LAPORAN PRAKTIKUM PBO

KELAS ABSTRAK, INTERFACE, DAN METACLASS

Andreas Pascalis Tristan(121140017)/ RB

Institut Teknologi Sumatera

2023

RINGKASAN

Kelas Abstrak

Class abstrak adalah class yang masih dalam bentuk abstrak. [1] Karena bentuknya masih abstrak, dia tidak bisa dibuat langsung menjadi objek. Class ini berperan sebagai sebuah kerangka dasar bagi class turunannya. Dalam sebuah class abstract setidaknya memiliki satu atau lebih abstract method. Abstract method bisa berupa method tanpa isi dengan parameter (bila ada). Abstract class akan digunakan dalam inheritance kelas turunannya agar seluruh kelas turunannya harus menggunakan method yang sama/override method. Dalam pyhton abstract class dapat diimplementasikan menggunakan Abstract Base Class (ABC) dari modul abc [2], yang berfungsi untuk mendecorate abstract method pada class abstract dengan syntax (@abstract method). Sebelum inisiasi.

Ilustrasi/contoh :

```
from abc import *

#Membuat abstract class
class Hewan(ABC):

    #Membuat method
    @abstractmethod
    def bersuara(self):
        pass
```

Gambar diatas adalah bentuk contoh untuk membuat sebuah abstract class. Pengambila modul abstract class diambil dari ABC dari modul abc. Dengan penggunaan @abstractmethod sebagai decorator. Implementasi dari penggunaan abstract class adalah sebagai berikut :

```
from abc import *

#Membuat abstract class
class Hewan(ABC):

    #Membuat method
    @abstractmethod
    def bersuara(self):
        pass

class Kucing(Hewan):
    def bersuara(self):
        return "meong"
```

Pada gambar di atas kelas Kucing merupakan turunan dari Hewan, sehingga penggunaan method harus sama dengan kelas Hewan. Dalam pembuatannya abstract class tidak hanya sebatas dalam abstract method, dalam membuatnya kita tetap bisa membuat concrete method/fungsi biasa.

```python
from abc import *

#Membuat abstract class
class Hewan(ABC):

    def __init__(self, nama):
        self.nama = nama
    #Membuat method
    @abstractmethod
    def bersuara(self):
        pass

    def tidur(self):
        return(f"Si {self.nama} sedang tidur. ", )
```

Pada gambar di atas meski kelas hewan merupakan sebuah abtract class, kita tetap bisa membuat sebuah concrete method, yag sifatnya bisa diwariskan maupun di override.

```python
class Kucing(Hewan):
    def bersuara(self):
        return "meong"

    def tidur(self):
        super().tidur()
        print(self.nama + " si kucing hitam tertidur.")

kit = Kucing("kitty")
print("suaranya " + kit.bersuara())
kit.tidur()
```

Pada gambar di atas fungsi tidur pada Kucing mengoverride fungsi tidur pada Hewan.

```
suaranya meong
kitty si kucing hitam tertidur.
PS C:\>
```

Gambar di atas merupakan contoh output dari fungsi bersuara dan tidur.

Interface

Interface adalah sebuah ‘kontrak’ atau perjanjian implementasi method [3]. Bagi *class* yang menggunakan *object interface*, *class* tersebut harus mengimplementasikan ulang seluruh *method* yang ada di dalam *interface*. Dalam pemrograman objek, penyebutan *object interface* sering disingkan dengan ‘*Interface’* saja. Interface sendiri bisa disebut sebagai bentuk lain dari abstract class, tetapi dengan pengertian dan tujuan yang berbeda. Sebuah interface dalam Bahasa python diimplementasikan dengan sebuah class dan adalah sebuah subclass dari interface yang merupakan parent dari semua interface yang ada. Pengimpplementasian interface akan membuat souce code menjadi lebih rapih [4].

- Informal Interface
  Informal Interface adalah bentuk kelas yang didefinisikan method atau fungsi yang bisa di override/implementasi namun tidak diharuskan untuk diimplementasikan.

```
class aritmatika:
    def __init__(self, x):
        self.x = x

    def __add__(self, y):
        return self.x + y.x

    def __sub__(self, y):
        return self.x - y.x

    def __mul__(self, y):
        return self.x * y.x

class aritmatika_lanjutan(aritmatika):
    pass

bilangan1 = aritmatika_lanjutan(5)
bilangan2 = aritmatika_lanjutan(3)

print(bilangan1 - bilangan2)
print(bilangan1 * bilangan2)
print(bilangan1 + bilangan2)
```

Pada gambar di atas, kelas “aritmatika” dapat menggunakan melakukan fungsi pengurangan, perkalian, dan penjumlahan menggunakan (__sub__), (__mul__), dan (__add__).

```
2
15
8
PS C:\>
```

Inilah hasil output dalam penggunaan interface, namun perlu diingat bahwa informal interface tidak mengharuskan implementasi fungsi. Bila kita mau kita dapat melakukan override pada kelas turunan.

```
class aritmatika:
    def __init__(self, x):
        self.x = x

    def __sub__(self, y):
        return self.x - y.x

    def __mul__(self, y):
        return self.x * y.x

class aritmatika_lanjutan(aritmatika):
    def __add__(self, y):
        return self.x + y.x

bilangan1 = aritmatika_lanjutan(5)
bilangan2 = aritmatika_lanjutan(3)

print(bilangan1 - bilangan2)
print(bilangan1 * bilangan2)
print(bilangan1 + bilangan2)
```

Pada implementasi kali ini __add__ kita letakkan pada kelas turunan sehingga fungsi penjumlahan tetap bisa dilakukan, bisa dilihat bahwa peletakan __add__ tidak diharuskan pada class parent, kita tetap bisa meletakkannya pada class child karena informal interface tidak memaksakan penempatan tersebut.

```
2
15
8
PS C:\>
```

Hasil output akan tetap sama walau sekarang method __add__ terletak pada class child.

- Formal Interface
  Berkebalikan dengan informal interface, formal interface akan memaksa setiap method yang dimiliki class parent dapat diimplementasikan pada class turunan. Pada formal interface class parent akan dibentuk degan kode sederhana/singkat, yang mana implementasi lengkapnya akan diimplemetasikan dalam class turunannya. Formal interface dibentuk dengan penggunaan abstract method.

```python
from abc import *

class Manusia:

    @abstractmethod
    def jenis_kelamin(self):
        pass

    @abstractmethod
    def mata(self):
        pass

    @abstractmethod
    def usia(self):
        pass
```

Pada gambar di atas merupakan penggunaan kelas abstrak. Untuk menggunakan abstract method dalam formal interface digunakan @abstractmehtod sebagai decorator sehingga class turunan dari Manusia akan menggunakan semua method dalam Manusia.

```python
class detail:
    def jenis_kelamin(self):
        return "laki-laki"
    def mata(self):
        return "merah"
    def usia(self):
        return "18"
```

Inilah bentuk kelas konret dalam turunan kelas abstrak. Semua fungsi dalam kelas Manusia akan dan harus diimplementasikan dalam class turunannya, inilah contoh formal interface.

Metaclass

Metaclass adalah salah satu konsep dalam OOP dalam Bahasa python [5]. Metaclass berperan untuk mendefinisikan bagaimana sebuah kelas berperilaku. Dalam konteksnya semua class adalah bagian dari metaclass. Dalam python terdapat keyword yang digunakan untuk metaclass yaitu (type). Type adalh metaclass yang instansnya merupakan class. Jadi segala class dalam python adalah sejatinya turunan dari type.

```python
class parent:
    pass

objek = parent()

print(type(objek))
print(type(parent))
```

```
<class '__main__.parent'>
<class 'type'>
PS C:\>
```

Dari gambar diatas dapat dilihat jenis output yang dihasilkan bahwa objek “objek” adlah parent dan tipe dari kelas parent tersebut adalah “type”.

- Bentuk implementasi dengan type.

```
gku = type('301', (), {'pagi' : 'PBO', 'sore' : 'stigma'})
print("Ruangan senin : ", gku)
print("Kelas pagi : ", gku.pagi)
print("Kelas sore : ", gku.sore)
```

```
Ruangan senin :  <class '__main__.301'>
Kelas pagi :  PBO
Kelas sore :  stigma
<__main__.301 object at 0x0000022B6A38AC80>
PS C:\>
```

Pada gambar kita membuat sebuah kelas dengan nama “gku” dengan alamat 0x0000022B6A38AC80. Untuk membuat kelas turunan type, digunakan bagian tuple () dengan nama parent yang diinginkan.

- Bentuk implementasi dengan parameter.

```
class parent(type):
    pass

class childe(metaclass=parent):
    pass
```

Pada gambar di atas adalah bentuk metaclass, maka kelas turunan dari type adalah metaclass juga.

Kesimpulan

- Apa itu interface dan kapan digunakannya?
  Interface merupakan salah satu konsep OOP tentang kumpulan method dalam kelas parent yang implementasiannya dilakukan secara formal dan informal. Konsep ini akan kita gunakan untuk membuat kelas turunan dari kelas parent. Ketika kita mau setiap method dalam kelas Parent ke turunannnya maka akan digunakan konsep interface formal, bila sebaliknya kita akan menggunakan konsep informal.
- Apa itu kelas abstrak dan kapan kita perlu memakainya? Apa perbedaannya dengan interface?
  Kelas abstrak adalah class yang tidak dapat diinisialisasi, sehingga peran utama kelas abstrak adalah sebatas sebagai kerangka turunannya. Kelas abstrak akan digunakan untuk fitur umum yang dimiliki leh kelas turunannya. Untuk perbedaanya dengan interface, interface adalah konsep berupa kontrak untuk kelas turunannya.
- Apa itu kelas konkret dan kapan kita meggunakannya?
  kelas konkret adalah jenis kelas yang dapat di gunakan sebagai objek. Kelas konkret dibuat dalam sebuah kelas abstract. Berbeda dengan kelas abstract yang tidak dapat dibuat sebagai objek, sehingga peran kelas konkret adalah untuk membantu class abstract.
- Apa itu metaclass dan kapan kita perlu memakainya? Apa bedanya dengan inherintance biasa? Metaclass adalah jenis kelas yang digunakan untuk membuat kelas lain. Metaclass digunakan untuk memodifikasi perilaku kelas yang dibuat dengan menggunakan kelas tersebut. Perbedaan dasar dari metaclass dan inheritance adalah metaclass digunakan untuk memodifikasi sedangkan inheritance hanya sebatas mewarisi kelas parent.

DAFTAR PUSTAKA

Bibliography

[1] bestharadhakrishna, "Abstract Classes in Python," geeksforgeeks, 10 October 2014. [Online]. Available: https://www.geeksforgeeks.org/abstract-classes-in-python/.

[2] A. N., "How to Use Abstract Classes in Python," Towards Data Science, 17 October 2021. [Online]. Available: https://towardsdatascience.com/how-to-use-abstract-classes-in-python-d4d2ddc02e90.

[3] B. Kumar, "Introduction to Python Interface," PythonGuides, 24 December 2020. [Online]. Available: https://pythonguides.com/python-interface/.

[4] W. Murphy, "Implementing an Interface in Python," Real Python, [Online]. Available: https://realpython.com/python-interface/.

[5] D. Mwiti, "Introduction to Python Metaclasses," datacamp, December 2018. [Online]. Available: https://www.datacamp.com/tutorial/python-metaclasses.