RESUME PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
MINGGU 6

Disusun Oleh :
Nama : Novia Eka Putri
NIM : 121140030
Kelas : RB

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI PRODUKSI DAN INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2023

A. ABSTRACT CLASSES

1. Kelas Abstrak dalam Python

Dalam pemrograman berorientasi objek, kelas abstrak merupakan kelas yang tidak dapat dibuat *instance*-nya. Namun,dapat membuat kelas yang mewarisi kelas abstraknya. Biasanya kita menggunakan kelas abstrak untuk membuat cetak biru dari kelas lain. Dengan demikian, *abstract method* adalah *method* tanpa implementasi. Sebuah kelas abstrak mungkin saja termasuk *abstract method*. Python tidak secara langsung mendukung kelas abstrak. Tapi itu menawarkan modul yang memungkinkan untuk mendefinisikan kelas abstrak. Untuk mendefinisikan sebuah kelas abstrak, dapat menggunakan modul abc (kelas dasar abstrak). Modul abc menyediakan infrastruktur untuk menentukan kelas dasar abstrak.

```
from abc import ABC

class AbstractClassName(ABC):
    pass
```

Untuk mendefinisikan abstract method digunakan decorator @abstractmethod :

```
from abc import ABC, abstractmethod


class AbstractClassName(ABC):
    @abstractmethod
    def abstract_method_name(self):
        pass
```

Contoh :

```
from abc import ABC, abstractmethod

class Shape(ABC): #kelas  Shape merupakan kelas parent
    @abstractmethod
    def area(self):
        pass

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

Untuk membuat kelas “child” dari kelas abstrak harus membuat kembali seluruh fungsi yang ada pada kelas abstraknya (parent class).

```
class Rectangle(Shape):
    def __init__(self, length, width):
```

```
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

# Mencoba membuat objek dari kelas Shape yang merupakan kelas abstrak
# Akan menghasilkan TypeError
# shape = Shape()

rectangle = Rectangle(4, 5)
print("Area of Rectangle:", rectangle.area())
print("Perimeter of Rectangle:", rectangle.perimeter())
```

Pada contoh di atas, kelas Shape merupakan kelas abstrak yang memiliki dua metode abstrak: area() dan perimeter(). Metode abstrak tidak memiliki implementasi di kelas abstrak dan harus diimplementasikan oleh kelas turunannya. Kelas Rectangle adalah turunan dari kelas Shape yang mengimplementasikan metode-metode abstrak yang didefinisikan di kelas Shape. Anda dapat melihat bahwa jika metode abstrak tidak diimplementasikan di kelas turunan, akan menghasilkan TypeError saat mencoba membuat objek dari kelas tersebut.

Output yang dihasilkan:

```
Area of Rectangle: 20
Perimeter of Rectangle: 18
```

B. INTERFACE

Pada level tinggi, *interface* bertindak sebagai *blue print* untuk mendesain kelas. Seperti kelas, *interface* menentukan *methods*. Tidak seperti kelas, metode ini abstrak. Metode abstrak adalah metode yang didefinisikan oleh *interface*. Hal tersebut tidak mengimplementasikan metode. Ini dilakukan oleh kelas-kelas yang kemudian mengimplementasikan *interface* dan memberi arti konkret pada metode abstrak *interface*.

Pendekatan Python untuk desain *interface* agak berbeda jika dibandingkan dengan bahasa seperti Java, Go, dan C++. Semua bahasa ini memiliki kata kunci *interface*, sedangkan Python tidak. Python menyimpang dari bahasa lain dalam satu aspek lainnya. Itu tidak memerlukan kelas yang mengimplementasikan antarmuka untuk mendefinisikan semua metode abstrak antarmuka.

Secara umum, Python tidak mendukung konsep antarmuka formal seperti yang ada dalam bahasa pemrograman lain seperti Java. Namun, masih dapat membuat antarmuka informal di

Python dengan menggunakan kelas abstrak Python. Berikut adalah contoh kode program untuk antarmuka informal dan formal di Python:

1. **Informal Interface**

   Sifat dinamis Python memungkinkan kita mengimplementasikan *informal interface*. *Informal interface* Python adalah kelas yang mendefinisikan metode yang dapat diganti, tetapi tidak ada penerapan yang ketat.

   Contoh implementasinya :

```
class Shape:
    def area(self):
        raise NotImplementedError("Subclasses harus mengimplementasikan method area()")

    def perimeter(self):
        raise NotImplementedError("Subclasses harus mengimplementasikan method perimeter()")

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

rectangle = Rectangle(4, 5)
print("Area of Rectangle:", rectangle.area())
print("Perimeter of Rectangle:", rectangle.perimeter())
```

   Metode area() dan perimeter() adalah metode abstrak dalam antarmuka Shape. Di kelas Shape, metode ini hanya dideklarasikan tetapi tidak diimplementasikan. Metode ini digunakan untuk menghitung luas dan keliling suatu bentuk geometri. Di kelas turunan, metode ini harus diimplementasikan sesuai dengan logika perhitungan luas dan keliling untuk bentuk geometri yang spesifik. Sedangkan kelas Rectangle adalah kelas turunan dari Shape yang mengimplementasikan antarmuka informal Shape.

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 adalah dengan menggunakan abstract class method (abc).

Contoh implementasinya :

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)


rectangle = Rectangle(4, 5)
print("Area of Rectangle:", rectangle.area())
print("Perimeter of Rectangle:", rectangle.perimeter())
```

Pada contoh ini, kelas Shape merupakan kelas abstrak yang menggunakan modul abc untuk mendefinisikan metode abstrak area() dan perimeter(). Kelas Rectangle merupakan implementasi konkret dari kelas abstrak Shape. Meskipun Python tidak memiliki konsep antarmuka formal bawaan seperti di Java, penggunaan kelas abstrak atau konvensi antarmuka informal seperti di atas dapat membantu mencapai tujuan yang sama dalam menggambarkan antarmuka suatu objek.

C. METACLASS

Metaclass adalah konsep dalam pemrograman Python yang memungkinkan kita untuk menentukan perilaku kelas. Dalam Python, kelas juga dianggap sebagai objek. Metaclass adalah kelas yang digunakan untuk membuat kelas-kelas baru. Saat kita mendefinisikan sebuah kelas dalam Python, sebuah objek kelas dibuat oleh interpreter Python untuk mewakili kelas tersebut. Objek kelas ini sebenarnya merupakan instance dari metaclass, dan secara default metaclass yang digunakan adalah tipe (type). Namun, kita juga dapat menentukan metaclass yang berbeda dengan mengatur atribut __metaclass__ pada kelas yang kita definisikan.

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

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

```
class Meta(type):
    def __new__(cls, name, bases, dct):
        dct['nama'] = 'Nobita'
        dct['karakter']     =     lambda     self:     print(f'Kartun
{self.nama}^^')

class MyClass(metaclass = Meta):
    pass

obj = MyClass()
obj.karakter() #Output Kartun Nobita^^
```

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

Dalam contoh ini, metaclass dapat digunakan untuk membuat kelas baru dengan atribut dan metode yang dibuat secara dinamis. Sebagai contoh, kita dapat membuat kelas baru yang memiliki atribut "nama" dan metode "sapa" yang menerima parameter nama dan mencetak pesan sambutan.

```
class Meta(type):
    def __new__(cls, name, bases, dct):
        dct['karakter']     =     lambda     self:     print(f'Kartun
{self.nama}^^')

        if 'nama' not in dct:
            dct['nama'] = 'Nobita'
```

```
        return super().__new__(cls, name, bases, dct)

class MyClass(metaclass = Meta):
    pass

obj = MyClass()
obj.nama = 'Sisuka'
obj.karakter() #Output Kartun Sisuka^^
```

3. **Validasi kelas sebelum dibuat:**

Dalam contoh ini, metaclass dapat digunakan untuk melakukan validasi pada kelas sebelum dibuat. Sebagai contoh, kita dapat memastikan bahwa setiap kelas memiliki atribut "nama" sebelum kelas tersebut dibuat.

```
class Meta(type):
    def __new__(cls, name, bases, dct):
        if 'nama' not in dct:
              raise TypeError(f'Kelas {name} harus memiliki atribut
nama!')

        return super().__new__(cls, name, bases, dct)

class MyClass(metaclass = Meta):
    nama = 'Nobita'

obj = MyClass() #Output Typeerror: Kelas MyClass harus memiliki
atribut nama!
```

D. KESIMPULAN

Kelas abstrak adalah kelas yang tidak dapat diinstansiasi dan berfungsi sebagai kerangka kerja untuk kelas turunan. Kelas abstrak dapat berisi metode abstrak (tanpa implementasi) dan metode konkret. Sedangkan interface adalah sebuah kontrak yang mendefinisikan perilaku yang harus diimplementasikan oleh kelas-kelas yang menggunakannya. Interface hanya berisi deklarasi metode-metode tanpa implementasi. Meskipun Python tidak memiliki konsep antarmuka formal bawaan seperti di Java, penggunaan kelas abstrak atau konvensi antarmuka informal dapat membantu mencapai tujuan yang sama dalam menggambarkan antarmuka suatu objek.

Metaclass adalah sebuah kelas yang digunakan untuk mengatur pembuatan kelas-kelas baru. Penggunaan metaclass diperlukan ketika kita ingin mengendalikan proses pembuatan kelas dan hubungannya dengan objek lain. Melalui metaclass, kita dapat menentukan atribut, metode, dan perilaku lain yang akan ditambahkan ke kelas saat kelas tersebut dibuat.

DAFTAR PUSTAKA

https://www.scaler.com/topics/interface-in-python/

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

https://realpython.com/python-interface/#python-interface-overview

https://realpython.com/python-metaclasses/#new-style-classes

https://drive.google.com/file/d/1r7y8HR_DFU7C8TNRtQVXGlC4W5xBZheq/view

https://drive.google.com/file/d/1Rz0uXfUc5CfguLP0AGpMJinjnR4NbuIu/view