TUGAS PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK

Minggu 6

Disusun Oleh :
Aginda Dufira (121140058)

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA
2023

DAFTAR ISI

BAB I

RINGKASAN

A. Kelas Abstrak

Dalam bahasa pemrograman, kelas abstrak adalah kelas generik (atau jenis objek) yang digunakan sebagai dasar untuk membuat objek tertentu yang sesuai dengan protokolnya, atau rangkaian operasi yang didukungnya. Kelas abstrak, dalam konteks Java, adalah kelas super yang tidak dapat dibuat instance-nya dan digunakan untuk menyatakan atau mendefinisikan karakteristik umum. Objek tidak dapat dibentuk dari kelas abstrak Java; mencoba membuat instance kelas abstrak hanya menghasilkan kesalahan kompiler. Kelas abstrak dideklarasikan menggunakan kata kunci abstract. Subclass yang diperluas dari kelas abstrak memiliki semua atribut kelas abstrak, selain atribut khusus untuk setiap subclass. Kelas abstrak menyatakan karakteristik kelas dan metode untuk implementasi, sehingga mendefinisikan keseluruhan antarmuka. Kelas abstrak berguna saat membuat hierarki kelas yang memodelkan realitas karena memungkinkan untuk menentukan tingkat fungsionalitas invarian dalam beberapa metode, tetapi meninggalkan implementasi metode lain hingga implementasi khusus dari kelas tersebut (kelas turunan) diperlukan. Kelas abstrak berfungsi sebagai templat untuk subkelasnya. Misalnya, kelas abstrak Tree dan subclass, Banyan_Tree, memiliki semua karakteristik pohon serta karakteristik yang spesifik untuk pohon beringin.

Pada contoh kode di atas, terdapat sebuah class abstract bernama Shape yang memiliki sebuah method abstract bernama area(). Kemudian, terdapat dua class turunan dari Shape, yaitu Square dan Circle. Kedua class ini harus mengimplementasikan method area() karena mereka merupakan turunan dari class abstract Shape. Kemudian, pada bagian bawah kode, terdapat pembuatan instance objek dari class Square dan Circle serta pemanggilan method area() pada kedua objek tersebut. (“Python-interface module”)

```python
from abc import ABC, abstractmethod

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

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2

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

    def area(self):
        return 3.14 * (self.radius ** 2)

# Membuat instance objek Square dan Circle
square = Square(4)
circle = Circle(3)

# Memanggil method area()
print("Luas Square:", square.area())
print("Luas Circle:", circle.area())
```

```
Luas Square: 16
Luas Circle: 28.26
```

B. Interface

Dalam bahasa berorientasi objek seperti Python, interface adalah kumpulan tanda tangan metode yang harus disediakan oleh kelas pelaksana. Menerapkan interface adalah cara menulis kode terorganisir dan mencapai abstraksi. Paket zope.interface menyediakan implementasi "antarmuka objek" untuk Python. Itu dikelola oleh proyek Zope Toolkit. Paket mengekspor dua objek, 'interface' dan 'Atribut' secara langsung. Itu juga mengekspor beberapa metode pembantu. Ini bertujuan untuk memberikan semantik yang lebih ketat dan pesan kesalahan yang lebih baik daripada modul abc bawaan Python.

interface tidak didukung secara native oleh Python, meskipun kelas abstrak dan metode abstrak dapat digunakan untuk menyiasatinya. Pada perspektif yang lebih tinggi, interface berfungsi sebagai template untuk desain kelas. interface membuat metode dengan cara yang sama seperti kelas, tetapi tidak seperti kelas, metode ini abstrak.

Dalam python, interface didefinisikan menggunakan pernyataan kelas python dan merupakan subkelas dari interface. interface yang merupakan interface induk untuk semua interface.

```
Syntax :
class IMyInterface(zope.interface.Interface):
    # methods and attributes
```

```python
from abc import ABC, abstractmethod

class Pastry(ABC):
    @abstractmethod
    def get_name(self):
        pass

    @abstractmethod
    def get_price(self):
        pass

class Cake(Pastry):
    def __init__(self, name, price):
        self._name = name
        self._price = price

    def get_name(self):
        return self._name

    def get_price(self):
        return self._price

class Cookie(Pastry):
    def __init__(self, name, price):
        self._name = name
        self._price = price

    def get_name(self):
        return self._name

    def get_price(self):
        return self._price
```

```python

class PastryShop:
    def __init__(self, pastries):
        self._pastries = pastries

    def get_pastries(self):
        return self._pastries

    def get_total_price(self):
        total_price = 0
        for pastry in self._pastries:
            total_price += pastry.get_price()
        return total_price

# Membuat instance objek Cake dan Cookie
chocolate_cake = Cake("Chocolate Cake", 25000)
vanilla_cookie = Cookie("Vanilla Cookie", 5000)

# Membuat instance objek PastryShop
pastry_shop = PastryShop([chocolate_cake, vanilla_cookie])

# Memanggil method get_pastries() dan get_total_price()
pastries = pastry_shop.get_pastries()
print("Daftar pastri yang tersedia di toko:")
for pastry in pastries:
    print(f"- {pastry.get_name()} ({pastry.get_price()})")

total_price = pastry_shop.get_total_price()
print(f"Total harga pastri: {total_price}")

```

```
Daftar pastri yang tersedia di toko:
- Chocolate Cake (25000)
- Vanilla Cookie (5000)
Total harga pastri: 30000
```

Pada contoh kode di atas, terdapat sebuah "interface" bernama Pastry yang memiliki dua method abstract, yaitu get_name() dan get_price(). Kemudian, terdapat dua class turunan dari Pastry, yaitu Cake dan Cookie. Kedua class ini mengimplementasikan method get_name() dan get_price() karena mereka merupakan turunan dari "interface" Pastry.

Kemudian, terdapat sebuah class PastryShop yang memiliki sebuah method get_pastries() untuk mendapatkan daftar pastri yang tersedia di toko dan sebuah method get_total_price() untuk mendapatkan total harga dari semua pastri yang ada di toko. Class PastryShop menerima sebuah list objek Pastry pada konstruktor.

Pada bagian bawah kode, terdapat pembuatan instance objek Cake dan Cookie, serta pembuatan instance objek PastryShop dengan argumen list yang berisi instance objek Cake dan Cookie. Kemudian, terdapat pemanggilan method get_pastries() untuk mendapatkan daftar pastri yang tersedia di toko dan method get_total_price() untuk mendapatkan total harga dari semua pastri yang ada di toko.

1) **Informal interface**

Dalam keadaan tertentu, Anda mungkin tidak memerlukan aturan ketat dari antarmuka Python formal. Sifat dinamis Python memungkinkan Anda mengimplementasikan antarmuka informal. Antarmuka informal Python adalah kelas yang mendefinisikan metode yang dapat diganti, tetapi tidak ada penegakan yang ketat.Antarmuka informal dapat berguna untuk proyek dengan basis kode kecil dan sejumlah pemrogram. Namun, antarmuka informal akan menjadi pendekatan yang salah untuk aplikasi yang lebih besar.

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def move(self):
        raise NotImplementedError()

class Cat(Animal):
    def __init__(self, name):
        super().__init__(name)

    def move(self):
        print(f'{self.name} walks like a cat')

class Bird(Animal):
    def __init__(self, name):
        super().__init__(name)

    def move(self):
        print(f'{self.name} flies like a bird')

def animal_move(animal):
    animal.move()

cat = Cat('Kitty')
bird = Bird('Tweety')

animal_move(cat)
animal_move(bird)
```

```
Kitty walks like a cat
Tweety flies like a bird
```

Pada contoh di atas, terdapat sebuah class Animal yang memiliki method move() yang belum diimplementasikan. Class Cat dan Bird merupakan subclass dari Animal dan mengimplementasikan method move() dengan perilaku yang berbeda. Kemudian terdapat sebuah fungsi animal_move() yang menerima sebuah objek Animal sebagai parameter dan memanggil method move() pada objek tersebut. Fungsi ini dapat menerima objek dari class apa pun yang merupakan subclass dari Animal.

Dalam penggunaannya, kita tidak perlu mendefinisikan sebuah Formal Interface untuk Animal, Cat, atau Bird. Penggunaannya mengandalkan kesepakatan bahwa semua subclass dari Animal harus mengimplementasikan method move(). Hal ini dapat dianggap sebagai sebuah Informal Interface.

Contoh di atas menunjukkan fleksibilitas dan kemudahan penggunaan Informal Interface pada Python. Namun, hal ini juga menimbulkan risiko bahwa sebuah subclass dari Animal dapat tidak mengimplementasikan method move(), yang dapat menyebabkan error saat program dijalankan. Oleh karena itu, dalam penggunaannya, kita perlu memastikan bahwa kesepakatan dalam penggunaan method-method telah didefinisikan dengan jelas dan dipahami oleh semua pengguna program.

2) Formal Interface

Antarmuka formal adalah antarmuka yang diberlakukan secara formal. Dalam beberapa situasi, pengetikan protokol atau bebek menciptakan kebingungan, seperti pertimbangkan contoh kita memiliki dua kelas FourWheelVehicle dan TwoWheelVehicle keduanya memiliki metode SpeedUp ( ), sehingga objek dari kedua kelas dapat dipercepat, tetapi kedua objek tersebut tidak sama meskipun keduanya kelas mengimplementasikan antarmuka yang sama. Jadi untuk mengatasi kebingungan ini, kita bisa menggunakan antarmuka formal. Untuk membuat interface formal, kita perlu menggunakan ABC (Abstract Base Classes).

ABC sederhana karena antarmuka atau kelas dasar didefinisikan sebagai kelas abstrak di alam dan kelas abstrak berisi beberapa metode sebagai abstrak. Selanjutnya, jika ada kelas atau objek yang mengimplementasikan atau menggerakkan dari kelas dasar ini, maka kelas dasar ini dipaksa untuk mengimplementasikan semua metode tersebut. Perhatikan bahwa antarmuka tidak dapat dibuat instance-nya, yang berarti bahwa kita tidak dapat membuat objek antarmuka. Jadi kami menggunakan kelas dasar untuk membuat objek, dan kami dapat mengatakan bahwa objek tersebut mengimplementasikan antarmuka. Dan kami akan menggunakan fungsi tipe untuk mengonfirmasi bahwa objek mengimplementasikan antarmuka tertentu atau tidak.

```python
from abc import ABC, abstractmethod

class Hewan(ABC):
    @abstractmethod
    def suara(self):
        pass

    @abstractmethod
    def jenis_makanan(self):
        pass

class Kucing(Hewan):
    def suara(self):
        return "Meong"

    def jenis_makanan(self):
        return "Daging"

class Anjing(Hewan):
    def suara(self):
        return "Guk Guk"

    def jenis_makanan(self):
        return "Tulang"

kucing = Kucing()
print(kucing.suara())
print(kucing.jenis_makanan())

anjing = Anjing()
print(anjing.suara())
print(anjing.jenis_makanan())
```

```
Meong
Daging
Guk Guk
Tulang
```

Pada contoh di atas, terdapat sebuah abstract class Hewan yang mendefinisikan sebuah Formal Interface untuk sebuah hewan. Interface ini memiliki dua method, yaitu suara() dan jenis_makanan() yang harus diimplementasikan oleh semua subclass dari Hewan.

Kemudian terdapat dua subclass dari Hewan, yaitu Kucing dan Anjing yang mengimplementasikan method suara() dan jenis_makanan() dengan perilaku yang berbeda untuk masing-masing hewan.

Dalam penggunaannya, kita perlu mengimplementasikan seluruh method yang terdapat pada Formal Interface Hewan pada setiap subclass yang menggunakan Formal Interface tersebut. Dalam contoh di atas, kita harus mengimplementasikan method suara() dan jenis_makanan() pada setiap subclass dari Hewan.

Penggunaan Formal Interface pada Python menggunakan ABC memberikan jaminan bahwa sebuah class atau objek akan mengimplementasikan seluruh method yang terdapat pada sebuah Formal Interface. Jika sebuah subclass tidak mengimplementasikan seluruh method tersebut, maka program tidak akan dapat dijalankan dan akan menimbulkan error. Hal ini memudahkan dalam pemeliharaan dan pengembangan program.

C. Metaclass

Istilah metaprogramming mengacu pada potensi suatu program untuk memiliki pengetahuan atau memanipulasi dirinya sendiri. Python mendukung bentuk metaprogramming untuk kelas yang disebut metaclasses. Metaclass adalah konsep OOP esoteris, bersembunyi di balik hampir semua kode Python. Anda menggunakannya apakah Anda menyadarinya atau tidak. Sebagian besar, Anda tidak perlu menyadarinya. Kebanyakan programmer Python jarang, jika pernah, harus memikirkan tentang metaclass.

Namun, ketika diperlukan, Python menyediakan kemampuan yang tidak didukung oleh semua bahasa berorientasi objek: Anda dapat membukanya dan menentukan metaclass khusus. Penggunaan metaclass kustom agak kontroversial, seperti yang disarankan oleh kutipan berikut dari Tim Peters, guru Python yang menulis Zen of Python:

“Metaclass adalah keajaiban yang lebih dalam dari yang seharusnya dikhawatirkan oleh 99% pengguna. Jika Anda bertanya-tanya apakah Anda membutuhkannya, Anda tidak (orang-orang yang benar-benar membutuhkannya tahu dengan pasti bahwa mereka membutuhkannya, dan tidak memerlukan penjelasan mengapa).

Dalam Python, semuanya memiliki beberapa tipe yang terkait dengannya. Misalnya, jika kita memiliki variabel yang memiliki nilai integer maka tipenya adalah int. Anda bisa mendapatkan tipe apapun menggunakan fungsi type().

```python
class MyMeta(type):
    def __new__(cls, name, bases, attrs):
        new_attrs = {}
        for attr_name, attr_value in attrs.items():
            if callable(attr_value):
                new_attrs[attr_name.upper()] = attr_value
            else:
                new_attrs[attr_name] = attr_value

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

class MyClass(metaclass=MyMeta):
    def hello(self):
        print("Hello, gesss")

my_object = MyClass()
my_object.HELLO()

```

```
Hello, gesss
```

Pada contoh di atas, kita mendefinisikan sebuah metaclass MyMeta dengan menggunakan type sebagai parent class. MyMeta memiliki method __new__ yang akan dipanggil ketika sebuah class yang menggunakan MyMeta sebagai metaclass dibuat.

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.

Setelah itu, kita memanggil method __new__ dari parent class menggunakan super().__new__ untuk membuat class baru dengan atribut dan method yang telah dimodifikasi. Terakhir, kita mendefinisikan sebuah class MyClass yang menggunakan MyMeta sebagai metaclass. Class ini memiliki method hello.

Kita membuat sebuah objek my_object dari class MyClass, dan memanggil method HELLO() (bukan hello()) pada objek tersebut. Hal ini memanggil method hello() pada class MyClass karena method ini telah diubah namanya menjadi uppercase oleh metaclass MyMeta.

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

Dalam bahasa berorientasi objek seperti Python, interface adalah kumpulan tanda tangan metode yang harus disediakan oleh kelas pelaksana. Menerapkan interface adalah cara menulis kode terorganisir dan mencapai abstraksi. Dalam interface, kita hanya menentukan nama dan tipe parameter dari setiap method tanpa memberikan implementasi. Interface digunakan untuk menentukan suatu standard yang harus dipenuhi oleh class-class yang akan diimplementasikan. Kita perlu menggunakan interface ketika kita ingin membuat sebuah standard atau kontrak yang harus dipenuhi oleh class-class yang akan diimplementasikan. Dengan menggunakan interface, kita dapat memastikan bahwa class-class tersebut akan memiliki method-method yang diperlukan dengan nama dan tipe parameter yang sama, sehingga memudahkan dalam integrasi antara class-class yang berbeda. Selain itu, penggunaan interface juga dapat mempermudah dalam melakukan testing, debugging, dan maintenance pada kode kita.

kelas abstrak adalah kelas generik (atau jenis objek) yang digunakan sebagai dasar untuk membuat objek tertentu yang sesuai dengan protokolnya, atau rangkaian operasi yang didukungnya. Dalam kelas abstrak, kita dapat mendefinisikan method yang harus ada pada subclass-nya, namun kita tidak memberikan implementasi pada method tersebut. Kita perlu menggunakan kelas abstrak ketika kita ingin membuat beberapa class dengan behaviour yang sama namun dengan implementasi yang berbeda. Dengan menggunakan kelas abstrak, kita dapat mengurangi duplikasi kode dan memastikan bahwa subclass-subclass yang dibuat akan mengikuti kerangka kerja yang telah ditentukan. Selain itu, kelas abstrak juga dapat mempermudah dalam melakukan testing, debugging, dan maintenance pada kode kita. Perbedaan utama antara kelas abstrak dan interface adalah pada kelas abstrak kita dapat memberikan implementasi dari beberapa method yang kita definisikan, sedangkan pada interface kita hanya menentukan nama dan tipe parameter dari setiap method tanpa memberikan implementasi. Selain itu, kelas abstrak dapat memiliki implementasi pada beberapa method, sedangkan interface tidak memiliki implementasi pada seluruh method yang didefinisikan.

Kelas konkret adalah sebuah kelas yang dapat di-instantiate atau dibuat menjadi objek. Kelas ini memiliki implementasi pada setiap method yang didefinisikan. Kelas konkret dapat ketika ingin membuat objek yang spesifik dengan implementasi yang sudah lengkap. Kelas konkret juga dapat memiliki beberapa method atau atribut tambahan yang tidak terdapat pada class abstrak atau interface, sehingga memungkinkan kita untuk membuat objek yang lebih kompleks dan sesuai dengan kebutuhan dari program kita. Selain itu, penggunaan kelas tertentu juga dapat mempermudah dalam melakukan testing, debugging, dan maintenance pada kode kita.

Metaclass adalah sebuah kelas yang digunakan untuk membuat kelas baru. Dalam Python, setiap kelas pada dasarnya adalah sebuah objek dari sebuah metaclass. Metaclass dapat digunakan untuk mengontrol cara pembuatan kelas, termasuk membuat atribut, method, dan behavior dari kelas. Kita perlu menggunakan metaclass ketika kita ingin membuat sebuah kelas yang memiliki behavior atau struktur yang berbeda dari kelas pada umumnya. Contohnya ketika kita ingin membuat kelas dengan behavior yang berbeda dari kelas konkret atau kelas abstrak, atau ketika kita ingin membatasi cara subclassing dari kelas kita. Perbedaan utama antara metaclass dan

inheritance biasa adalah pada penggunaan metaclass kita dapat mengontrol pembuatan class, sedangkan pada inheritence biasa kita hanya dapat menurunkan sifat-sifat class dari superclass ke subclass-nya. Metaclass juga memungkinkan kita untuk membuat konstruktor, method, atau atribut pada class dengan cara yang lebih dinamis dan fleksibel. Selain itu, penggunaan metaclass dapat mempermudah dalam melakukan pengaturan pada struktur dan perilaku dari kelas pada program kita.

DAFTAR PUSTAKA

"Abstract Classes in Python." *GeeksforGeeks*, 19 March 2021, https://www.geeksforgeeks.org/abstract-classes-in-python/ Accessed 11 April 2023.

Banu, Afshan. "Interface in Python | How to Create Interface in Python with Examples." *eduCBA*, https://www.educba.com/interface-in-python/. Accessed 11 April 2023.

"Metaprogramming with Metaclasses in Python." *GeeksforGeeks*, 11 October 2021, https://www.geeksforgeeks.org/metaprogramming-metaclasses-python/. Accessed 11 April 2023.

Murphy, William. "Implementing an Interface in Python – Real Python." *Real Python*, https://realpython.com/python-interface/. Accessed 11 April 2023.

"Python-interface module." *GeeksforGeeks*, 26 March 2020, https://www.geeksforgeeks.org/python-interface-module/. Accessed 11 April 2023.

Rouse, Margaret. "What is an Abstract Class? - Definition from Techopedia." *Techopedia*, 6 June 2022, https://www.techopedia.com/definition/17408/abstract-class. Accessed 10 April 2023.

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