

TUGAS MODUL 6

LAPORAN PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

KELAS ABSTRAK, INTERFACE, DAN METACLASS



Disusun Oleh :

Nama : Abira Husnia

NIM : 121140113

Kelas : RB

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA
2023

DAFTAR ISI

RINGKASAN.....	3
Kelas Abstrak.....	3
1. Konsep Abstraksi	3
Interface.....	5
1. Definisi	5
2. Informal Interface	5
3. Formal Intetface	6
Metaclass	8
1. Definisi	8
2. Implementasi metaclass Menggunakan type.....	8
3. Implementasi metaclass Menggunakan Parameter metaclass.....	9
KESIMPULAN	10
DAFTAR PUSTAKA	11

RINGKASAN

Kelas Abstrak

1. Konsep Abstraksi

Dalam pengembangan perangkat lunak, tidak semua hal yang dilihat dan digunakan adalah objek yang nyata. Beberapa juga bersifat abstrak. Abstraksi dalam konsep Pemrograman Berorientasi Objek (OOP) merupakan proses menciptakan objek yang hanya menampilkan atribut yang penting dan menyembunyikan detail-detail yang tidak relevan bagi pengguna. Tujuannya adalah untuk mengurangi kompleksitas. Pengguna dapat mengetahui apa yang objek lakukan, tetapi tidak perlu tahu bagaimana mekanisme yang sebenarnya terjadi di balik layar. Misalnya, saat mengendarai mobil, pengguna hanya perlu mengetahui cara menyalakan, menjalankan, atau menghentikan mobil, tanpa perlu memahami detail teknis tentang apa yang terjadi pada mobil saat menerima perintah tersebut.

a) Kelas Abstrak dalam python

Kelas abstrak dalam pemrograman adalah kelas yang tidak dapat diinstansiasi secara langsung dan berfungsi sebagai kerangka dasar bagi kelas turunannya. Biasanya, kelas abstrak akan memiliki satu atau lebih metode abstrak. Metode abstrak adalah metode yang harus diimplementasikan ulang di kelas turunan (child class). Metode abstrak ditulis tanpa isi (implementasi), hanya memiliki "signature" yang terdiri dari nama metode dan parameter (jika ada). Kelas abstrak digunakan dalam pewarisan (inheritance) untuk memaksa atau mengganti metode yang sama di seluruh kelas turunan dari kelas abstrak tersebut. Kelas abstrak digunakan untuk membentuk struktur logika penurunan dalam pemrograman berorientasi objek. Contohnya, penggunaan kelas abstrak dapat diterapkan dalam konsep mobil, di mana setiap mobil memiliki kesamaan tertentu tetapi memiliki implementasi yang berbeda dalam hal kecepatan, bahan bakar, dan sebagainya.

b) Implementasi Kelas Abstrak dengan modul ABC

Python mempunyai modul untuk menggunakan Abstract Base Classes (ABC). Pada kelas abstrak, fungsi yang memiliki karakteristik abstrak ditandai dengan penggunaan decorator `@abstractmethod` saat mendefinisikan fungsi tersebut. Dengan menggunakan modul ABC, Python memungkinkan pembuatan kelas abstrak dan definisi metode abstrak dengan mudah.

```
from abc import ABC, abstractmethod

class Hewan(ABC):
    @abstractmethod
    def berjalan(self):
        pass

    @abstractmethod
    def berlari(self):
        pass

# menginstansiasi objek dari kelas abstrak
Kucing = Hewan()
```

Gambar 1. Implementasi kelas abstrak

Kode di atas akan menghasilkan **TypeError: Can't instantiate abstract class Hewan with abstract methods berjalan, berlari**. Ini terjadi karena kelas **Hewan** adalah kelas abstrak dan memiliki metode **berjalan, berlari** yang merupakan metode abstrak. Kelas abstrak hanya dapat digunakan sebagai kelas dasar untuk kelas turunannya. Untuk menggunakan metode-metode dari kelas abstrak, perlu membuat kelas turunan atau 'child' yang mengimplementasikan metode-metode abstrak tersebut.

```
PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/ASUS/Downloads/prak 6.py"
Traceback (most recent call last):
  File "c:/Users/ASUS/Downloads/prak 6.py", line 13, in <module>
    Kucing = Hewan()
              ^^^^^^^
TypeError: Can't instantiate abstract class Hewan with abstract methods berjalan, berlari
```

Gambar 2. Implementasi kelas abstrak Error

Apabila membuat kelas 'child' dari sebuah kelas abstrak, harus mengimplementasikan kembali seluruh metode/fungsi yang ada pada kelas induk (kelas abstrak), seperti contoh pada gambar berikut:

```
1  from abc import ABC, abstractmethod
2
3  class Hewan(ABC):
4      @abstractmethod
5      def berjalan(self):
6          pass
7
8      @abstractmethod
9      def berlari(self):
10         pass
11
12  class Kucing(Hewan):
13      def __init__(self, nama):
14          self.nama = nama
15
16      def berjalan(self):
17          print(f"Kucing {self.nama} berjalan dengan empat kaki")
18
19      def berlari(self):
20          print(f"Kucing {self.nama} berlari dengan kecepatan tinggi")
21
22  kucing = Kucing("Catty")
23
24  # memanggil metode dari objek
25  kucing.berjalan()
26  kucing.berlari()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/ASUS/Downloads/prak 6.py"
Kucing Catty berjalan dengan empat kaki
Kucing Catty berlari dengan kecepatan tinggi
```

Gambar 3. Implementasi kelas child

Pada contoh di atas, membuat kelas **Kucing** yang merupakan kelas child dari **Hewan**, sebuah kelas abstrak, mengimplementasikan kembali metode **berjalan** dan **berlari** yang ada di kelas abstrak. Kemudian, membuat objek **kucing** dari kelas **Kucing**, yang merupakan kelas turunan dari **Hewan**. Setelah itu, memanggil metode **berjalan** dan **berlari** dari objek **kucing** untuk mendapatkan output yang diharapkan. Selanjutnya, menambahkan method **__init__** pada kelas **Kucing** yang menerima satu parameter **nama**. Di dalam method **__init__**, menginisialisasi atribut **nama** dengan nilai yang diberikan saat membuat objek. Selanjutnya, ketika memanggil metode **berjalan** dan **berlari**, menggunakan atribut **nama** untuk memberikan informasi yang lebih spesifik tentang kucing yang sedang berjalan dan berlari.

Interface

1. Definisi

Interface adalah kumpulan metode atau fungsi yang harus disediakan oleh kelas yang mengimplementasikannya (kelas turunan). Interface terdiri dari metode yang bersifat abstrak, yang hanya memiliki deklarasi tanpa implementasi. Pengimplementasian sebuah interface merupakan cara untuk menulis kode yang elegan dan terorganisir. Perbedaan antara interface dan abstract class adalah sebagai berikut:

- Semua metode dalam sebuah interface adalah abstrak, sedangkan dalam abstract class, metode bisa berupa abstrak maupun konkrit.
- Interface digunakan ketika setiap fitur perlu diimplementasikan secara berbeda untuk setiap objek yang berbeda. Abstract class digunakan ketika ada beberapa fitur umum yang dimiliki oleh semua objek.
- Interface cenderung lebih lambat dalam kinerjanya dibandingkan dengan abstract class karena semua implementasi metode harus ditulis pada kelas turunan.
- Abstract class lebih cepat karena tidak semua implementasi perlu ditulis pada setiap kelas turunan.

Dengan demikian, interface memberikan fleksibilitas dalam menerapkan fungsionalitas yang berbeda untuk objek yang berbeda, sementara abstract class memberikan keleluasaan dalam membagikan fitur umum kepada kelas turunan.

2. Informal Interface

Dalam Python, pengimplementasian interface memiliki pendekatan yang berbeda dibandingkan dengan bahasa lain seperti C++ dan Java. Di Python, interface sering kali diartikan sebagai sebuah konsep yang tidak memiliki aturan yang terlalu ketat. Informal interface dalam Python mengacu pada kelas yang mendefinisikan metode atau fungsi yang dapat di-override atau diimplementasikan, tetapi tidak ada paksaan yang mengharuskan implementasi tersebut (hanya diimplementasikan jika diperlukan). Untuk mengimplementasikannya, perlu membuat kelas konkret yang merupakan subclass dari interface (biasanya dalam bentuk kelas abstrak) dan memberikan implementasi untuk metode atau fungsi yang ada di dalam interface. Implementasi ini dilakukan melalui inheritance (pewarisan).

```
1 class Hewan:
2     def __init__(self, hewan):
3         self.__list_hewan = hewan
4
5     def __len__(self):
6         return len(self.__list_hewan)
7
8     def __contains__(self, hewan):
9         return hewan in self.__list_hewan
10
11 class Zoo(Hewan):
12     def __iter__(self):
13         return iter(self.__list_hewan)
14
15 gembiraloka = Zoo(["Jerapah", "Kancil", "Beruang", "Macan"])
16
17 print(len(gembiraloka))
18 print("Jerapah" in gembiraloka)
19 print("Kancil" in gembiraloka)
20 print("Beruang" not in gembiraloka)
21 print("Macan" not in gembiraloka)
22 for hewan in gembiraloka:
23     print(hewan)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
4
True
False
Jerapah
Kancil
Beruang
Macan
```

Gambar 4. Implementasi informal interface

Pada contoh di atas class **Hewan** memiliki konstruktor `__init__` yang menerima argumen **hewan**, `__list_hewan` adalah variabel instance yang menyimpan daftar hewan yang diberikan saat objek **Hewan** dibuat, `__len__` digunakan untuk mengembalikan panjang daftar hewan (`__list_hewan`) ketika fungsi `len()` dipanggil pada objek **Hewan**, `__contains__` digunakan untuk memeriksa apakah suatu hewan ada dalam daftar hewan (`__list_hewan`) ketika menggunakan operator `in` pada objek **Hewan**, class **Zoo** adalah kelas turunan dari **Hewan**. Dalam kelas ini, `__iter__` diimplementasikan untuk mengubah objek **Zoo** menjadi iterable, dengan menggunakan `iter()` pada `__list_hewan` dari kelas **Hewan**, dan **gembiraloka** dibuat sebagai instance dari kelas **Zoo** dengan daftar hewan yang diberikan. Kemudian, berbagai operasi dan iterasi dilakukan pada objek **gembiraloka**. Output dari contoh di atas menunjukkan bahwa **gembiraloka** memiliki jumlah hewan 4, "Jerapah" dan "Kancil" ada dalam **gembiraloka**, "Beruang" dan "Macan" tidak ada dalam **gembiraloka**, dan saat melakukan iterasi pada **gembiraloka**, semua hewan dalam daftar (`__list_hewan`) akan dicetak.

3. Formal Interface

Formal interface adalah sebuah konsep dalam pemrograman berorientasi objek di mana kelas abstrak dapat menetapkan metode yang harus diimplementasikan oleh kelas-kelas turunannya. Dalam formal interface, implementasi metode tersebut menjadi wajib dan dipaksakan pada kelas turunan. Cara yang umum digunakan untuk membuat formal interface adalah dengan menggunakan metode abstrak pada kelas abstrak. Metode abstrak adalah metode yang dideklarasikan tanpa implementasi di kelas abstrak, dan kelas turunan yang mewarisi kelas abstrak harus mengimplementasikan metode tersebut.

Untuk membuat formal interface, perlu membuat sebuah kelas abstrak yang memiliki satu atau lebih metode abstrak. Metode abstrak ditandai dengan decorator `@abstractmethod`. Kelas-kelas turunan dari kelas abstrak ini akan mengimplementasikan metode-metode tersebut sesuai dengan kebutuhan mereka.

```
from abc import ABC, abstractmethod

class Hewan(ABC):
    @abstractmethod
    def suara(self):
        pass

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

Gambar 5. Contoh kelas abstrak

```
class Kucing(Hewan):
    def suara(self):
        return "Meow"

    def makan(self):
        return "Kucing itu makan ikan"

catty = Kucing()
print(catty.suara())
print(catty.makan())
```

Gambar 6. Contoh kelas konkret

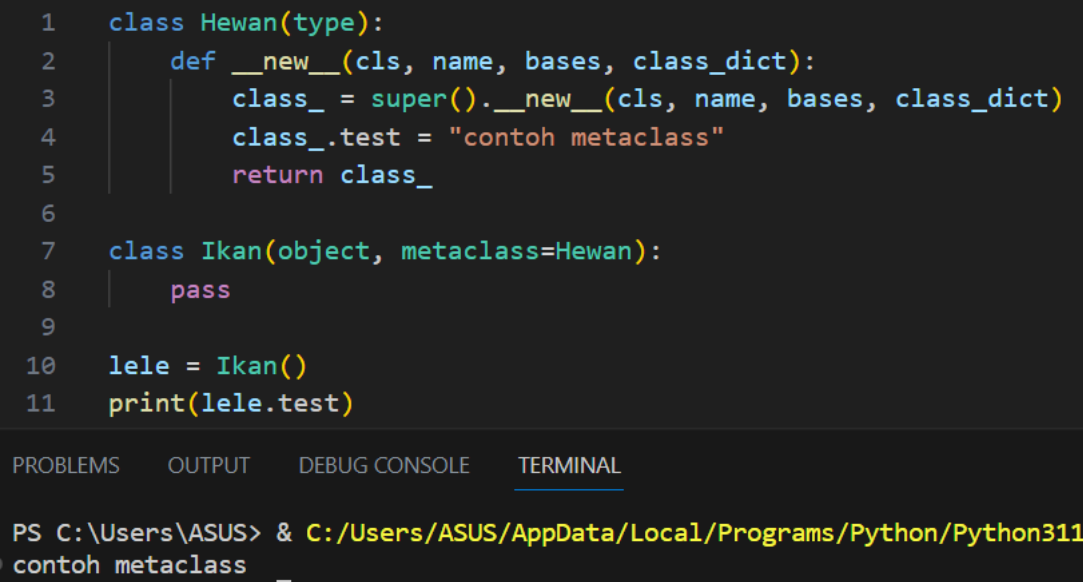
1. Definisi

2. Implementasi metaclass Menggunakan type

Gambar 8 : Implementasi metaclass menggunakan type

Pada contoh di atas, ketika variabel "hewan" dipanggil, metaclass **type** akan membuat sebuah kelas yang bernama "kucing". Kelas ini tidak memiliki parent class, karena tuple parent class dikosongkan. Atribut-atribut dari kelas tersebut diberikan dalam bentuk kamus (dictionary). Baris terakhir menampilkan alamat memori dari kelas "kucing" yang telah dibuat, yaitu 0x00000147EEF1E5D0. Kemudian jika sebuah child class dibuat yang mewarisi dari parent class, ketika variabel "spesies_hewan" dipanggil, metaclass akan membuat sebuah child class yang bernama "spesies_kucing" yang mewarisi dari parent class "hewan".

3. Implementasi metaclass Menggunakan Parameter metaclass



```
1 class Hewan(type):
2     def __new__(cls, name, bases, class_dict):
3         class_ = super().__new__(cls, name, bases, class_dict)
4         class_.test = "contoh metaclass"
5         return class_
6
7 class Ikan(object, metaclass=Hewan):
8     pass
9
10 lele = Ikan()
11 print(lele.test)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Programs/Python/Python311
contoh metaclass

Gambar 8. Implementasi metaclass dengan parameter metaclass

Pada contoh di atas mendefinisikan metaclass **Hewan** yang merupakan turunan dari **type**. Di dalam metode **__new__** pada **Hewan**, lalu menambahkan atribut **test** ke kelas yang dibuat. Selanjutnya, mendefinisikan kelas **Ikan** dengan menggunakan metaclass **Hewan** dan mewarisi dari kelas **object**. Pada saat definisi kelas, menggunakan parameter **metaclass** untuk mengindikasikan bahwa **Ikan** menggunakan metaclass **Hewan** untuk mengontrol pembuatan kelas. Objek **lele** dibuat dari kelas **Ikan**. Karena telah menambahkan atribut **test** melalui metaclass, maka saat mencetak **lele.test**, outputnya akan menjadi "contoh metaclass".

Metaclass digunakan untuk mengontrol pembuatan kelas. Ketika kelas baru dibuat, metaclass memungkinkan untuk memodifikasi kelas tersebut atau menambahkan atribut/properti khusus yang dapat diwarisi oleh instance kelas yang dibuat dari kelas tersebut. Metaclass memberikan fleksibilitas yang lebih dalam mengontrol perilaku dan atribut kelas. Namun, penggunaan metaclass perlu dipertimbangkan dengan hati-hati karena kompleksitasnya yang lebih tinggi dan dapat mempersulit pemahaman dan pemeliharaan kode.

KESIMPULAN

Dari ringkasan yang telah di buat, dapat disimpulkan bahwa Interface adalah kumpulan metode atau fungsi yang harus disediakan oleh kelas yang mengimplementasikannya. Interface digunakan ketika setiap fitur perlu diimplementasikan secara berbeda untuk setiap objek yang berbeda. Interface memberikan fleksibilitas dalam menerapkan fungsionalitas yang berbeda untuk objek yang berbeda. Kita perlu menggunakan interface ketika ingin memastikan bahwa kelas-kelas turunan mengimplementasikan metode-metode yang diperlukan oleh interface tersebut.

Kelas abstrak adalah kelas yang tidak dapat diinstansiasi secara langsung dan berfungsi sebagai kerangka dasar bagi kelas turunannya. Kelas abstrak memiliki satu atau lebih metode abstrak yang harus diimplementasikan ulang di kelas turunan. Kelas abstrak digunakan untuk membentuk struktur logika penurunan dalam pemrograman berorientasi objek. Kita perlu menggunakan kelas abstrak ketika ingin memaksa atau mengganti metode yang sama di seluruh kelas turunan dari kelas abstrak tersebut. Perbedaan antara kelas abstrak dan interface adalah bahwa kelas abstrak dapat memiliki metode abstrak maupun konkrit, sedangkan semua metode dalam sebuah interface adalah abstrak. Selain itu, kelas abstrak cenderung lebih cepat dalam kinerjanya dibandingkan dengan interface.

Kelas konkret adalah kelas yang dapat diinstansiasi langsung dan memiliki implementasi lengkap untuk semua metode yang didefinisikan dalam kelas tersebut. Kita perlu menggunakan kelas konkret ketika ingin membuat objek yang memiliki implementasi konkret dari metode-metode yang ada.

Metaclass adalah kelas yang digunakan untuk mendefinisikan perilaku kelas itu sendiri. Metaclass merupakan "kelas kelas" yang mengontrol pembuatan kelas. Metaclass dapat digunakan untuk mengubah perilaku kelas secara keseluruhan, seperti mengontrol cara instansiasi kelas, validasi atribut, dan mengubah metode-metode yang diwarisi dari kelas dasar. Penggunaan metaclass perlu dipertimbangkan dengan hati-hati karena kompleksitasnya yang lebih tinggi dan dapat mempersulit pemahaman dan pemeliharaan kode. Metaclass berbeda dengan inheritance biasa karena metaclass mengontrol pembuatan kelas sedangkan inheritance biasa mengatur pewarisan antara kelas-kelas yang sudah ada.

DAFTAR PUSTAKA

<https://www.upgrad.com/blog/how-does-abstraction-work-in-python/>

<https://www.tutorialspoint.com/abstract-base-classes-in-python-abc>

<https://www.educative.io/answers/what-are-the-formal-elements-of-the-user-interface-design>

<https://www.datacamp.com/tutorial/python-metaclasses>

<https://realpython.com/python-metaclasses/>