

**RESUME PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
Minggu 6**



ITERA

Disusun oleh :

Nama : RA Siti Zakiyah
NIM : 121140103

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI PRODUKSI DAN INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2023**

A. KELAS ABSTRAK

Kelas abstrak merupakan kelas yang berisi satu atau lebih metode abstrak. Kelas abstrak dapat dianggap sebagai cetak biru untuk kelas lain yang memungkinkan untuk membuat sekumpulan metode yang harus dibuat di dalam kelas anak mana pun yang dibangun dari kelas abstrak. Metode abstrak adalah metode yang memiliki deklarasi tetapi tidak memiliki implementasi. Kelas abstrak memiliki minimal satu atau lebih fungsi yang bersifat abstrak, namun kita juga dapat mendefinisikan fungsi biasa. Sub-kelas yang mewarisi kelas abstrak harus mengimplementasikan (override) semua fungsi abstrak yang didefinisikan kelas abstrak.

Secara default, Python tidak menyediakan kelas abstrak. Python hadir dengan modul yang menyediakan basis untuk mendefinisikan kelas Basis Abstrak (ABC) dan nama modul itu adalah ABC. ABC bekerja dengan metode dekorasi kelas dasar sebagai abstrak dan kemudian mendaftarkan kelas beton sebagai implementasi dari basis abstrak. Fungsi abstrak pada kelas abstrak ditandai dengan memberikan decorator **@abstractmethod** pada fungsi. Karena kelas abstrak tidak dapat dibuat objeknya, namun dapat mendefinisikan konstruktor yang akan dipanggil dari sub-kelas yang mewarisi kelas abstrak.

```
1  from abc import ABC, abstractmethod #(bentuk kelas abstrak)
2
3  # kelas Bentuk2D merupakan kelas abstrak dan tidak bisa diinstansiasi
4  class Bentuk2D(ABC): #(mewarisi kelas ABC)
5
6      @abstractmethod #(memberi dekorator pada fungsi)
7      def get_luas(self):
8          # fungsi ini wajib diimplementasikan pada child-class
9          pass #(Fungsi yang bersifat abstrak tidak perlu ada implementasinya di kelas Abstrak)
10
11     @abstractmethod #(memberi dekorator pada fungsi)
12     def get_keliling(self):
13         # fungsi ini wajib diimplementasikan pada child-class
14         pass #(Fungsi yang bersifat abstrak tidak perlu ada implementasinya di kelas Abstrak)
```

Gambar 1. Implementasi kelas abstrak

Dari potongan kode tersebut jika langsung dijadikan sebuah objek, maka akan menghasilkan error. Untuk mengatasi dan melakukan implementasi terhadap kelas abstrak diperlukan sebuah kelas anak atau “child” dari kelas abstraknya. Untuk membuat kelas “child” dari kelas abstrak harus membuat kembali seluruh fungsi yang ada pada kelas abstraknya (*parent class*).

```
16  class PersegiPanjang(Bentuk2D):
17      def __init__(self, warna, panjang, lebar):
18          self.w = warna
19          self.p = panjang
20          self.l = lebar
21
22      def get_luas(self):
23          return self.p * self.l
24
25      def get_keliling(self):
26          return 2 * (self.p + self.l)
27
28      def get_warna(self):
29          return self.w
```

Gambar 2. Implementasi kelas child dengan konstruktor

```

31 p = PersegiPanjang("Biru", 4, 3)
32 print(p.get_warna())
33 print(p.get_keliling())
34 print(p.get_luas())

```

Gambar 3. Implementasi kelas abstrak dengan objek

```

PS C:\Users\zakiyah\OneDrive\Dokumen\semester 4\pemrograman berorientasi objek\praktikum\minggu 6> & C:/Users/zakiyah/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/zakiyah/OneDrive/Dokumen/semester 4/pemrograman berorientasi objek/praktikum/minggu 6/resumemg6.py"
Biru
14
12
PS C:\Users\zakiyah\OneDrive\Dokumen\semester 4\pemrograman berorientasi objek\praktikum\minggu 6>

```

Gambar 4. Output yang diberikan

B. INTERFACE

Interface dalam python merupakan koleksi dari method atau fungsi-fungsi yang perlu disediakan oleh implementing class (child class). Interface biasanya digunakan untuk mendefinisikan apa saja yang bisa dilakukan sebuah kelas, tanpa memperdulikan bagaimana implementasinya. Interface mengandung metode yang bersifat abstract. Metode abstract akan memiliki satu-satunya deklarasi karena tidak adanya implementasi. dengan pengimplementasian interface kita dapat menulis kode dengan elegan dan terorganisir.

Terdapat dua jenis interface, yaitu :

1. Informal interface

Informal interface adalah kelas yang mendefinisikan metode yang dapat diganti, tetapi tidak ada penerapan yang ketat. interface ini mendefinisikan fungsi-fungsi yang bisa di implementasi tapi tanpa adanya unsur paksaan. Implementasi akan terjadi jika membuat kelas konkret yang mewarisi dari kelas “parent”. Kelas konkret merupakan subclass dari abstrak interface yang menyediakan implementasi dari method di kelas interface. gunakan inheritance untuk melakukan implementasinya.

```

1  class Buah:
2      def __init__(self, buah):
3          self.list_buah = buah
4
5      def __len__(self):
6          return len(self.list_buah)
7
8      def __constraint__(self, buah):
9          return buah in self.list_buah
10
11 class BuahBuahan(Buah):
12     pass
13
14 merah = BuahBuahan(["Apel", "Stroberi", "Cherry"])
15 #cek banyaknya buah
16 print(len(merah))
17
18 #cek apakah terdapat buah tertentu
19 print("Apel" in merah)
20 print("Pisang" in merah)
21 print("Cherry" not in merah)
22
23 #iterasi buah yang ada
24 for buah in merah:
25     print(buah)

```

Gambar 5. informal interface

Pada contoh code di atas, buah buahan Merah dapat mengecek banyaknya buah dengan (__len__) dan mengecek keberadaan buah dengan (__constraint__) pada buah

buah tersebut. Tetapi ketika dibutuhkan list dari buah yang berada pada buah buahan merah, maka akan menghasilkan error. Untuk mengatasi hal tersebut bisa dilakukan dengan mengimplementasikan sebuah fungsi baru yaitu `__iter__` sehingga memungkinkan untuk mendapatkan list buah yang ada.

```
1  class Buah:
2      def __init__(self, buah):
3          self.list_buah = buah
4
5      def __len__(self):
6          return len(self.list_buah)
7
8      def __constraint__(self, buah):
9          return buah in self.list_buah
10
11 class BuahBuahan(Buah):
12     def __iter__(self):
13         return iter(self.list_buah)
14
15 merah = BuahBuahan(["Apel", "Stroberi", "Cherry"])
16 #cek banyaknya buah
17 print(len(merah))
18
19 #cek apakah terdapat buah tertentu
20 print("Apel" in merah)
21 print("Pisang" in merah)
22 print("Cherry" not in merah)
23
24 #iterasi buah yang ada
25 for buah in merah:
26     print(buah)
```

Gambar 6. informal interface

Fungsi `__iter__` pada kelas `BuahBuahan` perlu diimplementasikan karena dibutuhkan untuk mendapatkan list buah yang ada. Tetapi tidak dipaksakan oleh kelas parent, yaitu `Buah`. Oleh sebab itu, konsep ini disebut dengan informal interface.

```
3
True
False
False
Apel
Stroberi
Cherry
PS C:\Users\zakiyah\OneDrive\Dokumen\semester 4\pemrograman berorientasi objek
\praktikum\minggu 6>
```

Gambar 7. informal interface

2. Formal Interface

Pada formal interface, kelas parent dapat dibuat hanya dengan sedikit kode, kemudian diimplementasikan pada kelas turunannya (konkret). Implementasi tersebut bersifat dipaksakan sehingga dinamakan formal interface. salah satu caranya dapat menggunakan abstract class method.

```

1  from abc import ABC
2  from abc import abstractmethod
3
4  class GPS(ABC):
5
6      @abstractmethod
7      def aktifkan_gps(self):
8          pass
9
10     @abstractmethod
11     def matikan_gps(self):
12         pass
13
14     @abstractmethod
15     def get_location(self):
16         pass

```

Gambar 7. formal interface

```

18  class Android(GPS):
19      def aktifkan_gps(self):
20          print("gps aktif")
21
22      def matikan_gps(self):
23          print("gps mati")
24
25      def get_location(self):
26          print("lokasi anda saat ini di sini")
27
28 Oppo=Android()
29 Oppo.get_location()

```

Gambar 8. contoh kelas abstract dan konkret

C. METACLASS

Metaclass adalah sebuah konsep dalam pemrograman Python yang memungkinkan Anda untuk mendefinisikan perilaku kelas. Dalam Python, semuanya adalah sebuah objek, termasuk kelas itu sendiri. Sebuah metaclass adalah sebuah kelas yang mendefinisikan cara-cara untuk membuat kelas-kelas baru.

Dalam Python, ketika Anda mendefinisikan sebuah kelas, interpreter Python membuat sebuah objek kelas untuk mewakili kelas tersebut. Objek kelas ini adalah instance dari sebuah metaclass, dan secara default metaclass yang digunakan adalah `type`. Namun, Anda dapat menentukan metaclass yang berbeda dengan mengatur atribut `__metaclass__` pada kelas Anda.

1. Membuat kelas dengan atribut dan metode yang telah ditentukan sebelumnya:

Dalam contoh ini, kita dapat menggunakan metaclass untuk membuat kelas baru yang memiliki atribut dan metode yang telah ditentukan sebelumnya. Misalnya, kita dapat membuat kelas baru yang memiliki atribut "nama" dan metode "sapa" yang akan mencetak pesan sambutan.

```

1  class Meta(type):
2      def __new__(cls, name, bases, dct):
3          dct['nama'] = 'Contoh'
4          dct['sapa'] = lambda self: print(f'Halo {self.nama}!')
5
6  class MyClass(metaclass=Meta):
7      pass
8
9  obj = MyClass()
10 obj.sapa() # Output: Halo Contoh!
11

```

2. Membuat kelas dengan atribut dan metode yang dibuat secara dinamis:

Dalam contoh ini, kita dapat menggunakan metaclass untuk membuat kelas baru dengan atribut dan metode yang dibuat secara dinamis. Misalnya, kita dapat membuat kelas

baru yang memiliki atribut "nama" dan metode "sapa" yang akan menerima parameter nama dan mencetak pesan sambutan.

```
1  class Meta(type):
2      def __new__(cls, name, bases, dct):
3          dct['sapa'] = lambda self: print(f'Halo {self.nama}!')
4
5          if 'nama' not in dct:
6              dct['nama'] = 'Contoh'
7
8          return super().__new__(cls, name, bases, dct)
9
10 class MyClass(metaclass=Meta):
11     pass
12
13 obj = MyClass()
14 obj.nama = 'John'
15 obj.sapa() # Output: Halo John!
```

3. Validasi kelas sebelum dibuat:

Dalam contoh ini, kita dapat menggunakan metaclass untuk memvalidasi kelas sebelum dibuat. Misalnya, kita dapat memastikan bahwa setiap kelas memiliki atribut "nama" sebelum dibuat.

```
1  class Meta(type):
2      def __new__(cls, name, bases, dct):
3          if 'nama' not in dct:
4              raise TypeError(f'Kelas {name} harus memiliki atribut nama!')
5
6          return super().__new__(cls, name, bases, dct)
7
8 class MyClass(metaclass=Meta):
9     nama = 'Contoh'
10
11 obj = MyClass() # Output: TypeError: Kelas MyClass harus memiliki atribut nama!
```

Metaclass merupakan salah satu fitur yang kuat di dalam pemrograman berorientasi objek yang memungkinkan kita untuk mengontrol cara objek dibuat dan diatur di dalam kelas. Namun, metaclass adalah fitur yang lebih maju dan jarang digunakan di dalam pemrograman berorientasi objek.

D. KESIMPULAN

1. Kelas abstrak dapat dianggap sebagai cetak biru untuk kelas lain yang memungkinkan untuk membuat sekumpulan metode yang harus dibuat di dalam kelas anak mana pun yang dibangun dari kelas abstrak. Sub-kelas yang mewarisi kelas abstrak harus mengimplementasikan (override) semua fungsi abstrak yang didefinisikan kelas abstrak.
2. Interface biasanya digunakan untuk mendefinisikan apa saja yang bisa dilakukan sebuah kelas, tanpa memperdulikan bagaimana implementasinya. Interface mengandung metode yang bersifat abstract. Metode abstract akan memiliki satu-satunya deklarasi karena tidak adanya implementasi.
3. Perbedaan utama antara kelas abstrak dan interface adalah bahwa kelas abstrak dapat memiliki variabel anggota dan implementasi metode, sementara interface hanya mendefinisikan metode tanpa implementasi. Sehingga semua metode yang berada pada interface harus merupakan abstract method sedangkan pada kelas abstrak boleh memiliki metode biasa.
4. Kelas konkret (concrete class) adalah sebuah kelas yang dapat diinstansiasi secara langsung dan memiliki implementasi lengkap dari semua metode yang didefinisikan di dalamnya. Kelas konkret ini biasanya digunakan untuk membuat objek-objek yang sesuai dengan konsep yang didefinisikan oleh kelas tersebut.

5. Metaclass adalah sebuah kelas yang mendefinisikan cara-cara untuk membuat kelas-kelas baru. Kita perlu menggunakan metaclass ketika kita ingin mengontrol bagaimana sebuah kelas dibuat dan berinteraksi dengan objek lainnya. Metaclass memungkinkan kita untuk menentukan atribut, metode, dan perilaku lainnya yang akan ditambahkan ke kelas saat dibuat.
6. Perbedaan antara metaclass dan inheritance biasa adalah bahwa inheritance biasa menghasilkan kelas baru dengan mewarisi sifat-sifat dan perilaku dari kelas induknya, sedangkan metaclass menghasilkan kelas baru dengan mengontrol cara pembuatan kelas dan menambahkan perilaku khusus ke kelas baru tersebut. Metaclass memungkinkan kita untuk mengontrol seluruh proses pembuatan kelas, sementara inheritance hanya memungkinkan kita untuk mewarisi sifat dan perilaku yang telah ditentukan sebelumnya.

DAFTAR PUSTAKA

<https://www.pythontutorial.net/python-oop/python-abstract-class/>

<https://www.geeksforgeeks.org/abstract-classes-in-python/>

<https://www.genius.education/blog/cara-membuat-interface-python-pysimplegui-dan-beberapa-fitur-lain>

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

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

Modul 6 Praktikum PBO ITERA

Modul 9 PBO “Kelas Abstrak dan Interface”