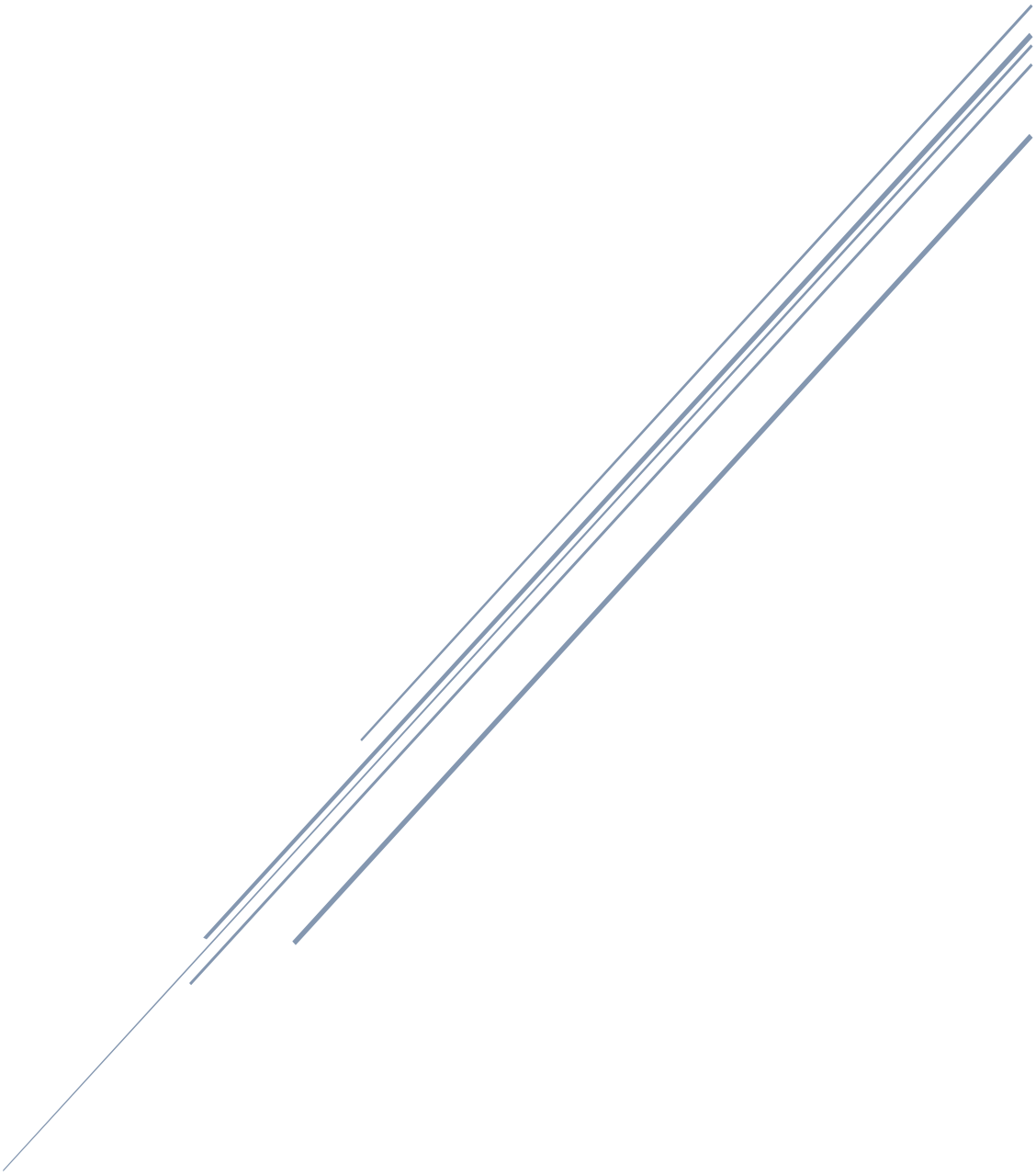
LAPORAN PRAKTIKUM PBO

Kelas Abstrak, Interface, Metaclass

Richard Arya Winarta [121140035] RB



Institut Teknologi Sumatera

2023

Ringkasan

Kelas Abstrak

Abstract Class adalah sebuah class yang tidak bisa di-instansiasi (tidak bisa dibuat menjadi objek). Abstract class berperan sebagai ‘kerangka dasar’ bagi class turunannya. Di dalam abstract class umumnya akan memiliki abstract method. Abstract Method sendiri adalah adalah sebuah ‘method dasar’ yang harus diimplementasikan ulang di dalam class anak (child class). Abstract method bisa berupa method tanpa isi dengan parameter (jika ada). Abstract class digunakan dalam inheritance kelas turunannya agar seluruh kelas turunannya dipaksa untuk mengimplementasi method yang sama (override method) dengan abstract class. Dalam bahasa pemrograman python, konsep pengimplementasian abstract class harus menggunakan Abstract Base Classes (ABC) dari modul abc. Modul ini berfungsi untuk mendecorate abstract method pada class abstract dengan menambahkan syntax (@abstractmethod) sebelum inisiasi method.

Ilustrasi/ contoh :

```python
from abc import *

# pembuatan abstract class
class Hewan(ABC):

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

Gambar di atas adalah potongan kode untuk membuat sebuah abstract class. Karena dalam python konsep abstract class harus diambil dari modul lain, maka abstract class yang akan dibuat harus mewarisi class ABC dari modul abc. Untuk membuat abstract method, perlu menambahkan decorator @abstractmethod. Decorator ini berfungsi agar class turunan abstract class harus mengimplementasikan method di abstract class.

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python/Python311/python.exe "d:/Python
/praktikum pbo/test_praktikum.py"
Traceback (most recent call last):
  File "d:\Python\praktikum pbo\test_praktikum.py", line 11, in <module>
    objek = Hewan()
            ^^^^^^^
```

Hasil output error akan dihasilkan apabila abstract class diinisasi menjadi sebuah objek.

```python
from abc import *

# pembuatan abstract class
class Hewan(ABC):

    # pembuatan abstract method
    @abstractmethod
    def bersuara(self):
        pass

# class turunan dari abstract class
class Sapi(Hewan):

    # method yang wajib diimplementasikan
    def bersuara(self):
        return "mooooooo"
```

Gambar di atas adalah potongan kode untuk membuat kelas turunan dari abstract class. Kelas turunan ini harus mengimplementasikan setiap abstract method yang telah dibuat di abstract class. Karena ini merupakan kelas turunan dari abstract method, maka kelas ini bisa diinisiasi menjadi sebuah objek.

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python/Python311/python.exe "d:/Python
/praktikum pbo/test_praktikum.py"
Traceback (most recent call last):
  File "d:\Python\praktikum pbo\test_praktikum.py", line 15, in <module>
    sapi = Sapi()
           ^^^^^^
TypeError: Can't instantiate abstract class Sapi with abstract method bersuara
```

Hasil output error akan dihasilkan apabila kelas turunan tidak mengimplementasikan abstract method.

```
sapi = Sapi()
print("Suara sapi adalah " + sapi.bersuara())
```

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python
/Python311/python.exe "d:/Python/praktikum pbo/test_praktikum.py"
Suara sapi adalah mooooooooo
```

Gambar di atas adalah potongan kode dan hasil output untuk penginisiasian objek class turunan “Sapi” dari abstract class “Hewan”.

```
# pembuatan abstract class
class Hewan(ABC):
    # constructor pada abstract class
    def __init__(self, nama):
        self.nama = nama

    # pembuatan abstract method
    @abstractmethod
    def bersuara(self):
        pass

    # pembuatan concrete method
    def makan(self):
        print(f"hewan {self.nama} makan ", end='')
```

Gambar di atas adalah potongan kode yang menunjukkan bahwa selain abstract method, abstract class dapat memiliki konstruktor dan juga fungsi biasa (concrete method). Fungsi ini nantinya akan diwariskan ke kelas turunannya ataupun di override oleh kelas turunannya (tidak wajib).

```
# class turunan dari abstract class
class Sapi(Hewan):
    # method yang wajib diimplementasikan
    def bersuara(self):
        return "mooooooooo"

    def makan(self):
        super().makan()
        print("rumput")

sapi = Sapi("Tono")
print("Suara sapi adalah " + sapi.bersuara())
sapi.makan()
```

Gambar di atas adalah potongan kode untuk proses override fungsi normal abstract class “Hewan” di kelas turunan “Sapi” dan proses pemanggilan fungsi makan objek “sapi”.

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python
/Python311/python.exe "d:/Python/praktikum pbo/test_praktikum.py"
Suara sapi adalah moooooooo
hewan Tono makan rumput
```

Gambar di atas adalah hasil output dari 2 potongan kode sebelumnya.

Interface

Secara sederhana, interface adalah sebuah ‘kontrak’ atau perjanjian implementasi method. Interface adalah koleksi dari method atau fungsi yang telah disediakan dan perlu untuk diimplementasikan oleh child class. Interface mengandung method yang abstract secara defaultnya. Metode abstrac akan hanya memiliki 1 kali deklarasi karena tidak adanya implementasi. Sebuah interface di bahasa python diimplementasikan dengan sebuah class dan adalah sebuah subclass dari interface yang merupakan parent dari semua interface yang ada. Pengimplementasian interface akan membuat source code menjadi lebih elegan dan terorganisir.

- Informal Interface
  Informal interface adalah kelas yang mendefinisikan method atau fungsi yang bisa di override/implementasi namun tanpa adanya unsur paksaan untuk diimplementasikan. Untuk mengimplementasikan konsep informal interface, perlu dibuat sebuah kelas konkrit.

  Ilustrasi / contoh :

```python
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):
    pass

bilangan1 = aritmatika_lanjutan(3)
bilangan2 = aritmatika_lanjutan(2)

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

  Pada gambar di atas, kelas “aritmatika” dapat melakukan pengurangan (__sub__) dan perkalian (__mul__) terhadap 2 objek berbeda dari kelas turunan “aritmatika” yaitu “aritmatika_lanjutan”.

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python/Python
311/python.exe "d:/Python/praktikum pbo/test_praktikum.py"
1
6
Traceback (most recent call last):
  File "d:\Python\praktikum pbo\test_praktikum.py", line 19, in <module>
    print(bilangan1 + bilangan2)
          ~~~~~~~~~~~~^~~~~~~~~~~~
TypeError: unsupported operand type(s) for +: 'aritmatika_lanjutan' and 'aritmatika
_lanjutan'
```

Hasil output untuk proses pengurangan dan perkalian dapat ditunjukkan, namun proses pertambahan menghasilkan error karena fungsi untuk mengeksekusi proses pertambahan belum diimplementasikan. Untuk mengatasi ini, bisa dilakukan pendefinisian fungsi pertambahan di class child.

```python
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(3)
bilangan2 = aritmatika_lanjutan(2)

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

Pada gambar di atas, fungsi __add__ perlu diimplementasikan karena dibutuhkan untuk proses pertambahan. Namun implementasi __add__ di kelas turunan “aritmatika_lanjutan” tidak dipaksakan oleh kelas parent “aritmatika”. Pengimplementasian __add__ di kelas child dilakukan secara informal (tidak memaksa) sehingga ini disebut sebagai interface informal.

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python
/Python311/python.exe "d:/Python/praktikum pbo/test_praktikum.py"
1
6
5
```

Hasil output untuk proses pengurangan, perkalian, dan pertambahan.

- Formal Interface

  Formal interface memaksa untuk setiap method yang dimiliki oleh kelas parent untuk diimplementasikan pada kelas turunannya. Pada formal interface, method kelas parent dapat dibangun hanya dengan sedikit baris kode, yang nantinya akan lebih dispesifikkan di kelas turunannya. Karena wajibnya pengimplementasian method, maka konsep ini disebut formal interface. Pengimplementasian formal interface dapat menggunakan abstract method.

Ilustrasi / Contoh :

```python
from abc import *

class kepala(ABC):

    @abstractmethod
    def mata(self):
        pass

    @abstractmethod
    def rambut(self):
        pass

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

Gambar di atas merupakan contoh dari kelas abstrak. Untuk membuat abstract method yang mendukung konsep formal interface, ditambahkan decorator @abstractmethod. Dekorator ini akan membuat kelas turunan dari “kepala” untuk mengimplementasikan method-methodnya.

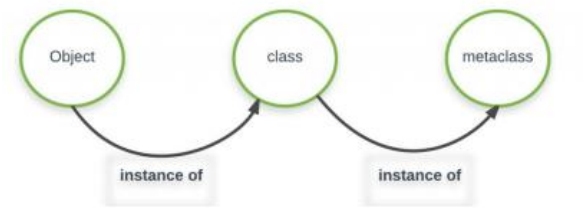
```python
class manusia(kepala):
    def mata(self):
        return "biru"

    def rambut(self):
        return "panjang"

    def otak(self):
        return "besar"
```

Gambar di atas merupakan contoh kelas konkret yang diperoleh dari turunan kelas abstrak. Karena kelas abstrak “kepala” memiliki beberapa abstract method, maka kelas turunan harus mengimplementasikan semua abstract method dari abstract class.

Metaclass

Metaclass adalah sebuah konsep OOP di bahasa python. Metaclass adalah class dari sebuah class yang mendefinisikan bagaimana sebuah class bekerja/berperilaku. Sebuah class sendiri termasuk dalam metaclass. Bahasa python menyediakan fungsionalitas untuk membuat metaclass custom dengan menggunakan keyword (type). Type adalah metaclass yang instansnya adalah class. Jadi kelas apapun yang dibuat dalam pemrograman python, sejatinya adalah turunan dari type metaclass. Metaclass digunakan untuk memodifikasi perilaku kelas-kelas yang dibuat dengan menggunakan kelas tersebut.

Ilustrasi :



Metaclass bisa dikatakan sebagai tipe kelas spesial yang merupakan parent dari semua kelas-kelas pada python (class factory).

```
class Parent:
    pass

objek = Parent()

print(type(objek))
print(type(Parent))
```

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python/Python311/python.exe "d:/Python
/praktikum pbo/test_praktikum.py"
<class '__main__.Parent'>
<class 'type'>
```

Dari hasil output bisa dilihat bahwa tipe dari objek “objek” adalah “Parent” dan ditipe dari kelas “Parent” adalah “type”. Hal ini karena adanya konsep hierarki yang dijelaskan pada ilustrasi sebelumnya.

- Implementasi metaclass dengan keyword <type>
  Untuk membuat sebuah objek metaclass dapat menggunakan keyword type.
  Ilustrasi / Contoh :

```
panggil = type('makan', (), {'manis': 'coklat',
                             'asin' : 'garam'})

print("kelas baru : ", panggil)
print("makanan manis : ", panggil.manis)
print("makanan asin : ", panggil.asin)
print(panggil())
```

```
PS D:\Python\praktikum pbo> & C:/Users/Richard/AppData/Local/Programs/Python/Python311/python.exe "d:/Python
/praktikum pbo/test_praktikum.py"
kelas baru :  <class '__main__.makan'>
makanan manis :  coklat
makanan asin :  garam
<__main__.makan object at 0x000001DFF5A2E490>
```

Sebuah class terbentuk dengan nama ‘makan’ yang juga merupakan sebuah objek di alamat 0x000001DFF5A2E490. Untuk membuat kelas turunan type, dapat mengisi bagian tuple () dengan nama parent class yang diinginkan.

- Implementasi metaclass dengan parameter

  Untuk membuat sebuah kelas turunan hasil modifikasi terhadap type itu sendiri, dapat menggunakan sebuah kelas turunan dengan basis dari type.

  Ilustrasi / Contoh :

```
class Parent(type):
    pass

class Child(metaclass=Parent):
    pass
```

  Karena type adalah sebuah metaclass, maka kelas turunan dari type adalah metaclass juga. Secara default, metaclass yang diginakan saat membuat kelas adalah type. Sehingga kelas “Parent” dan “Child” bisa dikatakan sebagai metaclass dan bisa kita modifikasi.

Kesimpulan

- Apa itu interface dan kapan kita perlu memakainya?
  Interface adalah sebuah konsep OOP tentang kumpulan method di kelas Parent yang pengimplementasiannya di kelas turunan dari kelas Parent dapat dilakukan secara formal (wajib) atau informal (tidak wajib). Konsep interface dipakai ketika kita akan membuat sebuah kelas turunan dari kelas Parent. Ketika kita mau menerapkan setiap method di kelas Parent ke kelas turunannya, maka kita akan menggunakan konsep interface formal, dan jika sebaliknya kita dapat menggunakan konsep interface informal.

- Apa itu kelas abstrak dan kapan kita perlu memakainya? Apa perbedaannya dengan interface?
  Kelas abstrak adalah class yang tidak bisa di-instansiasi, sehingga kelas abstrak hanya akan menjadi kerangka bagi kelas turunannya. Kelas abstrak digunakan ketika ada beberapa fitur umum / method yang akan dimiliki oleh semua objek kelas turunannya. Perbedaan yang membedakan kelas abstrak dan interface adalah dari konsepnya. Interface adalah konsep untuk kontrak/perjanjian implementasi method di kelas turunannya, sedangkan kelas abstrak yang mengadopsi konsep interface formal dalam pengimplementasiannya.

- Apa itu kelas konkret dan kapan kita perlu memakainya?
  Kelas konkret adalah kelas yang dapat langsung di-instansiasi menjadi objek. Kelas konkret dibuat ketika terdapat sebuah kelas abstract. Hal ini dikarenakan kelas abstract yang tidak bisa di-instansiasi menjadi objek. Sehingga membutuhkan bantuan kelas konkret yang dibentuk dari warisan kelas abstract.

- Apa itu metaclass dan kapan kita perlu memakainya? Apa bedanya dengan inheritance biasa?
  Metaclass adalah kelas yang digunakan untuk membuat kelas lain. Karena python merupakan bahasa pemrograman yang berorientasi OOP, maka kelas-kelas yang dibentuk di python pada dasarnya berasal dari metaclass. Metaclass digunakan untuk memodifikasi perilaku kelas-kelas yang dibuat dengan menggunakan kelas tersebut. Perbedaan dasar dari metaclass dan inheritance adalah metaclass dapat memodifikasi perilaku kelas yang dibuat dari kelas, sedangkan inheritance hanya mewarisi method, atribut dari kelas lain. Metaclass digunakan ketika pembuatan kelas baru, sedangkan konsep inheritance ketika kelas itu diturunkan.

Daftar Pustaka

Andre. (2014, October 10). *Tutorial Belajar OOP PHP Part 15 : Pengertian Abstract Class dan Abstract Method PHP*. Retrieved from Duniailkom: https://www.duniailkom.com/tutorial-belajar-oop-php-pengertian-abstract-class-dan-abstract-method-php/

bestharadhakrisna. (2021, March 19). *Abstract Classes in Python*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/abstract-classes-in-python/

Kumar, B. (2020, December 24). *Introduction to Python Interface*. Retrieved from PythonGuides: https://pythonguides.com/python-interface/

Mwiti, D. (2018, December). *Introduction to Python Metaclasses*. Retrieved from datacamp: https://www.datacamp.com/tutorial/python-metaclasses

Singh, S. (2022, August 18). *What is a metaclass in Python?* Retrieved from tutorialspoint: https://www.tutorialspoint.com/What-is-a-metaclass-in-Python#