Pertemuan 11

Object-Oriented Programming (OOP)

Object Oriented Programming

OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek. **Tujuan dari OOP diciptakan** adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Jadi setiap bagian dari suatu permasalahan adalah objek, nah objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi. Sebagai salah satu contoh Pesawat, Pesawat adalah sebuah objek. Pesawat itu sendiri terbentuk dari beberapa objek yang lebih kecil lagi seperti mesin, roda, baling-baling, kursi, dll. Pesawat sebagai objek yang terbentuk dari objek-objek yang lebih kecil saling berhubungan, berinteraksi, berkomunikasi dan saling mengirim pesan kepada objek-objek yang lainnya. Begitu juga dengan program, sebuah objek yang besar dibentuk dari beberapa objek yang lebih kecil, objek-objek itu saling berkomunikasi, dan saling berkirim pesan kepada objek yang lain.

Istilah	Penjelasan
Class	Prototipe yang ditentukan pengguna untuk objek yang mendefinisikan seperangkat atribut yang menjadi ciri objek kelas apa pun. Atribut adalah data anggota (variabel kelas dan variabel contoh) dan metode, diakses melalui notasi titik.
Object	Contoh unik dari struktur data yang didefinisikan oleh kelasnya. Objek terdiri dari kedua anggota data (variabel kelas dan variabel contoh) dan metode.
Method	Jenis fungsi khusus yang didefinisikan dalam definisi kelas.
Instance	Objek individu dari kelas tertentu. Obyek obj yang termasuk dalam Lingkaran kelas, misalnya, adalah turunan dari Lingkaran kelas.
Instantiation	Penciptaan sebuah instance dari sebuah kelas.
Class variable	Sebuah variabel yang dibagi oleh semua contoh kelas. Variabel kelas didefinisikan dalam kelas tapi di luar metode kelas manapun. Variabel kelas tidak digunakan sesering variabel contoh.
Data member	Variabel kelas atau variabel contoh yang menyimpan data yang terkait dengan kelas dan objeknya.
Function overloading	Penugasan lebih dari satu perilaku ke fungsi tertentu. Operasi yang dilakukan bervariasi menurut jenis objek atau argumen yang terlibat.
Operator overloading	Penugasan lebih dari satu fungsi ke operator tertentu.
Inheritance	Pengalihan karakteristik kelas ke kelas lain yang berasal darinya.
Instantiation	Penciptaan sebuah instance dari sebuah kelas.

Konsep OOP

- **Kelas** merupakan deskripsi abstrak informasi dan tingkah laku dari sekumpulan data.
- **Kelas** dapat diilustrasikan sebagai suatu cetak biru(blueprint) atau prototipe yang digunakan untuk menciptakan objek.
- **Kelas** merupakan tipe data bagi objek yang mengenkapsulasi data dan operasi pada data dalam suatu unit tunggal.
- **Kelas** mendefinisikan suatu struktur yang terdiri atas data kelas (data field), prosedur atau fungsi (method), dan sifat kelas (property).

Class

Pada konsep pemrograman berbasis object, anda tidak akan asing lagi mendengar istilah class, object, attribute, behaviour, inheritance, dll. Semua itu pasti akan anda temui disemua bahasa pemrograman yang support OOP. Jika dianalogikan, class merupakan suatu tubuh dari OOP. Class merupakan abstraksi atau *blueprint* yang mendefinisikan suatu object tertentu. Class akan menampung semua attribute dan perilaku dari object itu. Berikut contoh implementasi class pada Python:

```
class Car:
color = 'black'
transmission = 'manual'
def __init__(self, transmission):    self.transmission = transmission
print('Engine is ready!') def drive(self):    print('Drive') def
reverse(self):   print('Reverse. Please check your behind.')
```

```
class Car:
color = 'black'
transmission = 'manual'
def __init__(self, transmission):
      self.transmission = transmission
       print('Engine is ready!')

def drive(self):
      print('Drive')

def reverse(self):
      print('Reverse.  Please  check  your
behind.')
```

Jika diperhatikan, dalam class **Car** terdapat 2 attribute yaitu **color = 'black', transmission = 'manual'** dan method yaitu **drive(), reverse().** Method dalam konsep OOP mewakili suatu 'behaviour' dari class atau object itu sendiri. Kita akan bahas lebih detail mengenai method.

Function/Method

Fungsi method dalam konsep OOP adalah untuk merepresentasikan suatu behaviour. Dalam contoh di atas suatu object 'mobil' memiliki behaviour antara lain adalah bergerak dan mundur. Suatu method bisa juga memiliki satu atau beberapa parameter, sebagai contoh:

```
gear_position = 'N'
def change_gear(self, gear):
      self.gear_position = gear
      print('Gear positiion on: ' + self.gear_position)
```

Pada method **change_gear()** terdapat 1 parameter yaitu **gear**. Ketika method tersebut dipanggil dan anda tidak memberikan value pada parameter tersebut, maka program akan melempar error. Bagaimanapun juga parameter yang sudah didefinisikan pada suatu method harus memiliki value meskipun value tersebut None. Cara lainnya adalah dengan mendefinisikan default value pada parameter tersebut sejak awal method tersebut dibuat:

```
gear_position = 'N'

def change_gear(self, gear='N'):
      self.gear_position = gear
      print('Gear positiion on: ' + self.gear_position)

self.change_gear()
>>> 'Gear position on: N`
self.change_gear('R')
>>> 'Gear position on: R
```

Jika diperhatikan, terdapat keyword **self** pada salah satu parameter method di atas. Keyword self mengacu pada Class Instance untuk mengakses attribute atau method dari class itu sendiri. Dalam bahasa pemrograman Java, terdapat keyword **this** yang memiliki fungsi yang mirip dengan keyword **self** pada Python. Pemberian keyword **self** pada parameter awal suatu method menjadi wajib jika anda mendefinisikan method tersebut di dalam block suatu class.

Suatu method juga bisa mengembalikan suatu value ketika method tersebut dipanggil. Berikut contoh implementasinya:

```
def get_gear_position(self):
        return self.gear_position

gear_position = self.get_gear_position()
```

Constructor

Pada contoh awal tentang penjelasan class, terdapat sebuah method bernama **__init__().** Method itulah yang disebut dengan constructor. Suatu constructor berbeda dengan method lainnya, karena constructor akan otomatis dieksekusi ketika membuat object dari class itu sendiri.

```
class Car:
        color = 'black'
        transmission = 'manual'

def __init__(self, transmission):
        self.transmission = transmission
        print('Engine is ready!')
...

honda = Car('automatic')
>>> 'Engine is ready!'
```

Ketika object honda dibuat dari class **Car**, constructor langsung dieksekusi. Hal ini berguna jika anda membutuhkan proses inisialisasi ketika suatu object dibuat. Suatu constructor juga bisa memiliki satu atau beberapa parameter, sama seperti method pada umumnya namun constructor tidak bisa mengembalikan value.

Object

Object merupakan produk hasil dari suatu class. Jika class merupakan blueprint dari suatu rancangan bangunan, maka object adalah bangunan itu sendiri. Begitulah contoh analogi yang bisa saya gambarkan mengenai relasi antara class dan object. Berikut contoh implementasi dalam bentuk code program:

```
class Car:
        color = 'black'
        transmission = 'manual'
        gear_position = 'N'
def __init__(self, transmission):
        self.transmission = transmission
        print('Engine is ready!')
def drive(self):
        self.gear_position = 'D'
        print('Drive')
def reverse(self):
        self.gear_position = 'N'
        print('Reverse. Please check your behind.')
def change_gear(self, gear='N'):
        self.gear_position = gear
        print('Gear position on: ' + self.gear_position)
def get_gear_position(self):
        return
        self.gear_position
car1 = Car('manual')
car1.change_gear('D-1')

car2 = Car('automatic')
gear_position = car2.get_gear_position()
print(gear_position)
>>> 'N'
```

Dari contoh di atas, terdapat 2 buah object **car1** dan **car2** yang dibuat dari class yang sama. Masing-masing dari object tersebut berdiri sendiri, artinya jika terjadi perubahan attribute dari object **car1** tidak akan mempengaruhi object **car2** meskipun dari class yang sama.

Inheritance

Salah satu keuntungan dari konsep OOP ialah *reusable codes* yang bisa mengoptimalkan penggunaan code program agar lebih efisien dan meminimalisir redudansi.

• **Kita dapat mendefinisikan** suatu kelas baru dengan mewarisi sifat dari kelas lain yang sudah ada.

•**Penurunan sifat** ini bisa dilakukan secara bertingkat tingkat, sehingga semakin ke bawah kelas tersebut menjadi semakin spesifik.

• **Sub kelas** memungkinkan kita untuk melakukan spesifikasi detail dan perilaku khusus dari kelas supernya.

• **Dengan konsep pewarisan**, seorang programmer dapat menggunakan kode yang telah ditulisnya pada kelas super berulang kali pada kelas-kelas turunannya tanpa harus menulis ulang semua kodekode itu.

Semua itu berkat adanya fitur inheritance yang memungkinkan suatu class (parent) menurunkan semua attribute dan behaviour nya ke class (child) lain. Berikut contoh penerapannya:

```
class Tesla(Car):
    pass    # use 'pass' keyword to define
class only
tesla = Tesla()
tesla.drive()
>>> 'Drive'
```

Pada potongan code di atas, class **Tesla** merupakan turunan dari class **Car**. Jika diperhatikan pada class Tesla tidak didefinisikan method **drive()** namun class tersebut bisa memanggil method **drive()**. Method tersebut berasal dari class parentnya yaitu class **Car**, sehingga tidak perlu lagi didefinisikan ulang pada class childnya. Dengan cara seperti ini anda bisa melakukan reusable codes sehingga source code menjadi lebih *clean*.

Overriding

Ada suatu kondisi dimana suatu method yang berasal dari parent ingin anda modifikasi atau ditambahkan beberapa fitur sesuai kebutuhan pada class child, disinilah peran dari 'overriding method'. Dengan menggunakan fungsi super(), anda bisa memanggil instance dari class parent di dalam suatu method untuk memanggil fungsi dari parent tersebut. Perhatikan contoh di bawah ini:

```
class Tesla(Car):
    def drive(self):
        super().drive()
        print('LOL Gas')
```

Private Attribute/Function

Tidak semua attribute maupun method bisa diturunkan pada class child. Anda bisa menentukan mana attribute atau method yang ingin diproteksi agar tidak bisa digunakan pada class turunannya. Berikut caranya:

```
__factory_number = '0123456789'

def __get_factory_number(self):
     return self.__factory_number
```

Polymorphism

Polimorfisme yang memungkinkan anda untuk membuat banyak bentuk dari satu object. Berikut contoh implementasinya:

```
class Car:
      def fuel(self):
            return 'gas'

class Honda(Car):
      pass

class Tesla(Car):
      def fuel(self):
            return 'electricity'
def get_fuel(car):
      print(car.fuel())

get_fuel(Tesla())
get_fuel(Honda())
>>> 'electricity'>>> 'gas
```

• **Polimorfisme** merupakan kemampuan objek objek yang berbeda kelas namun terkait dalam pewarisan untuk merespon secara berbeda terhadap suatu pesan yang sama.

• **Polimorfisme** juga dapat dikatakan kemampuan sebuah objek untuk memutuskan method mana yang akan diterapkan padanya, tergantung letak objek tersebut pada jenjang pewarisan.