju, mundur, berbelok, dan berhenti pada contoh sebelumnya. Atribut adalah variabel yang menjadi identitas dari objek atau kelas, seperti warna dan kecepatan pada contoh sebelumnya.

Mari sederhanakan dengan tabel berikut.

| **Nama** | **Deskripsi** | **Contoh** |
| --- | --- | --- |
| Class (Kelas) | Cetakan (blueprint) untuk membuat objek-objek dengan karakteristik dan perilaku yang serupa. | Mobil; Manusia. |
| Object (Objek) | Turunan atau perwujudan dari kelas. | Mobil Dicoding; Budi, Herman, Asep. |
| Perilaku (Method) | Perilaku atau tindakan yang dapat dilakukan oleh objek atau kelas. | Maju, mundur, berbelok, berhenti. |
| Atribut | Variabel yang menjadi identitas dari objek atau kelas. | Warna, kecepatan, merek. |

**Class**

Pembuatan class dalam Python mirip seperti fungsi, yakni perlu menggunakan keyword untuk bisa membuatnya. Keyword atau kata kunci untuk membuat kelas dalam Python adalah "class".

1. class Mobil:
2. pass

**Object (Objek)**

Untuk memanggil kelas yang telah dibuat, kita membuat sebuah objek. Berdasarkan KBBI dari kemendikbud, [objek](https://kbbi.kemdikbud.go.id/entri/objek) merupakan benda, hal, dan sebagainya yang dijadikan sasaran untuk diteliti, diperhatikan, dan sebagainya. Keterkaitan antara objek dan class sangat erat. Contohnya, jika Anda membuat kelas bernama manusia, objeknya adalah manusia dengan nama yang berbeda.

Anda bisa umpamakan kelas adalah bentuk abstrak dari objek layaknya cetakan atau *blueprint*. Saat kelas diwujudkan menjadi bentuk yang lebih nyata, proses ini disebut sebagai instansiasi. Itulah sebabnya objek disebut juga sebagai instance atau **instance of the class**.

1. class Mobil:
2. # Atribut
3. warna = "Merah"
5. mobil\_1 = Mobil()

**Atribut**

Dalam Python, ada dua jenis atribut kelas yang dapat dibagi, yaitu **atribut kelas**dan **atribut objek atau instance**. Atribut kelas adalah jenis atribut yang secara otomatis terdefinisi dan menjadi bawaan kelas ketika instance dibuat berdasarkan kelas tersebut. Anda dapat menganggapnya sebagai nilai default atau bawaan dari kelas. Jika Anda membuat beberapa objek berdasarkan kelas yang memiliki jenis atribut ini, setiap objek akan memiliki atribut yang sama dengan nilai yang sama.

Namun, perlu diperhatikan bahwa jenis atribut kelas memiliki kelemahan, yaitu ketika nilai atribut kelas diubah, perubahan tersebut akan memengaruhi semua objek yang dibuat berdasarkan kelas tersebut.

1. **Atribut Kelas**  
   Atribut kelas adalah jenis atribut yang melekat pada kelas sehingga menjadi bawaan ketika membuat sebuah instance.
   1. class Mobil:
   2. # Atribut kelas
   3. warna = "Merah"
   5. mobil1 = Mobil()
   6. print(mobil1.warna)
2. **Atribut Instance**  
   Atribut instance atau atribut objek adalah atribut yang terkait dengan instance atau objek itu sendiri, bukan kelas.
   1. class Mobil:
   2. # Atribut Instance
   3. def \_\_init\_\_(self):
   4. self.warna = 'Merah'
   6. mobil\_1 = Mobil()
   7. print(mobil\_1.warna)

Untuk membuat atribut instance, kita perlu membuat atribut tersebut melalui class constructor.

**Class Constructor**

Pembangun kelas atau class constructor adalah sebuah **fungsi khusus**dalam Python yang digunakan untuk menentukan nilai atau kondisi awal dari suatu kelas. Dengan fungsi ini, saat kita melakukan proses instansiasi atau pembuatan objek baru, hal pertama yang dilakukan adalah memanggilnya terlebih dahulu.

Tidak hanya untuk membuat atribut, kita dapat menambahkan parameter lain dalam class constructor.

1. class Mobil:
2. def \_\_init\_\_(self, warna, merek, kecepatan):
3. self.warna = warna
4. self.merek = merek
5. self.kecepatan = kecepatan
7. mobil\_1 = Mobil('Merah', 'DicodingCar', 160)
9. print(mobil\_1.warna)
10. print(mobil\_1.merek)
11. print(mobil\_1.kecepatan)
13. """
14. Output:
15. Merah
16. DicodingCar
17. 160
18. """

**Method**

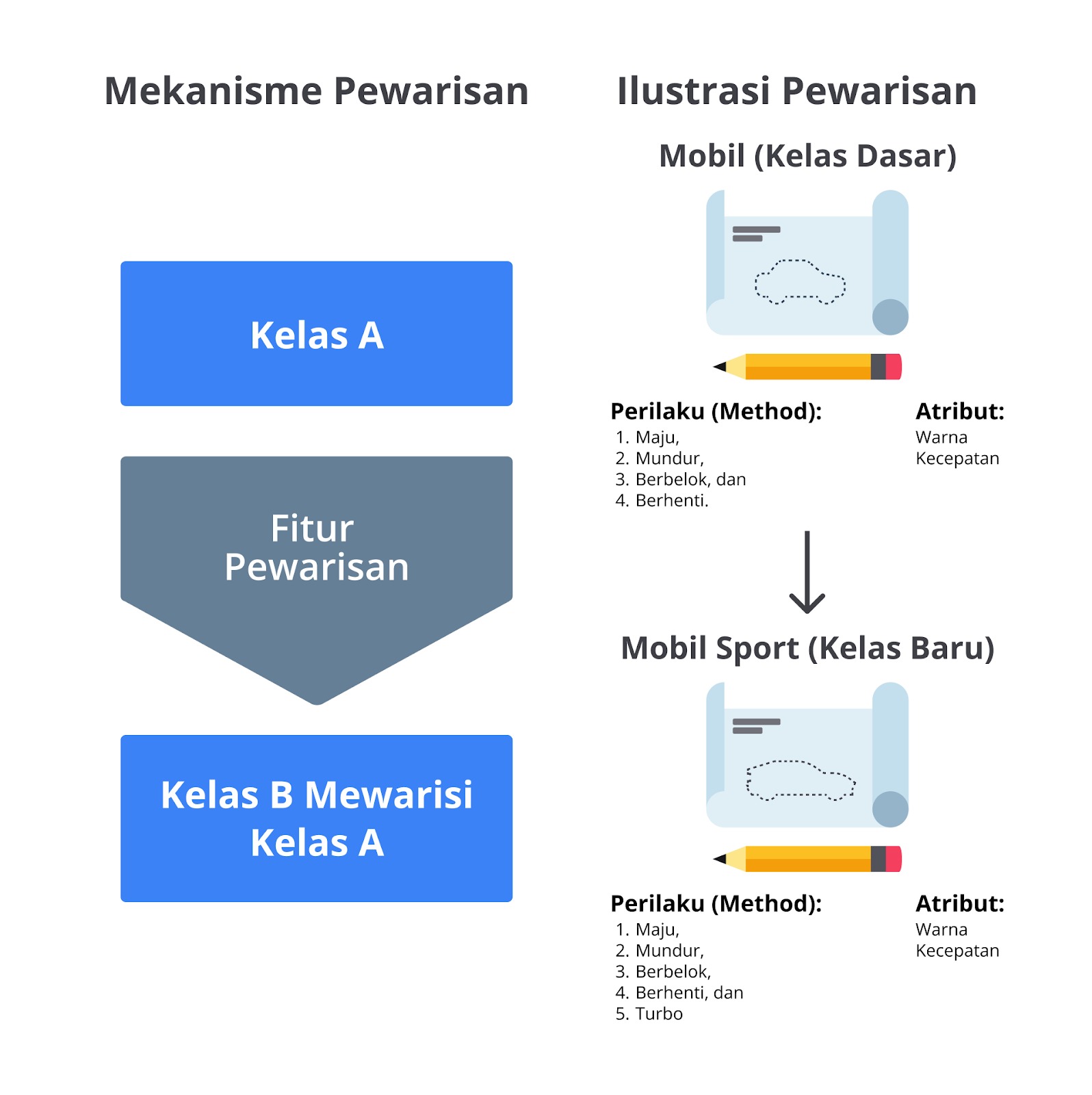
Method merupakan perilaku atau tindakan yang dapat dilakukan oleh objek atau kelas. Dalam pembuatan metode, sebenarnya kita membuat fungsi di dalam kelas itu sendiri. Dengan kata lain, kita menggunakan kata kunci **"def"** atau membuat fungsi sebagai suatu metode. Method sendiri dibagi menjadi tiga jenis.

1. **Metode dari Object (Object Method)**  
   Jenis pertama adalah method yang melekat terhadap objek. Ciri dari jenis metode ini adalah **adanya parameter *self***yang merujuk pada objek saat ini yang dibuat.
   1. class Mobil:
   2. def \_\_init\_\_(self, warna, merek, kecepatan):
   3. self.warna = warna
   4. self.merek = merek
   5. self.kecepatan = kecepatan
   7. def tambah\_kecepatan(self):
   8. self.kecepatan += 10
2. **Metode secara Statis(Static Method)**  
   Static method adalah fungsi atau method pada sebuah kelas yang bersifat statis. Artinya metode atau fungsi ini bersifat independen dan tidak terikat pada instance kelas. Metode ini dapat dianggap seperti kita membuat fungsi seperti biasa, tetapi didefinisikan dalam kelas. Jadi, ini menjadi perilaku untuk kelas tersebut. Untuk membuat static method, Anda bisa menambahkan dekorator **@staticmethod**tepat sebelum mendefinisikan fungsi atau metode.
   1. class Mobil:
   2. def \_\_init\_\_(self, merek):
   3. self.merek = merek
   5. @staticmethod
   6. def intro\_mobil():
   7. print("Ini adalah metode dari kelas Mobil")
   9. Mobil.intro\_mobil()
   10. mobil\_1 = Mobil("DicodingCar")
   11. mobil\_1.intro\_mobil()
3. **Metode dari Class(Class Method)**  
   Metode terakhir adalah class method yang termasuk jenis metode cukup spesial dalam Python. Jika object method identik dengan parameter *self* yang merujuk pada objek, **class method**juga memerlukan sebuah parameter yang merujuk pada kelas.
   1. class Mobil:
   2. def \_\_init\_\_(self, merek):
   3. self.merek = merek
   5. @classmethod
   6. def intro\_mobil(cls):
   7. print("Ini adalah metode dari kelas Mobil")
   9. Mobil.intro\_mobil()
   10. mobil\_1 = Mobil("DicodingCar")
   11. mobil\_1.intro\_mobil()

**Inheritance**

Berikut adalah rangkuman materi terkait inheritance atau dalam bahasa Indonesia disebut pewarisan.

**Mekanisme Pewarisan**

[](https://www.dicoding.com/academies/86/tutorials/33013)

Untuk melakukan pewarisan, anggap kita memiliki "kelas A" sebagai induk atau kelas dasar. Dari kelas A tersebut, kita membuat kelas baru bernama "kelas B" sebagai kelas turunan dari kelas yang didapatkan (kelas A). Ketika kelas B mewarisi kelas A, secara otomatis kelas ini memiliki fitur-fitur yang dimiliki oleh kelas A tersebut, dalam hal ini atribut-atribut dan metode-metode.

Jika kelas B memiliki nama metode yang sama dengan kelas A, metode tersebut akan menimpa metode yang diwariskan oleh kelas A.

1. class Mobil:
2. def \_\_init\_\_(self, warna, merek, kecepatan):
3. self.warna = warna
4. self.merek = merek
5. self.kecepatan = kecepatan
7. def tambah\_kecepatan(self):
8. self.kecepatan += 10

11. class MobilSport(Mobil):
12. def turbo(self):
13. self.kecepatan += 50

**Override**

Ketika kita membuat metode baru di kelas turunan (kelas baru) dengan nama yang sama seperti metode di kelas induk, itu akan menyebabkan metode baru menimpa (override) metode pada kelas induk.

1. class Mobil:
2. def \_\_init\_\_(self, warna, merek, kecepatan):
3. self.warna = warna
4. self.merek = merek
5. self.kecepatan = kecepatan
7. def tambah\_kecepatan(self):     # tambah\_kecepatan
8. self.kecepatan += 10
10. class MobilSport(Mobil):
11. def turbo(self):
12. self.kecepatan += 50
14. def tambah\_kecepatan(self):     # tambah\_kecepatan
15. self.kecepatan += 20

**Super**

Terakhir adalah super, yakni fungsi yang digunakan untuk menggunakan metode atau atribut dari kelas induk, tetapi tidak ingin menuliskan ulang semua kode. Nama super sebenarnya merujuk pada kelas induk yang disebut juga sebagai **super class**. Kita bisa memanfaatkan konsep ini untuk menghindari kode berulang dan memanfaatkan fungsi yang sudah ada pada kelas induk (super class).

1. class Mobil:
2. def \_\_init\_\_(self, warna, merek, kecepatan):
3. self.warna = warna
4. self.merek = merek
5. self.kecepatan = kecepatan
7. def tambah\_kecepatan(self):
8. self.kecepatan += 10

11. class MobilSport(Mobil):
12. def turbo(self):
13. self.kecepatan += 50
15. def tambah\_kecepatan(self):
16. super().tambah\_kecepatan()     # Super