TUGAS PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK

Minggu 6

Disusun Oleh :

Bagas Satrio 121140077

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK ELEKTRO, INFORMATIKA DAN SISTEM FISIS

INSTITUT TEKNOLOGI SUMATERA

2023

DAFTAR ISI

BAB I 3
A. Abstract Class 3
B. Interface 5
C. Metaclass 9
BAB II 11
KESIMPULAN 11
BAB III 13
DAFTAR PUSTAKA 13

BAB I
RESUME

A. Abstract Class

Abstract class atau kelas abstrak adalah kelas yang terletak di posisi tertinggi dalam hierarki class. Class ini tidak dapat diinstansiasi karena masih bersifat abstrak. Class ini hanya berisi variabel umum dan deklarasi method tanpa detail penggunaannya (abstract method). Selanjutnya class yang menjadi turunan dari abstract class ini yang akan menentukan detail penggunaan method tersebut. Kelas turunan dari abstract class wajib mengimplementasikan semua metode abstrak yang ada di dalam abstract class tersebut, sehingga membuat abstract class menjadi sebuah kerangka kerja atau blueprint bagi kelas-kelas turunannya. Abstract class sering digunakan dalam pemrograman berorientasi objek untuk membuat sebuah model atau template yang dapat digunakan oleh banyak kelas turunannya. Kelas abstrak dapat dianggap sebagai templat untuk kelas lain. Ini memungkinkan untuk menentukan kumpulan metode yang harus diimplementasikan di kelas anak mana pun yang berasal dari kelas abstrak. Kelas abstrak adalah kelas yang memiliki satu atau lebih metode abstrak. Metode yang memiliki deklarasi tetapi tidak ada implementasi disebut abstrak. Kami menggunakan kelas abstrak untuk membangun unit fungsional yang besar. Kelas abstrak digunakan ketika kami ingin menyediakan interface umum untuk beragam implementasi komponen.

Abstract class dapat membangun Application Program Interface (API) umum untuk sekelompok subkelas dengan mendefinisikan kelas dasar abstrak. Fungsionalitas ini sangat berguna saat pihak ketiga akan menawarkan implementasi, seperti dengan plugin, tetapi juga dapat membantu saat bekerja dalam tim besar atau dengan basis kode yang luas saat mengingat semua kelas sulit atau tidak mungkin. Python tidak menyertakan kelas abstrak secara default. ABC adalah nama modul Python yang menjadi dasar untuk membangun Abstract Base Classes (ABC). dapat membangun Application Program Interface (API) umum untuk sekelompok subkelas dengan mendefinisikan kelas dasar abstrak. Python tidak menyertakan kelas abstrak secara default. ABC adalah nama modul Python yang menjadi dasar untuk membangun Abstract Base Classes (ABC). Abstract Base Classes ABC bekerja dengan mengabstraksi metode kelas dasar dan kemudian mendaftarkan kelas konkret sebagai implementasi dari basis abstrak. Ketika sebuah metode dihiasi dengan istilah @abstractmethod, itu menjadi abstrak.

Contoh implementasi kelas abstrak pada pyhton :

Python memiliki modul untuk menggunakan Abstract Base Classes (ABC). Fungsi abstrak pada kelas abstrak ditandai dengan memberikan decorator @abstractmethod pada fungsi yang dibuat.

```
1. from abc import ABC, abstractmethod
2.
3. class Mahasiswa(ABC):
4.     def __init__(self, nama, nim, jurusan):
5.         self.nama = nama
6.         self.nim = nim
7.         self.jurusan = jurusan
8.
```

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

class MahasiswaAktif(Mahasiswa):
    def __init__(self, nama, nim, jurusan, semester):
        super().__init__(nama, nim, jurusan)
        self.semester = semester

    def tampil_data(self):
        print(f"Mahasiswa Aktif")
        print(f"Nama: {self.nama}")
        print(f"NIM: {self.nim}")
        print(f"Jurusan: {self.jurusan}")
        print(f"Semester: {self.semester}")
        print()

class MahasiswaLulus(Mahasiswa):
    def __init__(self, nama, nim, jurusan, tahun_lulus):
        super().__init__(nama, nim, jurusan)
        self.tahun_lulus = tahun_lulus

    def tampil_data(self):
        print(f"Mahasiswa Sudah Lulus")
        print(f"Nama: {self.nama}")
        print(f"NIM: {self.nim}")
        print(f"Jurusan: {self.jurusan}")
        print(f"Tahun Lulus: {self.tahun_lulus}")

#Contoh Penggunaan
mahasiswa_aktif = MahasiswaAktif("Bagas Satrio", "121140077", "Teknik
Informatika", 4)
mahasiswa_aktif.tampil_data()

mahasiswa_lulus = MahasiswaLulus("Jokowi", "119920098", "Teknik
Informatika", 2023)
mahasiswa_lulus.tampil_data()
```

Pada contoh di atas terdapat kelas abstrak “Mahasiswa” yang memiliki satu method abstract yaitu **“tampil_data()”**. Kemudian terdapat dua kelas turunan dari kelas **“Mahasiswa”**, yaitu **“MahasiswaAktif”** dan **“MahasiswaLulus”** yang masing-masing mengimplementasi method **“tampil_data()”** sesuai dengan kebutuhan masing-masing kelas. Kelas **“MahasiswaAktif”** memiliki atribut tambahan yaitu semester, sementara kelas **“MahasiswaLulus”** memiliki atribut tambahan yaitu tahun_lulus. Setiap kelas turunan juga memiliki konstruktor yang memanggil konstruktor kelas induk menggunakan super().__init__(). Terakhir, kita menginstansiasi objek dari masing-masing kelas turunan dan memanggil method **“tampil_data()”** untuk menampilkan biodata mahasiswa sesuai dengan kelas yang bersangkutan. Jika dijalankan akan mendapatkan output seperti berikut:

```
Mahasiswa Aktif
Nama: Bagas Satrio
NIM: 121140077
Jurusan: Teknik Informatika
Semester: 4

Mahasiswa Sudah Lulus
Nama: Jokowi
NIM: 119920098
Jurusan: Teknik Informatika
Tahun Lulus: 2023
```

B. Interface

Interface pada python biasanya digunakan untuk mendefinisikan kontrak dan mendefinisikan apa saja yang bisa dilakukan sebuah kelas, tanpa memperdulikan bagaimana implementasinya. Interface berfungsi sebagai template untuk desain kelas. Interface, seperti kelas, menentukan metode. Metode ini, tidak seperti kelas, bersifat abstrak. Metode abstrak adalah metode yang hanya ditentukan oleh interface. Itu tidak menerapkan metode. Kelas mencapai ini dengan mengimplementasikan interface dan memberikan makna nyata pada fungsi abstrak interface. Pendekatan Python untuk desain interface berbeda dari bahasa seperti Java, Go, dan C++. Semua bahasa ini mengandung kata kunci interface, meskipun Python tidak. Python menyimpang dari bahasa lain dengan satu cara lagi. Kelas yang mengimplementasikan interface tidak perlu mendefinisikan semua metode abstrak interface.
Perbedaan Antara Interface dengan Abstract Class
Berikut adalah perbedaan antara Abstract Class dengan Interface:

a. Interface
   - Karena interface publik secara default, itu tidak mendukung access specifier
   - Hanya dapat berisi tanda tangan dan tidak dapat berisi penerapannya
   - Kecepatan proses relative lebih lambat
   - Tidak ada field
   - Interface hanya dapat digunakan untuk memperluas metode abstrak.

b. Kelas Abstrak
   - Abstrak dapat memiliki access specifier
   - Pelaksanaannya dapat dilakukan secara bertahap
   - Kecepatan proses relative lebih cepat
   - Dapat memiliki field dan konstanta
   - Dapat menampung method non-abstract

Contoh implementasi interface pada pyhton :

```python
from abc import ABC, abstractmethod

class AlatMusik(ABC):
    @abstractmethod
    def tampil_jenis(self):
        pass

class Gitar(AlatMusik):
```

```
9.      def tampil_jenis(self):
10.         print("Ini adalah Gitar")
11.
12. class Piano(AlatMusik):
13.     def tampil_jenis(self):
14.         print("Ini adalah Piano")
15.
16. data1 = Gitar()
17. data2 = Piano()
18.
19. data1.tampil_jenis()
20. data2.tampil_jenis()
```

Pada contoh di atas, kita membuat sebuah interface dengan menggunakan modul abc dari Python. Interface ini memiliki satu method abstract yaitu **'tampil_jenis'**. Method ini tidak memiliki implementasi, sehingga kelas yang mengimplementasikan interface ini harus mengimplementasikan method **"tampil_jenis"** sendiri. Kemudian membuat dua kelas yaitu **"Gitar"** dan **"Piano"** yang mengimplementasikan interface **"Alat Musik"**. Kedua kelas ini harus mengimplementasikan method **"tampil_jenis"** yang didefinisikan dalam interface **"Alat Musik"**. Dalam implementasi ini, kita membuat sebuah formal interface yaitu kelas **"Alat Musik"** yang mengandung method abstract **"tampil_jenis"**. Kedua kelas **"Gitar"** dan **"Piano"** kemudian mengimplementasikan interface ini dengan mengimplementasikan method **"tampil_jenis"** sendiri-sendiri. Jika dijalankan akan mendapatkan output seperti berikut:

```
Ini adalah Gitar
Ini adalah Piano
```

1. Informal Interface

   Dalam beberapa kasus, batasan ketat dari interface Python formal mungkin tidak diperlukan. Karena Python bersifat dinamis, sehingga membuat interface informal. interfal informal Python adalah kelas yang mendefinisikan metode yang dapat diganti tanpa penerapan yang ketat. Informal interface dalam pemrograman adalah suatu pola atau konvensi yang digunakan oleh programer dalam mengimplementasikan fungsionalitas yang serupa di dalam kelas-kelas yang berbeda tanpa menggunakan fitur formal interface pada bahasa pemrograman yang digunakan.

   Dalam informal interface, para programer sepakat untuk menggunakan method-method yang memiliki nama dan tipe parameter yang sama pada kelas-kelas yang berbeda sebagai sebuah interface. Dengan kata lain, method yang memiliki nama dan tipe parameter yang sama pada kelas-kelas yang berbeda dianggap sebagai sebuah interface secara informal. Sebagai contoh, dalam bahasa pemrograman Python, kita tidak memiliki fitur formal interface seperti pada bahasa Java. Namun, kita dapat menggunakan informal interface dengan menggunakan method-method yang memiliki nama dan tipe parameter yang sama pada kelas-kelas yang berbeda sebagai sebuah interface secara informal.

   Penggunaan informal interface biasanya mempermudah programer dalam membuat kelas-kelas yang berhubungan dan dapat saling berinteraksi. Namun, penggunaan informal interface juga dapat menimbulkan kebingungan jika tidak ada dokumentasi atau

konvensi yang jelas dalam penggunaannya. Contoh implementasi Informal Interface pada python :

```
class AlatMusik:
    def __init__(self, jenis, produksi):
        self.jenis = jenis
        self.produksi = produksi

    def tampilkan_info(self):
        print(f"Jenis Alat Musik : {self.jenis}")
        print(f"Negara Produksi : {self.produksi}")

class Gitar:
    def __init__(self, merk, tahun):
        self.merk = merk
        self.tahun = tahun

    def tampilkan_info(self):
        print(f"Merk Gitar : {self.merk}")
        print(f"Tahun Produksi Gitar : {self.tahun}")
```

Pada contoh di atas, terdapat dua kelas **“AlatMusik”** dan **“Gitar”** yang memiliki method tampilkan_info(). Meskipun kelas-kelas tersebut tidak memiliki hubungan turunan atau implementasi formal interface, kita dapat menganggap method tampilkan_info() sebagai sebuah informal interface karena nama method tersebut konsisten pada kelas-kelas yang berbeda. Kemudian, kita dapat membuat kelas **“Pemilik”** yang ingin mengakses method tampilkan_info() pada kelas **“AlatMusik”** dan **“Gitar”**. Kita dapat memanggil method tampilkan_info() pada kelas-kelas tersebut dengan cara yang sama, seperti pada contoh berikut:

```
class Pemilik:
    def __init__(self, AlatMusik, Gitar):
        self.AlatMusik = AlatMusik
        self.Gitar = Gitar

    def tampilkan_info(self):
        print("Info Alat Musik")
        self.AlatMusik.tampilkan_info()
        print()
        print("Info Tentang Gitar")
        self.Gitar.tampilkan_info()

data1 = AlatMusik("Gitar", "USA")
data2 = Gitar("Vender", 1998)
pemilik = Pemilik(data1, data2)
pemilik.tampilkan_info()
```

Dalam contoh di atas, terdapat kelas **“Pemilik”** yang memiliki dua atribut **“AlatMusik”** dan **“Gitar”** yang berisi objek-objek dari kelas **“AlatMusik”** dan **“Gitar”**. Kemudian, kita membuat method tampilkan info() pada kelas **“Pemilik”** yang memanggil method tampilkan_info() pada objek-objek **“AlatMusik”** dan **“Gitar”**. Dengan menggunakan informal interface, kita dapat memastikan bahwa kelas **“Pemilik”**

dapat mengakses method tampilkan_info() pada kelas **“AlatMusik”** dan **“Gitar”** tanpa harus mengimplementasikan formal interface atau turunan dari kelas-kelas tersebut. Jika dijalankan akan mendapatkan output sebagai berikut :

```
Info Alat Musik
Jenis Alat Musik : Gitar
Negara Produksi : USA

Info Tentang Gitar
Merk Gitar : Vender
Tahun Produksi Gitar : 1998
```

2. Formal Interface

Formal interface dalam bahasa pemrograman Python adalah suatu konstruksi bahasa yang memungkinkan programer untuk menentukan suatu set method yang harus diimplementasikan pada kelas-kelas yang menggunakan interface tersebut. Formal interface digunakan untuk memastikan bahwa suatu kelas yang mengimplementasikan interface tersebut memiliki method-method yang diperlukan untuk berinteraksi dengan kelas-kelas lain. Dalam Python, formal interface dapat diimplementasikan dengan menggunakan modul abc (Abstract Base Class). Modul ini memungkinkan programer untuk membuat kelas-kelas abstrak yang dapat digunakan sebagai formal interface. Sebuah kelas abstrak adalah kelas yang memiliki satu atau lebih method abstract, yaitu method yang didefinisikan tetapi tidak diimplementasikan pada kelas tersebut.

Formal interface adalah salah satu yang ditegakkan secara eksplisit. Kita harus menggunakan ABC untuk membuat antarmuka formal. ABC (Abstract Base Class) mengajarkan untuk mendefinisikan kelas abstrak dan mendefinisikan metode pada kelas dasar sebagai metode abstrak sehingga objek apa pun yang diturunkan dari kelas dasar harus mengimplementasikan metode tersebut. Untuk mengimplementasikan sebuah formal interface, sebuah kelas harus mengextend kelas abstrak dan mengimplementasikan method-method abstract yang didefinisikan pada kelas abstrak tersebut. Jika sebuah kelas tidak mengimplementasikan seluruh method yang diperlukan pada kelas abstrak, maka kelas tersebut tidak dapat diinstantiasi. Contoh implementasi informal interface pada pyhton :

```
1. from abc import ABC, abstractmethod
2.
3. class BangunDatar(ABC):
4.     @abstractmethod
5.     def luas(self):
6.         pass
7.
8. class PersegiPanjang(BangunDatar):
9.     def __init__(self, panjang, lebar):
10.        self.panjang = panjang
11.        self.lebar = lebar
12.
13.    def luas(self):
14.        return self.panjang * self.lebar
15.
16.class Lingkaran(BangunDatar):
17.    def __init__(self, r):
```

```
        self.r = r

    def luas(self):
        return 3.14 * self.r ** 2

data1 = PersegiPanjang(5, 10)
print("Luas persegi panjang :", data1.luas())

data2 = Lingkaran(5)
print("Luas lingkaran :", data2.luas())
```

Pada contoh di atas, terdapat sebuah kelas **"BangunDatar"** yang merupakan kelas abstrak dan memiliki satu metode abstrak yaitu **"luas"**. Kelas Rectangle dan Circle merupakan kelas konkret yang mengimplementasikan metode **"luas"** dari kelas **"BangunDatar"**. Dengan menggunakan modul abc, kita dapat memastikan bahwa setiap kelas yang mengimplementasikan **"BangunDatar"** harus mengimplementasikan metode **"luas"** tersebut. Jika dijalankan akan mendapatkan output sebagai berikut :

```
Luas persegi panjang : 50
Luas lingkaran : 78.5
```

C. Metaclass

Pada Python, Metaclass adalah kelas yang menjelaskan bagaimana kelas bertindak. Kelas adalah turunan dari metaclass. Dalam Python, kelas menjelaskan bagaimana instance kelas akan bertindak. Untuk memahami metaclass dengan benar, diperlukan pengalaman sebelumnya dalam berurusan dengan kelas Python. Metaclass adalah kelas yang digunakan untuk membuat kelas baru. Metaclass dapat mengontrol pembuatan kelas, seperti mengubah atribut atau metode dari kelas, mengubah cara kelas diwariskan, dan sebagainya.

Metaclass biasanya digunakan dalam situasi di mana kita ingin membuat kelas dengan perilaku khusus, atau untuk melakukan validasi dan verifikasi pada kelas sebelum kelas tersebut dibuat. Misalnya, di beberapa framework web Python, metaclass digunakan untuk memastikan bahwa setiap kelas model database memiliki atribut dan metode yang benar. Dalam Python, setiap kelas memiliki sebuah metaclass yang secara default adalah kelas type. Namun, kita dapat membuat metaclass kustom dengan membuat sebuah kelas yang mewarisi dari kelas type, dan kemudian menggunakan metaclass tersebut saat membuat kelas baru.

```
class Meta(type):
    def __new__(cls, name, bases, attrs):
        # Tambahkan atribut 'created_by' ke dalam kelas
        attrs['created_by'] = 'Bagas'
        return super().__new__(cls, name, bases, attrs)

class Class(metaclass=Meta):
    x = 121140077
    def foo(self):
        pass

print(Class.x)
print(Class().created_by)
```

Pada contoh di atas, terdapat sebuah metaclass bernama **“Meta”**. Ketika sebuah kelas baru dibuat dengan menggunakan metaclass **“Meta”**, metode __new__ pada metaclass tersebut akan dipanggil. Di dalam metode __new__, kita menambahkan **atribut created_by** ke dalam kelas yang dibuat, kemudian memanggil metode __new__ dari kelas dasar (super().__new__) untuk membuat kelas tersebut. Kemudian, kita membuat kelas **“Class”** dengan menggunakan metaclass **“Meta”**. Ketika kelas **“Class”** dibuat, atribut x akan ditambahkan ke dalam kelas tersebut, dan juga atribut created_by yang sebelumnya telah ditambahkan oleh metaclass **“Meta”**. Hasilnya, kita dapat mengakses atribut x dari kelas **“Class”** dan atribut created_by dari objek yang dibuat dari kelas **“Class”**. Jika dijalankan akan mendapatkan output sebagai berikut :

```
121140077
Bagas
```

BAB II

KESIMPULAN

Interface pada python biasanya digunakan untuk mendefinisikan kontrak dan mendefinisikan apa saja yang bisa dilakukan sebuah kelas, tanpa memperdulikan bagaimana implementasinya. Interface berfungsi sebagai template untuk desain kelas. Interface, seperti kelas, menentukan metode. Metode ini, tidak seperti kelas, bersifat abstrak. Metode abstrak adalah metode yang hanya ditentukan oleh interface. Itu tidak menerapkan metode. Kelas mencapai ini dengan mengimplementasikan interface dan memberikan makna nyata pada fungsi abstrak interface. Interface dapat digunakan dalam beberapa situasi seperti ketika ingin membuat sebuah kelas atau objek yang dapat digunakan oleh banyak kelas atau objek lainnya, membuat sebuah plugin atau modul yang dapat diintegrasikan dengan sistem atau aplikasi lainnya dan ketika ingin membuat sebuah aplikasi yang mudah diuji (testable).

Abstract class atau kelas abstrak adalah kelas yang terletak di posisi tertinggi dalam hierarki class. Class ini tidak dapat diinstansiasi karena masih bersifat abstrak. Class ini hanya berisi variabel umum dan deklarasi method tanpa detail penggunaannya (abstract method). Selanjutnya class yang menjadi turunan dari abstract class ini yang akan menentukan detail penggunaan method tersebut. Kelas turunan dari abstract class wajib mengimplementasikan semua metode abstrak yang ada di dalam abstract class tersebut, sehingga membuat abstract class menjadi sebuah kerangka kerja atau blueprint bagi kelas-kelas turunannya. Kelas abstrak pada Python digunakan ketika ingin membuat sebuah abstraksi yang mendefinisikan perilaku umum atau fungsionalitas dari kelas, tanpa memberikan implementasi konkret dari kelas tersebut. Dalam hal ini, hanya menentukan metode atau atribut yang perlu ada pada kelas, tanpa harus menentukan bagaimana metode atau atribut tersebut harus diimplementasikan. Perbedaan antara Abstract Class dengan Interface:

a. Interface
   - Karena interface publik secara default, itu tidak mendukung access specifier
   - Hanya dapat berisi tanda tangan dan tidak dapat berisi penerapannya
   - Kecepatan proses relative lebih lambat
   - Tidak ada field
   - Interface hanya dapat digunakan untuk memperluas metode abstrak.
b. Kelas Abstrak
   - Abstrak dapat memiliki access specifier
   - Pelaksanaannya dapat dilakukan secara bertahap
   - Kecepatan proses relative lebih cepat
   - Dapat memiliki field dan konstanta
   - Dapat menampung method non-abstract

Kelas konkret dalam pemrograman adalah kelas yang memiliki implementasi lengkap untuk semua metode dan propertinya, sehingga dapat langsung digunakan untuk membuat objek. Kelas konkret dibedakan dari kelas abstrak, yang hanya memiliki definisi metode dan properti tetapi tidak memiliki implementasi yang lengkap. Kelas abstrak sering digunakan sebagai kerangka dasar untuk kelas konkret yang mengimplementasikan metode dan properti yang diwarisi dari kelas abstrak. Kelas konkret dapat ketika ingin membuat objek yang spesifik dengan implementasi yang sudah

lengkap. Misalnya, jika kita ingin membuat sebuah objek alat musik, kita dapat membuat kelas konkret "AlatMusik" dengan implementasi untuk semua properti dan metodenya seperti jenis, merk, dan tahun produksinya. Setelah itu, dapat membuat objek mobil dengan menggunakan kelas konkret ini.

Pada Python, Metaclass adalah kelas yang menjelaskan bagaimana kelas bertindak. Kelas adalah turunan dari metaclass. Dalam Python, kelas menjelaskan bagaimana instance kelas akan bertindak. Untuk memahami metaclass dengan benar, diperlukan pengalaman sebelumnya dalam berurusan dengan kelas Python. Metaclass adalah kelas yang digunakan untuk membuat kelas baru. Metaclass dapat mengontrol pembuatan kelas, seperti mengubah atribut atau metode dari kelas, mengubah cara kelas diwariskan, dan sebagainya. Metaclass biasanya digunakan dalam situasi di mana ingin membuat kelas dengan perilaku khusus, atau untuk melakukan validasi dan verifikasi pada kelas sebelum kelas tersebut dibuat. Misalnya, di beberapa framework web Python, metaclass digunakan untuk memastikan bahwa setiap kelas model database memiliki atribut dan metode yang benar.

BAB III

DAFTAR PUSTAKA

bestharadhakrishna. (2021, Maret 19). *Abstract Classes in Python*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/abstract-classes-in-python/

Kumar, B. (2020, Desember 24). *Introduction to Python Interface*. Retrieved from PythonGuides.com: https://pythonguides.com/python-interface/

Muhardian, A. (2019, Desember 28). *Tutorial Java OOP: Memahami Interface di Java (dan Contohnya)*. Retrieved from Petani Kode: https://www.petanikode.com/java-oop-interface/

Murphy, W. (n.d.). *Implementing an Interface in Python*. Retrieved from Real Python: https://realpython.com/python-interface/

Mwiti, D. (2018, Desember). *Introduction to Python Metaclasses*. Retrieved from DataCamp: https://www.datacamp.com/tutorial/python-metaclasses

Sturtz, J. (n.d.). *Python Metaclasses*. Retrieved from Real Python: https://realpython.com/python-metaclasses/#author

Surya, A. A. (2019, Desember 9). *Interface dan Abstract Class, Apa perbedaannya?* Retrieved from IDCSHARP: https://idcsharp.com/2019/12/09/interface-dan-abstract-class-apa-perbedaannya/