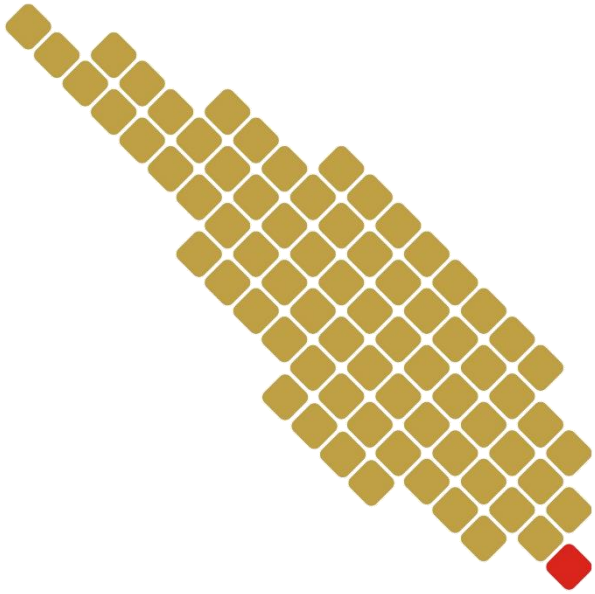
TUGAS MODUL 6
PEMROGRAMAN BERBASIS OBJEK
RB



ITERA

Disusun Oleh :
Rafif Aditya – 121140090

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA
2023

DAFTAR ISI

KELAS ABSTRAK

Dalam Python, kelas abstrak adalah kelas yang tidak dapat diinstansiasi secara langsung dan berfungsi sebagai kerangka kerja untuk kelas-kelas turunannya. Kelas abstrak menyediakan metode dan properti yang harus ada dalam kelas turunan, tetapi tidak memberikan implementasi yang sebenarnya. Kelas abstrak berperan dalam pemodelan konsep umum atau abstrak yang digunakan oleh kelas-kelas turunan.

Untuk membuat kelas abstrak di Python, kita perlu menggunakan modul `abc` (Abstract Base Classes). Modul ini menyediakan kelas dasar bernama `ABC` (Abstract Base Class) yang dapat diwarisi oleh kelas abstrak kita.

Berikut adalah contoh sederhana yang menjelaskan penggunaan kelas abstrak dalam Python:

```
1. from abc import ABC, abstractmethod
2.
3. class Bentuk(ABC):
4.     @abstractmethod
5.     def luas(self):
6.         pass
7.
8.     @abstractmethod
9.     def keliling(self):
10.        pass
11.
12.class Persegi(Bentuk):
13.    def __init__(self, sisi):
14.        self.sisi = sisi
15.
16.    def luas(self):
17.        return self.sisi ** 2
18.
19.    def keliling(self):
20.        return 4 * self.sisi
21.
22.class Lingkaran(Bentuk):
23.    def __init__(self, radius):
24.        self.radius = radius
25.
26.    def luas(self):
27.        return 3.14 * self.radius ** 2
28.
29.    def keliling(self):
```

```
        return 2 * 3.14 * self.radius

persegi = Persegi(5)
print(persegi.luas())       # Output: 25
print(persegi.keliling())   # Output: 20

lingkaran = Lingkaran(3)
print(lingkaran.luas())       # Output: 28.26
print(lingkaran.keliling())   # Output: 18.84
```

Pada contoh di atas, kita mendefinisikan kelas `Bentuk` sebagai kelas abstrak dengan menggunakan modul `abc`. Kedua metode `luas()` dan `keliling()` dalam kelas `Bentuk` dideklarasikan sebagai metode abstrak dengan menggunakan dekorator `@abstractmethod`. Kelas `Persegi` dan `Lingkaran` mewarisi dari kelas `Bentuk` dan harus memberikan implementasi untuk kedua metode abstrak tersebut.

Dalam contoh tersebut, kita dapat melihat bahwa kelas abstrak `Bentuk` tidak dapat diinstansiasi langsung, tetapi kelas-kelas turunan seperti `Persegi` dan `Lingkaran` dapat diinstansiasi dan menggunakan metode yang telah didefinisikan dalam kelas abstrak. Dengan menggunakan kelas abstrak, kita dapat memastikan bahwa semua kelas turunan memiliki metode dan properti yang diharapkan, sehingga memudahkan pemodelan konsep yang lebih umum dan terstruktur.

INTERFACE

Dalam konteks pemrograman, sebuah interface adalah kontrak atau spesifikasi yang mendefinisikan metode atau perilaku yang harus diimplementasikan oleh kelas-kelas yang mengadopsinya. Interface menyediakan semacam panduan atau blueprint untuk kelas-kelas tersebut.

Pada Python, meskipun tidak ada tipe data khusus yang disebut "interface", kita dapat menggunakan kelas abstrak dan metode abstrak untuk menciptakan interface. Konsep interface diimplementasikan menggunakan modul `abc` (Abstract Base Classes) yang disediakan oleh Python.

Berikut adalah contoh sederhana yang menjelaskan penggunaan interface dalam Python:

```
1. from abc import ABC, abstractmethod
2.
3. class Shape(ABC):
4.     @abstractmethod
5.     def area(self):
6.         pass
7.
8.     @abstractmethod
9.     def perimeter(self):
10.        pass
11.
12.class Rectangle(Shape):
13.    def __init__(self, width, height):
14.        self.width = width
15.        self.height = height
16.
17.    def area(self):
18.        return self.width * self.height
19.
20.    def perimeter(self):
21.        return 2 * (self.width + self.height)
22.
23.class Circle(Shape):
24.    def __init__(self, radius):
25.        self.radius = radius
26.
27.    def area(self):
28.        return 3.14 * self.radius ** 2
29.
30.    def perimeter(self):
31.        return 2 * 3.14 * self.radius
```

```
32.
33.rectangle = Rectangle(5, 3)
34.print(rectangle.area())       # Output: 15
35.print(rectangle.perimeter()) # Output: 16
36.
37.circle = Circle(3)
38.print(circle.area())          # Output: 28.26
39.print(circle.perimeter())     # Output: 18.84
```

Pada contoh di atas, kita mendefinisikan sebuah interface bernama `Shape` menggunakan kelas abstrak. Interface ini memiliki dua metode abstrak, yaitu `area()` dan `perimeter()`, yang harus diimplementasikan oleh kelas-kelas yang mengadopsinya.

Kelas `Rectangle` dan `Circle` merupakan implementasi dari interface `Shape`. Kedua kelas ini mengimplementasikan metode-metode yang didefinisikan dalam interface, yaitu `area()` dan `perimeter()`.

Dalam contoh tersebut, kita dapat melihat bahwa kelas-kelas yang mengadopsi interface `Shape` harus memberikan implementasi untuk semua metode yang dideklarasikan dalam interface tersebut. Dengan menggunakan interface, kita dapat memastikan bahwa kelas-kelas tersebut memiliki perilaku yang sesuai dengan spesifikasi yang ditetapkan oleh interface tersebut.

Penting untuk dicatat bahwa dalam Python, implementasi interface bersifat opsional. Oleh karena itu, penggunaan interface lebih pada konvensi dan dokumentasi untuk menjelaskan kontrak yang harus dipenuhi oleh kelas-kelas yang mengimplementasikannya.

METACLASS

Dalam Python, metaclass adalah sebuah kelas yang digunakan untuk membuat dan mengendalikan perilaku kelas-kelas lain sebagai objek. Metaclass memberikan cara untuk mengubah atau memanipulasi cara pembuatan kelas atau perilaku kelas pada waktu eksekusi.

Secara umum, di Python, kelas adalah objek. Ketika kita mendefinisikan kelas, sebenarnya kita membuat objek baru yang merupakan instansi dari metaclass tertentu secara default (biasanya metaclass bawaan `type`). Metaclass mengatur cara pembuatan kelas ini dan memiliki pengaruh langsung terhadap perilaku kelas.

Dalam prakteknya, penggunaan metaclass sering kali kompleks dan jarang digunakan dalam skenario pemrograman sehari-hari. Namun, ada beberapa kasus penggunaan yang berguna untuk metaclass, seperti:

1. Validasi dan penyaringan: Metaclass dapat digunakan untuk melakukan validasi dan penyaringan terhadap definisi kelas. Misalnya, kita dapat menggunakan metaclass untuk memastikan bahwa kelas yang dibuat harus memiliki metode tertentu atau properti yang diperlukan.

2. Pemantauan dan logging: Metaclass dapat digunakan untuk memantau pembuatan dan penggunaan kelas-kelas dalam program. Dengan metaclass, kita dapat mencatat log atau melakukan tindakan lain saat kelas-kelas dibuat atau digunakan.

3. Implementasi pola desain: Metaclass dapat digunakan untuk menerapkan pola desain tertentu secara otomatis. Misalnya, metaclass dapat digunakan untuk menerapkan Singleton atau Factory Pattern pada kelas-kelas yang dihasilkan.

Berikut adalah contoh sederhana yang menggambarkan penggunaan metaclass dalam Python:

```
1. class SingletonMeta(type):
2.     _instances = {}
3.
4.     def __call__(cls, *args, **kwargs):
5.         if cls not in cls._instances:
6.             cls._instances[cls] = super().__call__(*args, **kwargs)
7.         return cls._instances[cls]
8.
9. class SingletonClass(metaclass=SingletonMeta):
10.     def __init__(self, name):
11.         self.name = name
```

```
12.
13.     def say_hello(self):
14.         print(f"Hello, {self.name}!")
15.
16. obj1 = SingletonClass("Object 1")
17. obj2 = SingletonClass("Object 2")
18.
19. print(obj1 is obj2)  # Output: True
20. obj1.say_hello()     # Output: Hello, Object 1!
21. obj2.say_hello()     # Output: Hello, Object 1!
```

Dalam contoh di atas, kita menggunakan metaclass `SingletonMeta` untuk mengimplementasikan pola Singleton pada kelas `SingletonClass`. Dengan metaclass ini, hanya satu instansi dari kelas `SingletonClass` yang akan dibuat selama runtime program. Ketika objek `obj1` dan `obj2` dibuat, keduanya secara efektif merujuk pada instansi yang sama.

Penting untuk dicatat bahwa penggunaan metaclass harus dilakukan dengan hati-hati, karena dapat mempersulit pemahaman dan pemeliharaan kode. Penggunaan metaclass lebih umum dalam lingkungan pengembangan kerangka kerja atau saat kita perlu melakukan operasi tingkat lanjut pada definisi kelas.

KESIMPULAN

Berdasarkan penjelasan di atas, kita dapat mengambil kesimpulan sebagai berikut:

1. Kelas Abstrak pada Python:
   - Kelas abstrak adalah kelas yang tidak dapat diinstansiasi secara langsung dan berfungsi sebagai kerangka kerja untuk kelas-kelas turunannya.
   - Kelas abstrak mengandung metode dan properti yang harus ada dalam kelas turunan, tetapi tidak memberikan implementasi yang sebenarnya.
   - Kelas abstrak digunakan untuk pemodelan konsep umum atau abstrak yang digunakan oleh kelas-kelas turunannya.
   - Dalam Python, kelas abstrak diimplementasikan menggunakan modul `abc` dan kelas dasar `ABC`.

2. Interface pada Python:
   - Interface adalah kontrak atau spesifikasi yang mendefinisikan metode atau perilaku yang harus diimplementasikan oleh kelas-kelas yang mengadopsinya.
   - Interface dapat digunakan untuk memastikan bahwa kelas-kelas tersebut memiliki perilaku yang sesuai dengan spesifikasi yang ditetapkan.
   - Meskipun tidak ada tipe data khusus yang disebut "interface" dalam Python, kita dapat menggunakan kelas abstrak dan metode abstrak untuk menciptakan interface.
   - Implementasi interface dalam Python lebih pada konvensi dan dokumentasi untuk menjelaskan kontrak yang harus dipenuhi oleh kelas-kelas yang mengimplementasikannya.

3. Metaclass pada Python:
   - Metaclass adalah kelas yang digunakan untuk membuat dan mengendalikan perilaku kelas-kelas lain sebagai objek.
   - Metaclass mengatur cara pembuatan kelas dan memiliki pengaruh langsung terhadap perilaku kelas.
   - Metaclass digunakan dalam kasus-kasus khusus, seperti validasi dan penyaringan, pemantauan dan logging, atau implementasi pola desain tertentu.
   - Dalam Python, kelas adalah objek, dan ketika kita mendefinisikan kelas, kita sebenarnya membuat objek baru yang merupakan instansi dari metaclass.
   - Penggunaan metaclass perlu dilakukan dengan hati-hati karena kompleksitasnya dan lebih umum dalam lingkungan pengembangan kerangka kerja atau operasi tingkat lanjut pada definisi kelas.

Ketiga materi di atas memberikan alat dan konsep yang berbeda untuk mengatur dan mengendalikan perilaku kelas-kelas dalam Python. Kelas abstrak digunakan untuk pemodelan konsep umum, interface digunakan untuk mendefinisikan kontrak perilaku, dan metaclass digunakan untuk mengubah cara pembuatan kelas atau mengendalikan

perilaku kelas. Dalam pengembangan perangkat lunak, pemahaman dan penggunaan yang tepat dari ketiga konsep ini dapat membantu dalam membangun kode yang terstruktur, terorganisir, dan mudah dipelihara.