COMPUTER
PROGRAMING 2

KONSEP PEWARISAN

MODUL 9

2023

DAFTAR ISI

KONSEP PEWARISAN

Di antara konsep paling dasar dalam paradigma OOP adalah konsep pewarisan. Dengan pewarisan, kita bisa membuat satu objek sebagai induk dari objek-objek lainnya. Setiap objek yang menjadi turunan dari objek induk akan memiliki sifat dan perilaku dasar yang sama.

A. Memahami Konsep Pewarisan Objek

Bagaimana konsep pewarisan bekerja?

Konsep pewarisan adalah konsep di mana sebuah kelas atau objek mewariskan sifat dan perilaku kepada kelas lainnya. Kelas yang menjadi "pemberi waris" dinamakan kelas induk atau kelas basis [1]. Sedangkan kelas yang menjadi "ahli waris" dinamakan sebagai kelas turunan.

B. Sifat Pewarisan

Secara umum, kelas turunan akan selalu memiliki sifat dan perilaku yang sama dengan kelas induknya: mulai dari atribut sampai fungsi-fungsinya. Akan tetapi tidak sebaliknya, belum tentu kelas induk memiliki semua atribut dan sifat dari kelas-kelas turunannya.

Contoh Sederhana Penerapan Pewarisan Objek:
Untuk mempermudah pemahaman, mari kita buat contoh kasus sederhana yang akan kita selesaikan dengan konsep pewarisan.

Kita akan membuat 3 buah objek:

- Orang
- Pelajar
- Pekerja

Masing-masing dari 3 objek tersebut memiliki beberapa sifat dan perilaku yang sama, misalkan:

- nama
- asal
- dan kemampuan memperkenalkan diri

Tetapi, sebagian objek tetap memliki ciri khasnya masing-masing, misalkan:

- Objek Pelajar memiliki atribut sekolah.
- Objek Pekerja memiliki atribut tempat kerja.

Cari hubungan "pewarisan" antar ketiganya
Untuk menyelesaikan kasus di atas, kita perlu menarik sebuah premis bahwa ketiganya memiliki hubungan "waris".

Seperti apa hubungan tersebut? Hubungannya adalah:

- Objek Pelajar sebenarnya adalah objek Orang juga, hanya saja objek Pelajar memiliki atribut tambahan yang tidak dimiliki objek Orang.
- Begitu pula objek Pekerja, ia sebenarnya adalah objek Orang juga, hanya saja ia memiliki atribut tambahan yang tidak dimiliki objek Orang.

Definisikan Kelas Basis dan Kelas Turunan
Setelah mengetahui gambaran hubungan “waris” antar ketiga keelas tersebut, kita bisa simpulkan bahwa:

- Objek Pelajar dan objek Pekerja adalah kelas turunan dari objek Orang.

Setelah berhasil menentukan mana objek induk dan mana objek turunan, mari kita tuangkan hal tersebut kedalam kode program.

Membuat Objek Induk (Parent)
Untuk membuat objek induk pada python, caranya sama dengan cara membuat objek biasa. Karena pada dasarnya, semua objek pada python bisa menjadi objek induk dari objek turunan lainnya.

Mari kita buat objek Orang seperti skenario yang telah kita bahas pada bagian sebelumnya:

```
class Orang:

  def __init__ (self, nama, asal):
    self.nama = nama
    self.asal = asal

  def perkenalan (self):
    print(f'Perkenalkan nama saya {self.nama} dari
{self.asal}')
```

Objek di atas sangat sederhana, ia hanya memiliki 2 atribut (nama dan asal) serta memiliki satu buah perilaku yaitu perkenalan().

Kita bisa membuat instance dari kelas Orang seperti biasanya:

```
andi = Orang('Andi', 'Surabaya')
andi.perkenalan()
```

Output:

```
Perkenalkan nama saya Andi dari Surabaya
```

Membuat Objek Turunan (Child)
Langkah selanjutnya adalah: membuat objek turunan dari kelas Orang. Sesuai dengan skenario yang telah kita bahas di atas, kita akan membuat dua buah objek baru yaitu objek Pelajar dan Pekerja, yang mana keduanya akan mewarisi objek Orang.

Caranya bagaimana?

Kita bisa membuat kelas turunan dengan cara mengirimkan kelas induk sebagai parameter saat mendefinisikan kelas.

Perhatikan contoh berikut:

```
class Pelajar (Orang):
  pass

class Pekerja (Orang):
  pass
```

Kita telah membuat dua buah kelas yang keduanya sama-sama memiliki setiap atribut dan fungsi dari kelas Orang.

Kita meletakkan perintah pass karena kita hanya ingin melakukan pewarisan apa adanya tanpa menambahkan apa pun lagi.

Sehingga, kita bisa membuat instance dari kelas Pelajar dan Pekerjaan, serta memanggil fungsi perkenalan() dengan cara yang benar-benar identik dengan kelas Orang:

```
andi = Orang('Andi', 'Surabaya')
andi.perkenalan()

deni = Pelajar('Deni', 'Makassar')
deni.perkenalan()

budi = Pekerja('Budi', 'Pontianak')
budi.perkenalan()
```

Output:

```
Perkenalkan nama saya Andi dari Surabaya
Perkenalkan nama saya Deni dari Makassar
Perkenalkan nama saya Budi dari Pontianak
```

Membuat Konstruktor Pada Kelas Turunan
Konstruktor pada kelas turunan memiliki perilaku yang sedikit berbeda dengan konstruktor yang terdapat pada kelas induk.

Apa yang terjadi ketika kelas turunan memiliki konstruktor sendiri?

Ia akan menimpa konstruktor dari kelas induk sehingga konstruktor kelas induk tidak akan pernah dieksekusi.

Coba kita ubah konstruktor pada kelas Orang menjadi seperti berikut:

```
class Orang:

  def __init__ (self, nama, asal):
    self.nama = nama
    self.asal = asal
    print('fungsi Orang.__init__() dieksekusi')
```

Jalankan kembali program kita, kita akan mendapatkan output seperti berikut:

```
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Andi dari Surabaya
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Deni dari Makassar
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Budi dari Pontianak
```

Sekarang, kita tambahkan konstruktor pada masing-masing kelas Pelajar dan Pekerja.

```
def __init__ (self, nama, asal):
  self.nama = nama
  self.asal = asal
```

Jalankan kembali kode program kita, dan kita akan mendapatkan output yang berbeda:

```
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Andi dari Surabaya
Perkenalkan nama saya Deni dari Makassar
Perkenalkan nama saya Budi dari Pontianak
```

Sangat jelas dari output di atas, bahwa konstruktor kelas Orang tidak lagi dieksekusi ketika kita membuat instance dari kelas Pelajar dan Pekerja. Hal itu terjadi karena sekarang kedua kelas tersebut telah memiliki konstruktor sendiri.

```
Fungsi super().__init__() atau KelasInduk.__init__()
```

Menimpa konstruktor kelas induk adalah ide yang buruk. Karena hal tersebut akan menghilangkan sebagian dari “pewarisan” itu sendiri. Di sisi lain, menambahkan fungsi konstruktor ke dalam kelas turunan ternyata justru menimpa fungsi konstruktor milik kelas induk.

Lalu bagiamana solusisnya?

Solusinya adalah dengan memanggil fungsi konstruktor pada kelas induk secara implisit.

Caranya ada 2:

- menggunakan fungsi super().__init__()
- menggunakan KelasInduk.__init__()

Apa beda keduanya?

Untuk cara pertama: super().__init__() kita cukup memanggil fungsinya seperti biasanya, tidak perlu mendefinisikan lagi data self.

Ada pun untuk cara yang kedua, kita harus mengirimkan data self secara manual. Agar lebih jelas, mari kita langsung praktikkan keduanya. Kita coba cara yang pertama untuk kelas Pelajar, dan cara yang kedua untuk kelas Pekerja:

```
class Pelajar (Orang):
  def __init__ (self, nama, asal):
    super().__init__(nama, asal)

class Pekerja (Orang):
  def __init__ (self, nama, asal):
    Orang.__init__(self, nama, asal)
```

Jalankan kembali aplikasi, dan kita akan mendapatkan output sebagai berikut:

```
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Andi dari Surabaya
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Deni dari Makassar
fungsi Orang.__init__() dieksekusi
Perkenalkan nama saya Budi dari Pontianak
```

Sekarang fungsi konstruktor dari kelas induk (yaitu kelas Orang) tetap terpanggil dari kelas turunannya.

C. Menambahkan Properti Baru Yang Tidak Ada Pada Induk

Seperti yang telah berlalu penjelasannya:

Bahwa kelas turunan akan memiliki semua sifat dan perilaku dari kelas induk, akan tetapi belum tentu kelas induk memiliki semua sifat dan perilaku dari kelas turunannya.

Itu artinya, kita bisa menambahkan atribut tertentu atau fungsi tertentu pada kelas turunan yang tidak ada pada kelas induk.

Bagaimana caranya?

Ya tinggal tambahkan saja, sederhana sekali 😁

Perhatikan contoh berikut, kita akan menambahkan atribut sekolah untuk kelas Pelajar, dan atribut tempat_kerja untuk kelas Pekerja:

```
class Pelajar (Orang):
  def __init__ (self, nama, asal, sekolah):
    super().__init__(nama, asal)
    self.sekolah = sekolah

class Pekerja (Orang):
  def __init__ (self, nama, asal, tempat_kerja):
    Orang.__init__(self, nama, asal)
    self.tempat_kerja = tempat_kerja
```

Dan terakhir, kita sesuaikan kode ketika membuat instance dari 2 kelas tersebut:

```
deni = Pelajar('Deni', 'Makassar', 'SMA Negeri 1 Makassar')
deni.perkenalan()
print(f'Saya sekolah di {deni.sekolah}')

budi = Pekerja('Budi', 'Pontianak', 'Google')
budi.perkenalan()
print(f'Saya bekerja di {budi.tempat_kerja}')
```

Oiya, sebelumnya kalian juga bisa menghapus kode yang menampilkan output fungsi Orang.__init__() dieksekusi karena kita sudah tidak membutuhkannya lagi.

Sehingga output akhir dari pertemuan kali ini adalah:

```
Perkenalkan nama saya Andi dari Surabaya
Perkenalkan nama saya Deni dari Makassar
Saya sekolah di SMA Negeri 1 Makassar
Perkenalkan nama saya Budi dari Pontianak
Saya bekerja di Google
```

=== To be Continue ===

REFERENSI

[1] N. Huda, "Jago Ngoding," 16 March 2021. [Online]. Available: https://jagongoding.com/python/menengah/oop/pewarisan/. [Accessed 28 November 2023].