TUGAS 6

Disusun oleh:

Nama : Hasna Dhiya Azizah
NIM : 121140029
Kelas : Praktikum PBO RB

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2023

1. Kelas Abstrak

Kelas abstrak merupakan sebuah kelas yang masih berada dalam bentuk abstrak, sementara bentuk abstrak tersebut tidak bisa dibuat menjadi objek secara langsung. Agar dapat disebut sebagai kelas abstrak, sebuah kelas setidaknya harus memiliki satu atau lebih *method* abstrak. *Method* abstrak adalah metode yang tidak memiliki implementasi atau bentuk konkret. Sementara itu, abstraksi data merupakan sebuah proses untu menyembunyikan detail tertentu sehingga hanya akan menampilkan informasi penting kepada penggunanya. Di Python, metode abstrak dapat dijalankan namun tidak memiliki implementasi apa pun. Kelas yang memiliki satu atau lebih metode abstrak disebut dengan kelas abstrak. Kelas abstrak tidak dapat dibuat *instance*-nya, sementara itu metode abstrak diperlukan untuk mengimplementasikan sub-kelas.

```
 8  public class abstraction {
 9      public static void main(String[] args)  {
10          Child a=new Child();
11          a.eat()
12          Child.sleep();}}
13  abstract class Parent{ //abstract class
14  abstract void eat(); //abstract method
15  public static void sleep(){//static method
16      system.out.printIn("sleeping");}}
17  class Child extends Parent {
18          public void eat() {
19              system.out.printIn("Eating");}}
20
```

Pada dasarnya kelas abstrak belum bisa digunakan secara langsung sehingga memerlukan bentuk konkret dengan cara membuat implementasi dari metode-metode yang masih abstrak. Python hadir dengan modul yang menyediakan basis untuk mendefinisikan Kelas Basis Abstrak (KBA). Modul KBA bekerja dengan menggunakan metode dekorasi kelas dasar sebagai sebuah bentuk abstrak dan kemudian akan mendaftarkan kelas sebagai implementasi dari basis abstrak. Implementasi kelas abstrak tersebut dapat dilakukan dengan membuat pewarisan (inheritance). Kelas abstrak biasanya digunakan sebagai induk kelas dari kelas-kelas yang lain sehingga kelas anak akan membuat versi konkret dari kelas abstrak.

Sebuah kelas harus dibuat abstrak karena pada kondisi tertentu, kelas induk tidak dapat dijadikan objek karena tidak memiliki bentuk implementasi method yang jelas. Kelas abstrak ini dapat digunakan ketika penggunanya hendak menyediakan interface umum dalam proses mengimplementasi komponen kelas yang berbeda. Proses tersebut membuat kelas harus dijadikan ke bentuk abstrak.

```
 8  class Database {
 9    String getTableName(){
10      return null;
11    }
12  }
13
14  class MySQLDatabase extends Database {
15    @override
16    String getTableName(){
17      return this.sql("SELECT table_name FROM database.tabel")
18    }
19  }
```

Kelas abstrak berisi sebagian implementasi serta subkelas yang berfungsi untuk melengkapi bentuk implementasinya. Pada kasus seperti ini, method abstrak dapat digunakan apabila terdapat satu atau lebih subclass yang berpotensi memiliki fungsionalitas yang sama namun dengan bentuk implementasi yang berbeda. Sementara itu, kelas abstrak berfungsi untuk mendefinisikan *behavior* secara umum sebagai sebuah bentuk superkelas ketika subkelas dapat menyediakan bentuk-bentuk yang lebih detail.

2. Interface

Pada Python, interface dapat didefinisikan dengan menggunakan pernyataan kelas Python. Sebuah interface dapat memperluas interface lain dengan mencantumkan interface lain sebagai sebuah interface dasar. Pada tingkat tinggi, interface berperan sebagai *blueprint* untuk mendesain kelas. Seperti kelas, interface juga akan menentukan metode, namun metode tersebut terbentuk secara abstrak. Metode abstrak merupakan sebuah metode yang didefinisikan oleh interface dan interface dapat memberi arti yang konkret pada metode abstrak interface.

Python tidak memerlukan kelas yang mengimplementasikan interface untuk dapat mendefinisikan semua metode abstrak interface, sehingga dalam keadaan tertentu tidak diperlukan aturan ketat dari interface Python secara formal. Sifat dinamis Python memungkinkan penggunanya untuk mengimplementasikan interface secara informal. Interface informal Python adalah kelas yang mendefinisikan metode yang dapat diganti namun tidak memiliki aturan yang ketat. Dalam contoh berikut, digunakan sebuah perspektif seorang ahli data yang perlu mengekstraksi teks dari berbagai jenis file tidak terstruktur, seperti PDF dan email. Pada prosesnya, akan dibuat interface informal dengan menentukan metode yang ada lalu akan diberikan pada dua kelas beton yakni kelas PdfParser dan kelas EmlParser

```
8   class InformalParserInterface:
9       def load_data_source(self, path: str, file_name: str) -> str:
10          """Load in the file for extracting text."""
11          pass
12
13      def extract_text(self, full_file_name: str) -> dict:
14          """Extract text from the currently loaded file."""
15          pass
```

Pada script di atas, InformalParserInterface mendefinisikan adanya dua metode, yakni metode.load_data_source() dan .extract_text(). Metode-metode ini didefinisikan tetapi tidak diimplementasikan sehingga proses implementasinya akan terjadi setelah pembuatan kelas konkret yang akan mewarisi InformalParserInterface. Untuk menggunakan interface, diperlukan pembuatan kelas konkret. Kelas beton merupakan sebuah subkelas interface yang menyediakan bentuk implementasi dari metode interface. Pada kasus ini, dibuat dua kelas konkret untuk mengimplementasikan interface. Kelas pertama adalah pembuatan PdfParser yang akan digunakan untuk mengurai teks dari file PDF, dan selanjutnya adalah pembuatan EmlParser yang akan digunakan untuk mengurai teks dari email. Pada script di bawah dapat dilihat bentuk implementasi konkret dari InformalParserInterface sehingga memungkinkan penggunanya untuk dapat mengekstrak teks dari file email.

```
#class PdfParser
    class PdfParser(InformalParserInterface):
    """Extract text from a PDF"""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides InformalParserInterface.load_data_source()"""
        pass

    def extract_text(self, full_file_path: str) -> dict:
        """Overrides InformalParserInterface.extract_text()"""
        pass

#class EmlParser
class EmlParser(InformalParserInterface):
    """Extract text from an email"""
    def load_data_source(self, path: str, file_name: str) -> str:
        """Overrides InformalParserInterface.load_data_source()"""
        pass

    def extract_text_from_email(self, full_file_path: str) -> dict:
        """A method defined only in EmlParser.
        Does not override InformalParserInterface.extract_text()
        """
        pass

```

Interface informal seperti contoh di atas baik digunakan untuk proyek kecil dengan hanya menggunakan beberapa pengembang yang mengerjakan kode sumber. Namun, saat proyek semakin besar dan tim bertambah, penggunaan Interface informal dapat menyebabkan pengembang kehabisan waktu selama berjam-jam hanya untuk mencari kesalahan logika yang sulit ditemukan dalam sebuah basis kode.

Sementara itu, interface formal merupakan interface yang diberlakukan secara formal. Dalam kasus tertentu, protokol atau pengetikan bebek menyebabkan ambiguitas. Untuk membuat interface formal, diperlukan penggunaan kelas basis abstrak yang akan dijelaskan ketika mendefinisikan suatu kelas yang bersifat abstrak. Pada contoh di bawah, akan dimasukkan modul ABC untuk mendefinisikan kelas makanan. *@abc.abstractmethod* merupakan dekorator yang menunjukkan metode abstrak.

```
import abc
class Food (abc.ABC):
  @abc.abstractmethod
  def taste( self ):
      pass
class north_indian(Food) :
  def taste(self):
     print(" Cooking! ")
s = north_indian ()

print( isinstance(s, Food))
```

3. Metakelas

Istilah metaprogramming mengacu pada potensi suatu program untuk dapat memiliki pengetahuan atau dapat memanipulasi dirinya sendiri. Python mendukung bentuk metaprogramming untuk kelas yang disebut metaclasses. Metaclass merupakan sebuah konsep OOP esoteris yang bersembunyi dibalik sebagian besar kode Python. Python menyediakan kemampuan yang tidak didukung oleh semua bahasa berorientasi objek yang dapat dibuka dengan menentukan metaclass khusus. Memahami metaclass pada Python dapat memberikan pemahaman yang lebih baik tentang *internal class* Python secara umum.

Metaclass dapat memeriksa apakah kelas dasar dalam sebuah program memiliki atribut kelasnya tersendiri. Jika demikian, turunan dari kelas yang diberikan dalam atribut kelas tersebut akan membentuk sebuah kelas baru. Kelas yang berada dalam sebuah atribut kelas akan disebut sebagai metaclass. Metaclass dapat menyimpan informasi tentang sebuah kelas ketika kelas tersebut telah berhasil dibuat. Sebagai contoh, perlu diperhatikan dampak yang terjadi ketika sebuah turunan dari Metaclass dibuat yang diilustrasikan dengan kelas sederhana. Contoh Metaclass di bawah ini akan menyimpan informasi tentang kelas ketika kelas tersebut dibuat. Pada contoh di atas, telah dibuat kelas yang merupakan turunan dari metaclass MyMetaClass.

```
8     class MyMetaClass:
9         def _init_(self, name, bases, namespace):
10            self._name_= name
11            self._bases_= bases
12            self._namespace_= namespace
13
14        def _call_(self ):
15            return Instance(self)
16
```

Pada contoh lain, sebuah kelas Human yang ditunjukkan di bawah ini akan dibuat seperti menggunakan tipe. Pertama, telah ditentukan metaclass khusus yang disebut Human dan memiliki atribut yang disetel ke *True* secara default. Perhatikan bahwa metode __new__ pada script di bawah ini akan mengembalikan kelas baru atau objek kelas. Setelah itu, ditentukan pula kelas Person yang menggunakan metaclass Human. Kelas Person akan memiliki atribut seperti yang ditunjukkan pada variabel kelas:

```
7
8     class Human(type):
9         def __new__(mcs, name, bases, class_dict):
10            class_ = super().__new__(mcs, name, bases, class_dict)
11            class_.freedom = True
12            return class_
13
14    class Person(object, metaclass=Human):
15        def __init__(self, name, age):
16            self.name = name
17            self.age = age
18    print(Person.__dict__)

In [6]: runfile('C:/Users/windows/untitled2.py', wdir='C:/Users/windows')
{'__module__': '__main__', '__init__': <function Person.__init__ at 0x000001F81365DA60>,
'__dict__': <attribute '__dict__' of 'Person' objects>, '__weakref__': <attribute
'__weakref__' of 'Person' objects>, '__doc__': None, 'freedom': True}
```

4. Kesimpulan

Berdasarkan materi yang telah dibuat, dapat disimpulkan bahwa kelas abstrak merupakan sebuah kelas yang memiliki satu atau lebih metode abstrak. Kelas abstrak digunakan untuk membuat struktur logika penurunan dalam proses pemrograman objek. Peggunaan kelas abstrak bertujuan untuk mengimplementasikan sebuah method yang sama ke seluruh class yang diturunkan oleh kelas abstrak. Sementara itu, interface merupakan representasi dari sebuah tindakan yang menggambarkan perilaku sebuah kelas dalam satu sistem. Interface dapat digunakan untuk menangkap kesamaan di antara kelas yang tidak terkait tanpa memaksakan adanya hubungan antar kelas dalam satu program. Selain itu, interface juga dapat digunakan untuk menerapkan metode pada satu atau lebih kelas. Sementara itu,

kelas konkret digunakan untuk mengimplementasikan interfacenya. Metaclass adalah kelas dari kelas yang dapat mendefinisikan perilaku kelasnya sebagai sebuah turunan dari Metaclass. Metaclass dapat digunakan ketika seseorang ingin mengubah nama kelas yang telah dibuat dalam suatu program.

DAFTAR PUSTAKA

Andersen, A. (1998). A note on reflection in Python 1.5. *Lancaster University*, *46*.

Geeksforgeeks.org. Python-interface module. Diakses pada 12 April 2023, dari: https://www.geeksforgeeks.org/python-interface-module/

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative analysis of Python and Java for beginners. *Int. Res. J. Eng. Technol*, *7*(8), 4384-4407.

Pythonguides.com. Introduction to Python Interface. Diakses pada 12 April 2023, dari https://pythonguides.com/python-interface/

Realpython.com. Implementing an Interface in Python. Diakses pada 12 April 2023, dari https://realpython.com/python-interface/

Realpython.com. Python Metaclasses. Diakses pada 12 April 2023, dari https://realpython.com/python-metaclasses/