RESUME PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK

Oleh :

Giovanni Lucy Faustine S (121140060)

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA
LAMPUNG SELATAN
2023

1. **Abstraksi**

Abstraksi digunakan untuk menyembunyikan fungsi internal dari user. User hanya dapat mengakses implementasi dasar dari program, namun operasi di dalam program akan tersembunyi. Sederhananya, user familiar dengan apa yang dilakukan program tanpa tahu bagaimana program itu berjalan. Konsep ini banyak diterapkan di smartphone, seperti fitur camera, voice reocrder, dan lain-lain. Contoh lain adalah remote TV. Ketika user menekan tombol volume up, user tidak mengetahui bagaimana proses yang dijalankan di dalamnya.

Konsep abstraksi ini bertujuan untuk mengurangi kompleksitas dalam OOP. Dalam phyton, abstraksi dapat dilakukan dengan menggunakan abstract class dan interface. Sebuah class bisa terdiri dari parent class dan child class. Abstrack method tidak mengandung implementasi dan dapat di inherited oleh subclass. Abstraksi menggunakan module Abstract Base Classes (ABC). Dan fungsi kelas abstrak ditandai dengan memberi decorator @abstractmethod pada fungsi yang dibuat.

```
# abstraction in python
from abc import ABC,abstractmethod
#abstract class
class Subject(ABC):
    @abstractmethod
    def subject(self):
        Pass
```

Pada potongan program di atas, ABC di import dan abstractmethod dari modul abc. Setelah itu, class Subject yang dibuat akan mewarisi class ABC dari modul abc. Lalu di atas method terdapat @abstractmethod untuk mendefinisikan method tersebut sebagai abstract method. Kode di atas akan error apabila kelas tersebut dijadikan sebuah objek.

```
import abc
from abc import ABC, abstractmethod
class Subject(ABC):
    # an abstract method
    @abstractmethod
    def subject(self):
        pass
class Kimia(Subject):
    def subject(self):
        print("Kelas Kimia")
class Fisika(Subject):
    def subject(self):
        print("Kelas Fisika")
# create an object of an abstract class
obj=Subject()
```

Output dari kode di atas:

```
Output:-


Traceback (most recent call last):


  File "main.py", line 87, in <module>


    obj=Subject()


TypeError: Can't instantiate abstract class Subject with abstract method
subject
```

Untuk mengatasi error di atas, maka class child harus dibuat dari kelas abstrak tersebut. Implementasi abstrak method pada subclass akan memanggil method secara langsung. Ketika membuat class child dari class abstract, maka seluruh method dan fungsi di class paremt harus dibuat kembali. Contohnya sebagai berikut.

```
import abc
from abc import ABC, abstractmethod
class Subject(ABC):
    def __init__(self, subject):
        self.subject = subject
    # an abstract method
    @abc.abstractproperty
    def subjectx(self):
        pass
class Kimia(Subject):
    def __init__(self):
        super().__init__("Kimia")

    @property
    def subjectx(self):
        return self.subject
class Fisika(Subject):
    def __init__(self):
        super().__init__("Fisika")

    @property
    def subjectx(self):
        return self.subject

obj=Kimia()
print("Kelasmu "+obj.subject)
```

```
Output:-


Kelasmu Kimia
```

Pada program di atas, abstract class Subject dibuat dan method subjectx menggunakan decorator @abstractmethod yang juga di beri konstruktor untuk menginisiasi subject variable. Class child terdiri dari Kimia dan Fisika, yang merupakan turunan dari class parent. Method subjectx diimplementasikan di setiap class child untuk assign semua mata kuliah dari variable subject. Di atas semua child class, diberikan keyword @property. Kemudian membuat objek dari class child sehingga konstruktor pertama akan dipanggil dan dijadikan value dari subject.

2. Interface

Intrerface berperan penting dalam software engineering. Konsep interface adalah merepresentasikan bagaimana sebuah objek bekerja. Sebuah interface mendeskripsikan apa yang sebuah item bisa lakukan dalam melakukan tugas dalam sistem. Interface adalah koleksi dari method atau functions yang perlu disediakan oleh child class. Method pada interface bersifat abstract dan methode abstract tidak memiliki implementasi dan akan meiliki satu-satunya deklarasi.

Pada phyton, tidak terdapat eksplisit keyword untuk membuat interface. Di phyton, pengimplementasian interface adalah sebuah cara untuk menulis yang elegan dan terorganisir.

Perbedaan interface dan abstract class

Phyton interface	Phyton abstract class
Semua metode dari sebuah (formal) interface adalah abstrak	Sebuah abstract class dapat mempunyai metode abstract begitu juga dengan metode konkrit
Interface digunakan jika semua fitur perlu untuk diimplementasikan secara berbeda untuk objek yang berbeda	Abstract class digunakan ketika ada beberapa fitur umum yang dimiliki oleh semua objek
Interface cenderung lebih lambat jika dibandingkan dengan abstract class	Abstract class lebih cepat (tidak semua implementasi perlu ditulis pada child class)

2.1 Informal Interface

Pada beberapa kasus, formal phyton interface akan memiliki aturan yang ketat. Karena phyton bersifat dinamis, maka user dapat membuat informal interface. Pada informal interface, sebuah class yang didefinisasikan sebagai method dapat diganti, namun tidak ada aturan yang ketat.

Informal interface adalah kelas yang mendefinisikan metode atau fungsi-fungsi yang bisa di override/implementasi namun tanpa adanya unsur paksaan (cukup diimplementasi hanya bila dibutuhkan). Untuk mengimplementasikannya kita harus membuat kelas konkrit. Kelas konkrit adalah subclass dari (abstrak) interface yang menyediakan implementasi dari method atau fungsi-fungsi di kelas interface. Untuk melakukan implementasi menggunakan inheritance.

```
class Subject:
   def __init__(self, room, capacity):
       self.room = room
       self.capacity = capacity
   def __str__(self):
       return 'Class info: room {} and capacity
{}'.format(self.room, self.capacity)
s1 = Subject('GKU 312', 40)
s2 = Subject('GKU 303', 42)
print(s1)
print(s2)
```

```
Output:-


Class info: room GKU 312 and capacity 40


Class info: room GKU 303 and capacity 42
```

Program di atas memudahkan user untuk mencetak object. Dengan menggunakan __str___() method, maka method yang diinginkan dapat dipanggil. Program akan memberikan output, dari letak object, dalam format string sesuai dengan output di atas.

2.2 Formal Interface

Pada formal interface kelas parent (abstrak) dapat dibangun hanya dengan sedikit baris kode, untuk kemudian diimplementasikan pada kelas turunannya (konkret). Implementasi ini bersifat wajib/dipaksakan sehingga dinamakan formal interface.

```
from abc import ABC, abstractmethod
class Bank(ABC):
    @abstractmethod
    def balance(self):
        pass
    @abstractmethod
    def income(self):
        pass

class Info(Bank):
    def balance(self):
        print("Your balance is Rp.200.000")
    def income(self):
        print("Your income this month is Rp.45.000")


s = Info()
s.balance()
s.income()
```

```
Output:-

Your balance is Rp.200.000

Your income this month is Rp.45.000
```

3. **Metaclass**

Metaclass adalah sebuah class di mana instansinya adalah sekumpulan class. Sebuah metaclass menyerupai sifat dari setiap class dan instansinya. Metaclass berfungsi untuk memodifikasi tingkah laku dari setiap class yang dibuat dengan clss tersebut.

Oleh karena itu, kita bisa saja membuat sebuah kelas turunan (subclass) dari int atau tipe dasar lainnya yang memiliki karakteristik tambahan. Konsep ini juga berlaku pada objek dari kelas yang masih kosong.

```
class Data(type):
    def __new__(mcs, name, bases, class_dict):
        class_obj = super().__new__(mcs, name, bases, class_dict)
        return class_obj
```

Kesimpulan

1. Apa itu interface dan kapan kita perlu memakainya?
   Interface adalah koleksi dari method atau functions yang perlu disediakan oleh child class. Method pada interface bersifat abstract dan methode abstract tidak memiliki implementasi dan akan memiliki satu-satunya deklarasi. Interface dapat diterapkan ketika user akan membuat class child dari class parent.
2. Apa itu kelas abstrak dan kapan kita perlu memakainya? Apa perbedaannya dengan interface?

   Abstraksi digunakan untuk menyembunyikan fungsi internal dari user. User hanya dapat mengakses implementasi dasar dari program, namun operasi di dalam program akan tersembunyi. Sederhananya, user familiar dengan apa yang dilakukan program tanpa tahu bagaimana program itu berjalan. Abstract class berbeda dari interface. Perbedaannya terdapat di implementasinya. Interface digunakan jika semua fitur perlu untuk diimplementasikan secara berbeda untuk objek yang berbeda. Abstract class digunakan ketika ada beberapa fitur umum yang dimiliki oleh semua objek.
3. Apa itu kelas konkret dan kapan kita perlu memakainya?
   Kelas konkret dapat digunakan ketika objek pada kelas abstrak tidak bisa instansiasi. Kelas konkret merupakan kelas yang kelas dapat instansiasi objeknya.
4. Apa itu metaclass dan kapan kita perlu memakainya? Apa bedanya dengan inheritance biasa?
   Metaclass adalah sebuah class di mana instansinya adalah sekumpulan class. Sebuah metaclass menyerupai sifat dari setiap class dan instansinya. Metaclass berfungsi untuk memodifikasi tingkah laku dari setiap class yang dibuat dengan class tersebut. Berbeda dengan inheritance, yang tidak dapat memodifikasi tingkah laku setiap class, dan hanya bisa mewarisi constructor dari sebuah class.

Daftar Pustaka

Mwiti, D. (2018, December). *Introduction to Python Metaclasses*. Diambil kembali dari datacamp: https://www.datacamp.com/tutorial/python-metaclasses

Python-interface module. (2020, Maret 22). Diambil kembali dari Geeksforgeek: https://www.geeksforgeeks.org/python-interface-module/

Rombang Mathew Raphael Clinton, R. S. (t.thn.). *Python Metaclass Example*. Diambil kembali dari Phyton Tutorial: https://www.pythontutorial.net/python-oop/python-metaclass-example/

Thamrin, H. (2017). PERANCANGAN TOOLS BERBASIS PYTHON UNTUK MEMANTAU KEAKTIFAN SERVER. *Jurnal Teknologi dan Komunikasi*, 9.