TUGAS PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK

Minggu 6

Disusun Oleh :

M. Akbar Purnama Putra (121140065)

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

2023

DAFTAR ISI

BAB I

RINGKASAN

A. Kelas Abstrak

Kelas abstrak adalah sebuah kelas di dalam pemrograman yang tidak dapat di-instansiasi secara langsung, tetapi hanya dapat diwarisi oleh kelas turunannya. Kelas abstrak berisi method abstrak, yang merupakan method yang hanya dideklarasikan tanpa diberikan implementasi di dalam kelas tersebut. Kelas turunan yang mewarisi kelas abstrak harus mengimplementasikan semua method abstrak dari kelas abstrak tersebut.

Kelas abstrak sering digunakan untuk membuat kerangka kerja atau pola perilaku yang harus diikuti oleh kelas-kelas turunannya. Kelas abstrak memungkinkan pengembang untuk memperkuat desain program dengan memaksa kelas-kelas turunannya untuk mengikuti aturan-aturan tertentu, sambil memberikan fleksibilitas dalam implementasi detail kelas-kelas turunan.

Contoh kelas abstrak adalah kelas yang tidak dapat diinstansiasi secara langsung, tetapi digunakan sebagai kerangka kerja untuk kelas-kelas turunannya. Kelas abstrak ini mendefinisikan metode dan properti yang harus diimplementasikan oleh kelas-kelas turunannya.

```python
from abc import ABC, abstractmethod

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

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

Pada contoh di atas, kita menggunakan modul "abc" yang merupakan modul untuk mengimplementasikan kelas abstrak di Python. Kelas Shape adalah kelas abstrak yang memiliki dua metode abstrak yaitu "area" dan "perimeter". Metode-metode ini tidak memiliki implementasi dan hanya berfungsi sebagai kerangka kerja bagi kelas turunan.

Kelas turunan dari kelas Shape dapat diimplementasikan seperti berikut ini:

```
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)

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

    def perimeter(self):
        return 2 * 3.14 * self.radius
```

Kelas Rectangle dan Circle merupakan kelas turunan dari kelas Shape. Kedua kelas ini mengimplementasikan metode-metode "area" dan "perimeter" yang telah didefinisikan di kelas Shape. Kelas Rectangle menghitung luas dan keliling persegi panjang, sedangkan kelas Circle menghitung luas dan keliling lingkaran.

B. Interface

Python tidak memiliki konsep interface seperti yang terdapat di beberapa bahasa pemrograman seperti Java dan C#. Namun, kita dapat membuat sebuah "interface" menggunakan kelas abstrak dengan hanya mendefinisikan metode-metode tanpa implementasi. Python sendiri tidak memiliki konsep interface secara native, namun kita dapat membuat sebuah interface dengan memanfaatkan kelas abstrak dan metode abstrak. Hal ini memungkinkan kita untuk mengatur kontrak atau blueprint yang harus dipenuhi oleh kelas-kelas turunan.

Sebagai contoh, kita dapat membuat kelas abstrak "Movable" yang memiliki empat metode tanpa implementasi:

```python
from abc import ABC, abstractmethod

class Movable(ABC):
    @abstractmethod
    def move_up(self):
        pass

    @abstractmethod
    def move_down(self):
        pass

    @abstractmethod
    def move_left(self):
        pass

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

Kelas Movable adalah kelas abstrak yang memiliki empat metode abstrak yaitu "move_up", "move_down", "move_left", dan "move_right". Kelas-kelas turunan dari kelas Movable harus mengimplementasikan seluruh metode yang didefinisikan di kelas Movable.

Sebagai contoh, kita dapat membuat kelas "Car" dan "Robot" yang mengimplementasikan kelas Movable:

```python
class Car(Movable):
    def move_up(self):
        # implementasi untuk memindahkan mobil ke atas

    def move_down(self):
        # implementasi untuk memindahkan mobil ke bawah

    def move_left(self):
        # implementasi untuk memindahkan mobil ke kiri

    def move_right(self):
        # implementasi untuk memindahkan mobil ke kanan

class Robot(Movable):
    def move_up(self):
        # implementasi untuk memindahkan robot ke atas

    def move_down(self):
        # implementasi untuk memindahkan robot ke bawah

    def move_left(self):
        # implementasi untuk memindahkan robot ke kiri

    def move_right(self):
        # implementasi untuk memindahkan robot ke kanan
```

Kelas Car dan Robot merupakan kelas turunan dari kelas Movable. Kedua kelas ini mengimplementasikan seluruh metode yang didefinisikan di kelas Movable.

1) **Informal interface**

   Informal interface dalam penggunaan Python merujuk pada konvensi atau aturan yang diikuti oleh kelas atau objek untuk memfasilitasi penggunaan antarmuka pada program.

   Contoh informal interface pada Python adalah penggunaan metode __str__ pada sebuah kelas untuk mengembalikan sebuah string yang mewakili objek tersebut. Metode ini umum digunakan untuk mencetak atau menampilkan objek dalam bentuk yang dapat dibaca manusia.

   Berikut ini adalah contoh penggunaan metode __str__ pada kelas Person:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}"
```

   Dengan menggunakan metode __str__ pada kelas Person, kita dapat mencetak objek Person dalam bentuk yang dapat dibaca manusia:

```
person = Person("John", 30)
print(person)  # Output: Name: John, Age: 30
```

   Penggunaan metode __str__ pada kelas Person adalah contoh dari informal interface pada Python, karena kita tidak perlu mengimplementasikan sebuah antarmuka resmi untuk menghasilkan sebuah representasi yang mudah dibaca dari objek Person.

2) **Formal Interface**

   Formal interface pada penggunaan Python merujuk pada sebuah kontrak antara kelas atau objek dengan kode pengguna yang menggunakan kelas atau objek tersebut. Kontrak ini menjelaskan metode-metode yang harus tersedia pada kelas atau objek, jenis argumen yang diterima dan nilai kembali yang diharapkan. Formal interface pada Python sering diimplementasikan sebagai abstraksi kelas atau sebagai definisi kelas abstrak.

   Contoh formal interface pada Python adalah penggunaan modul abc (Abstract Base Classes) untuk mendefinisikan sebuah kelas abstrak dan mengimplementasikan sebuah antarmuka formal yang harus diikuti oleh kelas-kelas yang mengimplementasikan kelas abstrak tersebut. Berikut ini adalah contoh penggunaan modul abc untuk

mendefinisikan sebuah kelas abstrak Shape dan mengimplementasikan sebuah antarmuka formal:

```
from abc import ABC, abstractmethod

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

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

Dalam contoh di atas, kelas Shape merupakan kelas abstrak yang memiliki dua metode abstrak, area dan perimeter. Kelas-kelas yang mengimplementasikan kelas Shape harus mengimplementasikan kedua metode tersebut. Dengan menggunakan modul abc dan definisi kelas abstrak, kita dapat memastikan bahwa kelas-kelas tersebut akan memiliki metode-metode yang diperlukan dan sesuai dengan antarmuka formal yang telah didefinisikan.

C. Metaclass

Dalam pemrograman berorientasi objek, metaclass adalah sebuah kelas yang digunakan untuk membuat kelas baru. Metaclass memungkinkan penggunaan sintaksis Python untuk membuat sebuah kelas menjadi lebih fleksibel dan dapat disesuaikan dengan kebutuhan pengguna.

Dalam Python, setiap kelas sebenarnya adalah sebuah objek dari sebuah kelas yang lebih tinggi yang disebut sebagai metaclass. Secara default, metaclass yang digunakan oleh Python adalah type, namun kita juga dapat membuat metaclass kustom kita sendiri dengan menggunakan pewarisan dari kelas type.

Berikut adalah contoh sederhana penggunaan metaclass dalam Python:

```
class MyMeta(type):
    def __new__(cls, name, bases, attrs):
        print(f"Creating class {name}")
        return super().__new__(cls, name, bases, attrs)

class MyClass(metaclass=MyMeta):
    pass
```

Pada contoh di atas, kita mendefinisikan sebuah metaclass MyMeta yang menerima parameter name, bases, dan attrs. Parameter tersebut merepresentasikan nama kelas,

kelas-kelas induk, dan atribut-atribut kelas. Metaclass tersebut juga mengimplementasikan sebuah metode __new__() yang akan dipanggil ketika sebuah kelas baru dibuat.

Kemudian, kita mendefinisikan sebuah kelas MyClass yang menggunakan metaclass MyMeta. Ketika kode di atas dijalankan, maka akan muncul pesan "Creating class MyClass" di konsol, menandakan bahwa metaclass MyMeta berhasil dipanggil saat kelas MyClass dibuat.

Dalam penggunaan sebenarnya, metaclass dapat digunakan untuk melakukan manipulasi pada kelas-kelas yang dibuat. Misalnya, kita dapat menambahkan atribut atau metode ke kelas baru, atau mengubah perilaku dari metode yang ada di kelas.

Method __new__ menerima 4 parameter, yaitu:

- cls: Class yang menjadi metaclass, yaitu MyMeta
- name: Nama dari class yang menggunakan MyMeta sebagai metaclass, yaitu MyClass
- bases: Tuple yang berisi base class dari class yang menggunakan MyMeta sebagai metaclass, yaitu object
- attrs: Dictionary yang berisi semua atribut dan method yang didefinisikan pada class, yaitu method hello
- Pada method __new__, kita melakukan looping terhadap semua atribut dan method yang didefinisikan pada class yang menggunakan MyMeta sebagai metaclass. Jika sebuah atribut atau method bersifat callable (dapat dipanggil), maka kita mengubah namanya menjadi uppercase dan menyimpannya pada new_attrs. Jika tidak, kita menyimpannya tanpa perubahan.

Penggunaan metaclass pada Python memungkinkan kita untuk memodifikasi perilaku dari class dan objek yang dibuat dari class tersebut secara otomatis, tanpa perlu mengubah kode di setiap class atau objek tersebut. Hal ini sangat berguna dalam pengembangan program yang besar dan kompleks.

BAB II
KESIMPULAN

Interface adalah sebuah kontrak yang menentukan perilaku atau fungsi-fungsi yang harus dimiliki oleh kelas yang mengimplementasikannya. Interface digunakan untuk memastikan bahwa kelas-kelas yang berbeda dapat saling berkomunikasi dan bekerja sama dengan benar.

Kelas abstrak adalah sebuah kelas yang tidak dapat diinstansiasi dan hanya berfungsi sebagai kerangka dasar untuk kelas turunannya. Kelas abstrak digunakan ketika kita ingin memaksa kelas-kelas turunannya untuk mengimplementasikan beberapa fungsi atau perilaku tertentu. Perbedaan utama antara kelas abstrak dan interface adalah bahwa kelas abstrak dapat memiliki implementasi dari beberapa fungsi, sedangkan interface tidak memiliki implementasi sama sekali.

Kelas konkret adalah kelas yang dapat diinstansiasi dan memiliki implementasi dari semua fungsi yang didefinisikan dalam kelas tersebut. Kelas konkret digunakan ketika kita ingin membuat sebuah objek atau instance dari kelas tersebut.

Metaclass adalah sebuah kelas yang digunakan untuk membuat kelas lainnya. Metaclass dapat mengontrol perilaku kelas-kelas yang dihasilkan. Metaclass biasanya digunakan ketika kita ingin membuat sebuah framework atau library yang memerlukan pembuatan kelas-kelas dinamis. Perbedaan utama antara metaclass dan inheritance biasa adalah bahwa inheritance biasa mengontrol perilaku instance, sedangkan metaclass mengontrol perilaku kelas.

Kita perlu memahami konsep-konsep ini untuk membangun struktur program yang baik dan mudah dipelihara. Penggunaan interface, kelas abstrak, kelas konkret, dan metaclass dapat membantu kita membuat program yang modular, fleksibel, dan mudah dipahami.

DAFTAR PUSTAKA

"abc — Abstract Base Classes — Python 3.11.3 documentation." *Python Docs*, https://docs.python.org/3/library/abc.html. Accessed 11 April 2023.

"Home." *Abstrak Baseclass*, https://docs.python.org/3/library/abc.html . Accessed 11 April 2023.

"Home." *Interface*, https://docs.python.org/3/library/abc.html. Accessed 11 April 2023.

Sturtz, John. "Python Metaclasses – Real Python." *Real Python*, https://realpython.com/python-metaclasses/. Accessed 11 April 2023.

"3. Data model — Python 3.11.3 documentation." *Python Docs*, https://docs.python.org/3/reference/datamodel.html#object.__str__. Accessed 11 April 2023.