COMPUTER
PROGRAMING 2

ENKAPULASI

MODUL 12

2023

DAFTAR ISI

OVERRIDING

A. Pengertian Enkapulasi

Access Modifiers (Enkapulasi) adalah sebuah konsep di dalam pemrograman berorientasi objek di mana kita bisa mengatur hak akses suatu atribut dan fungsi pada sebuah class [1]. Konsep ini juga biasa disebut sebagai enkapsulasi, di mana kita akan mendefinisikan mana atribut atau fungsi yang boleh diakses secara terbuka, mana yang bisa diakses secara terbatas, atau mana yang hanya bisa diakses oleh internal kelas alias privat.

Tujuan :

- Membatasi hak akses pada anggota/member dari sebuah class
- Memastikan bahwa nilai yang masuk ke dalam class adalah valid

Metode yang sering digunakan pada metode enkapsulasi

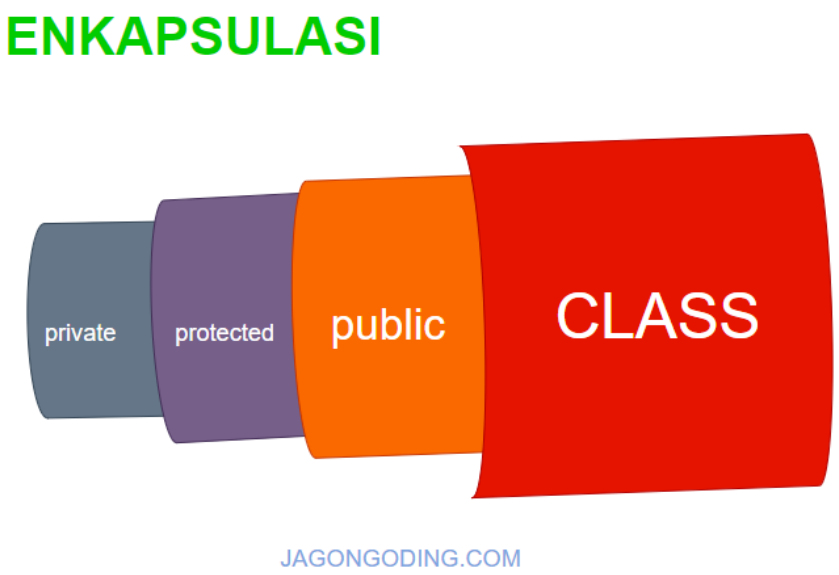
- Getter / Accessor : yaitu sebuah metode yang digunakan untuk mengakses field dari sebuah class.
- Setter / Mutator : yaitu sebuah metode yang digunakan untuk mengubah field dari sebuah class

B. Jenis Access Modifiers Pada Python

Sama seperti bahasa pemrograman berbasis objek lainnya semisal java, c++, php, dan lain-lain, python juga mendukung fitur access modifiers.

Access modifiers yang tersedia di dalam python ada 3:

- Public
- Protected
- dan Private



C. Public Access Modifier

Variabel atau atribut yang memiliki hak akses publik bisa diakses dari mana saja baik dari luar kelas mau pun dari dalam kelas. Perhatikan contoh berikut:

```
class Segitiga:

  def __init__(self, alas, tinggi):
    self.alas = alas
    self.tinggi = tinggi
    self.luas = 0.5 * alas * tinggi
```

Pada contoh di atas, kita membuat sebuah kelas dengan nama Segitiga. Kelas tersebut menerima dua buah parameter: yaitu alas dan tinggi.

Kemudian ia juga memiliki 3 buah atribut (alas, tinggi, dan luas) yang mana semuanya memiliki hak akses publik. Untuk membuktikannya, mari kita coba akses ke-3 atribut di atas dari luar kelas:

```
segitiga_besar = Segitiga(100, 80)

# akses variabel alas, tinggi, dan luas dari luar kelas
print(f'alas: {segitiga_besar.alas}')
print(f'tinggi: {segitiga_besar.tinggi}')
print(f'luas: {segitiga_besar.luas}')
```

Jika dijalankan, kita akan mendapatkan output seperti berikut:

```
alas: 100
tinggi: 80
luas: 4000.0
```

D. Protected Access Modifier

Variabel atau atribut yang memiliki hak akses protected hanya bisa diakses secara terbatas oleh dirinya sendiri (yaitu di dalam internal kelas), dan juga dari kelas turunannya.

Mari kita mencoba mempraktikkannya. Untuk mendefinisikan atribut dengan hak akses protected, kita harus menggunakan prefix **underscore**(_) sebelum nama variabel.

```
class Mobil:
  def __init__(self, merk):
    self._merk = merk
```

Pada kode program di atas, kita membuat sebuah kelas bernama Mobil. Dan di dalam kelas tersebut, kita mendefinisikan satu buah atribut bernama _merk, yang mana hak akses dari atribut tersebut adalah protected karena nama variabelnya diawali oleh underscore (_).

Mari kita coba akses dari luar kelas:

```
sedan = Mobil('Toyota')

# tampilkan _merk dari luar kelas
print(f'Merk: {sedan._merk}')
```

Ya, betul sekali. Kita masih bisa mengakses atribut _merk dari luar kelas, karena hal ini hanya bersifat convention alias adat atau kebiasaan saja yang harus disepakati oleh programmer. Di mana jika suatu atribut diawali oleh _, maka ia harusnya tidak boleh diakses kecuali dari internal kelas tersebut atau dari kelas yang mewarisinya.

Yang harusnya kita lakukan adalah: mengakses atribut protected hanya dari kelas internal atau dari kelas turunan, perhatikan contoh berikut:

```
class Mobil:
  def __init__(self, merk):
    self._merk = merk

class MobilBalap(Mobil):
  def __init__(self, merk, total_gear):
    super().__init__(merk)
    self._total_gear = total_gear

  def pamer(self):
    # akses _merk dari subclass
    print(
      f'Ini mobil {self._merk} dengan total gear
{self._total_gear}'
    )
Bikin objek dari kelas MobilBalap:

ferrari = MobilBalap('Ferrari', 8)
ferrari.pamer()
```

Output:

```
Ini mobil Ferrari dengan total gear 8
```

E. Private Access Modifier

Modifier selanjutnya adalah private. Setiap variabel di dalam suatu kelas yang memiliki hak akses private maka ia hanya bisa diakses di dalam kelas tersebut. Tidak bisa diakses dari luar bahkan dari kelas yang mewarisinya.

Untuk membuat sebuah atribut menjadi private, kita harus menambahkan dua buah underscore sebagai prefix nama atribut. Perhatikan contoh berikut:

```
class Mobil:
  def __init__(self, merk):
    self.__merk = merk
```

Pad kode di atas, kita telah membuat sebuah atribut dengan nama __merk. Dan karena nama tersebut diawali dua buah underscore, maka ia tidak bisa diakses kecuali dari dalam kelas Mobil saja.

Mari kita buktikan:

```
jip = Mobil('Jeep')
print(f'Merk: {jip.__merk}')
```

Kita akan mendapatkan sebuah error sebagai berikut:

```
AttributeError: 'Mobil' object has no attribute '__merk'
```

Tapi jika kita ubah kodenya menjadi seperti ini:

```
class Mobil:
  def __init__(self, merk):
    self.__merk = merk

  def tampilkan_merk(self):
    print(f'Merk: {self.__merk}')

jip = Mobil('Jeep')
jip.tampilkan_merk()
```

Kita akan mendapatkan output:

```
Merk: Jeep
```

Kenapa? Karena kita mengakses variabel __merk dari dalam internal kelas Mobil, bukan dari luar.

F. Accessor Mutator

Selanjutnya kita bisa membuat accessor dan mutator atau getter dan setter pada suatu kelas di python.

Untuk apa accessor dan mutator? Ia adalah sebuah fungsi yang akan dieksekusi ketika kita mengakses (aksesor) suatu atribut pada suatu kelas, atau fungsi yang dieksekusi ketika hendak mengatur (mutator) suatu atribut pada suatu kelas.

Untuk mendefinisikan accessor (getter), kita perlu mendefinisikan decorator @property sebelum nama fungsi.

Sedangkan untuk mengatur mutator (setter), kita perlu mendefinisikan descriptor @<nama-atribut>.setter.

Perhatikan contoh berikut:

```
class Mobil:
  def __init__(self, tahun):
    self.tahun = tahun

  @property
  def tahun(self):
    return self.__tahun

  @tahun.setter
  def tahun(self, tahun):
    if tahun > 2021:
      self.__tahun = 2021
    elif tahun < 1990:
      self.__tahun = 1990
    else:
      self.__tahun = tahun
```

Ketika kita nanti mengakses atribut tahun (tanpa underscore), maka fungsi tahun() yang pertama akan dieksekusi.

Sedangkan jika kita mengisi / mengubah / memberi nilai pada atribut tahun (tanpa underscore), maka yang dieksekusi adalah fungsi tahun() yang kedua.

Mari kita coba kode program berikut:

```
sedan = Mobil(2200)
print(f'Mobil ini dibuat tahun {sedan.tahun}')
```

Kode program di atas akan menampilkan tahun dari sedan yang sudah melalui if-else sebelumnya. Sehingga jika kita memberikan nilai tahun yang lebih dari tahun 2021, yang tersimpan tetaplah tahun 2021.

Berikut ini outputnya:

```
Mobil ini dibuat tahun 2021
```

Sedangkan jika kita mengakses atribut aslinya yaitu __tahun, kita akan mendapatkan error karena atribut tersebut bersifat private.

```
print(f'Mobil ini dibuat tahun {sedan.__tahun}')
```

Error:

```
AttributeError: 'Mobil' object has no attribute '__tahun'
```

Selanjutnya mari kita coba ubah atribut tahun seperti berikut:

```
sedan = Mobil(2000)
sedan.tahun = 1800
print(f'Mobil ini keluaran {sedan.tahun}')
```

Output:

```
Mobil ini keluaran 1990
```

Lihat, kita mengubah tahun menjadi 1800 akan tetapi yang tersimpan adalah 1990. Itu terjadi karena ketika kita mengubah atribut tahun, sistem masih memanggil terlebih dahulu fungsi setter tahun().

G. Menggabungkan Public, Protected, dan Private

Selain itu, kita juga bisa menggunakan 3 buah modifiers sekaligus dalam satu kelas, perhatikan contoh berikut:

```
class SebuahKelas:
  def __init__(self):
    self.publik = 'Atribut Publik'
    self._protected = 'Atribut Protected'
    self.__privat = 'Atribut Privat'
```

Pada contoh di atas kita memiliki 3 buah atribut:

```
Atribut publik dengan modifier public
Atribut _protected dengan modifier protected
Dan atribut __privat dengan modifier private
```

H. Kesimpulan

Dalam pemrograman python, kita bisa memanfaatkan fitur access modifier untuk mengenkapsulasi sebuah kode program. Alias mengatur mana atribut yang boleh diakses dari luar, dan mana atribut yang hanya konsumsi internal.

Sayangnya, kelemahan access modifier dalam python adalah:

- Tidak ada atribut yang benar-benar public atau protected. Selama sebuah variabel tidak diawali dua buah underscore, maka ia bisa diakses dari mana pun.
- Hal tersebut menjadikan konsep “protected” modifier hanyalah sebuah convention atau kebiasaan saja, di mana kalau kita mendefinisikan sebuah atribut yang diawali satu buah underscore, maka kita seharusnya tidak mengakses atribut tersebut kecuali dari dalam kelas itu sendiri atau dari kelas turunannya.

Pertemuan Selanjutnya
Pada pertemuan selanjutnya insyaallah kita akan membahas tentang overloading pada python, di mana kita akan mendefinisikan apa yang terjadi jika sebuah kelas dijadikan sebagai operan dari suatu operator.

Bagaimana caranya?

Simak terus tutorial ini, ya! Terima kasih banyak.

=== To be Continue ===

REFERENSI

[1] N. Huda, "Jago Ngoding," 24 May 2021. [Online]. Available: https://jagongoding.com/python/menengah/oop/overriding/. [Accessed 28 November 2023].