LAPORAN PRAKTIKUM
PEMROGRAN BERBASIS OBJEK

Disusun oleh:

Akhmad Fadilla Akbar (121140130)

Program Studi Teknik Informatika

Institut Teknologi Sumatera

2023

1. **Kelas Abstrak**

1.1 Defisini

Kelas abstra dapat dianggap sebagai cetak biru atau kerangka kerja untuk kelas lain. Ini berfungsi sebagai panduan untuk membuat kelas anak yang harus mengimplementasikan fungsi-fungsi yang dideklarasikan di dalam kelas abstrak tersebut. Sebuah kelas dianggap sebagai kelas abstrak jika memiliki satu atau lebih fungsi abstrak di dalamnya. Fungsi abstrak adalah fungsi yang hanya memiliki deklarasi tanpa implementasi. Ini berarti bahwa kelas abstrak tidak dapat diinstansiasi menjadi objek langsung. Sebaliknya, kelas anak yang diturunkan dari kelas abstrak harus mengimplementasikan semua fungsi abstrak yang dideklarasikan di kelas abstrak tersebut.

1.2 Alasan Penggunaan

Dengan menggunakan abstraksi kelas dasar, Anda dapat membuat definisi antarmuka Program Aplikasi yang dapat digunakan oleh sekelompok kelas anak. Fungsi ini sangat berguna ketika Anda perlu menyediakan implementasi oleh pihak ketiga, seperti dalam penggunaan plugin. Selain itu, ini juga dapat memberikan manfaat saat bekerja dalam tim besar atau dengan basis kode yang luas.

1.3 Cara Kerja Kelas Abstrak

Secara default, Python tidak menyediakan kelas abstrak. Python hadir dengan modul yang menyediakan basis untuk mendefinisikan kelas Basis Abstrak ABC) dan nama modul itu adalah ABC. ABCbekerja dengan fungsi dekorasi kelas dasar sebagai abstrak dan kemudian mendaftarkan kelas konkret sebagai implementasi dari basis abstrak. Sebuah fungsi menjadi abstrak ketika dihias dengan kata kunci @abstractmethod.

1.4 Implementasi Kelas Abstrak

```python
from abc import ABC, abstractmethod

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

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

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

    def calculate_area(self):
        return 3.14 * self.radius * self.radius

# Membuat objek Rectangle
rectangle = Rectangle(5, 3)
print("Luas persegi panjang:", rectangle.calculate_area())

# Membuat objek Circle
circle = Circle(4)
print("Luas lingkaran:", circle.calculate_area())
```

2. **Interface**

2.1 Definisi

Dalam bahasa pemrograman berorientasi objek seperti Python, antarmuka (interface) adalah kumpulan tanda tangan fungsi yang harus diberikan oleh kelas yang mengimplementasikannya. Implementasi antarmuka memungkinkan penulisan kode yang terorganisir dan mencapai abstraksi.

Secara umum, antarmuka (interface) berfungsi sebagai blueprint atau cetak biru untuk merancang kelas-kelas. Seperti halnya kelas-kelas, antarmuka (interface) mendefinisikan fungsi-fungsi. Namun, perbedaannya adalah fungsi-fungsi dalam antarmuka bersifat abstrak. Fungsi abstrak adalah fungsi yang didefinisikan dalam antarmuka tanpa implementasi konkret. Implementasi fungsi ini dilakukan oleh kelas-kelas yang mengimplementasikan antarmuka tersebut dan memberikan makna konkret pada fungsi abstrak dari antarmuka.

2.2 Implementasi Interface

Terdapat sebuah antarmuka **Shape** yang mendefinisikan dua metode abstrak**: area()** dan **perimeter()**. Kelas **Circle** dan **Rectangle** mengimplementasikan antarmuka **Shape** dengan menyediakan implementasi konkret untuk kedua metode abstrak tersebut. Kelas **Circle** menghitung luas dan keliling lingkaran berdasarkan jari-jari yang diberikan saat objek diinisialisasi. Sementara itu, kelas **Rectangle** menghitung luas dan keliling persegi panjang berdasarkan panjang dan lebar yang diberikan saat objek diinisialisasi. Pada contoh ini, antarmuka **Shape** berperan sebagai kontrak yang menjamin bahwa setiap kelas yang mengimplementasikan antarmuka tersebut harus menyediakan metode **area()** dan **perimeter().**

```python
from abc import ABC, abstractmethod

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

    @abstractmethod
    def perimeter(self):
        pass

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

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

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

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

# Menggunakan objek Circle
circle = Circle(5)
print("Luas lingkaran:", circle.area())
print("Keliling lingkaran:", circle.perimeter())

# Menggunakan objek Rectangle
rectangle = Rectangle(4, 6)
print("Luas persegi panjang:", rectangle.area())
print("Keliling persegi panjang:", rectangle.perimeter())
```

3. **Metaclass**

3.1 Definisi

Metaclass adalah sebuah kelas yang digunakan untuk membuat kelas lainnya. Metaclass mengontrol cara pembuatan objek kelas dan perilaku kelas tersebut. Dalam bahasa pemrograman Python, metaclass memungkinkan kita untuk memanipulasi atau memodifikasi perilaku default dari kelas. Dalam Python, setiap kelas adalah sebuah objek yang dihasilkan dari sebuah metaclass. Secara default, metaclass dari sebuah kelas adalah type. Ketika kita mendefinisikan sebuah kelas, interpreter Python menggunakan metaclass type untuk membuat objek kelas tersebut.

3.2 Implementasi Metaclass

Mendefinisikan sebuah metaclass bernama **MyMeta**. **Metaclass** ini merupakan turunan dari **type**. Di dalam metaclass, kita mengubah nilai atribut **my_attribut**e menjadi 42 dan menambahkan metode **my_method** ke kelas. Ketika kita membuat objek dari kelas **MyClass**, metaclass **MyMeta** akan dipanggil dan metode **__new__**-nya akan dieksekusi. Di dalam metode **__new__**, kita dapat memodifikasi atau menambahkan atribut dan metode ke kelas. Pada contoh ini, kita mengubah nilai atribut **my_attribute** menjadi 42 dan menambahkan metode **my_method** ke kelas.

```python
class MyMeta(type):
    def __new__(cls, name, bases, attrs):
        # Modifikasi atribut 'my_attribute' dalam kelas
        attrs['my_attribute'] = 42

        # Membuat instance baru dari kelas
        instance = super().__new__(cls, name, bases, attrs)

        # Menambahkan metode baru ke instance
        def my_method(self):
            print("Hello, World!")

        instance.my_method = my_method

        return instance

class MyClass(metaclass=MyMeta):
    pass

obj = MyClass()
print(obj.my_attribute)  # Output: 42
obj.my_method()  # Output: Hello, World!
```

Daftar Pustaka

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

Muhardian, A. (2020, januari 05). *petanicode*. Retrieved from Tutorial OOP: Mengenal Class Abstrak dan Cara Pakainya: https://www.petanikode.com/java-oop-abstract/

Rahmalia, N. (2021, Desember 14). *glints.* Retrieved from Graphical User Interface (GUI), Tampilan yang Sederhanakan Interaksi dengan Komputer: https://glints.com/id/lowongan/gui-adalah/

Sturtz, J. (2017, Mei 07). *Python Metaclasses*. Retrieved from Real Python: https://realpython.com/python-metaclasses/