BAB 4

Polymorphism

4.1 Tujuan

1. Dapat memahami konsep polymorphism pada Pemrograman Berorientasi Objek
2. Dapat mengimplementasikan polymorphism menggunakan Bahasa pemrograman python

4.2 Pengantar

4.2.1 Polymorphism

Polymorphism adalah salah satu konsep dalam OOP (Object-Oriented Programming) di mana suatu objek dapat memiliki banyak bentuk atau tipe data. Dengan Polymorphism kita dapat menggunakan suatu objek dengan cara yang berbeda-beda, tergantung dari konteks dan kebutuhan saat penggunaan.

Konsep polymorphism dapat dijumpai dalam berbagai aspek kehidupan sehari-hari, terutama pada hal-hal yang berkaitan dengan interaksi manusia dengan lingkungan sekitarnya. Berikut adalah beberapa contoh polymorphism :

- Kendaraan

  Kendaraan merupakan contoh nyata dari polymorphism. Meskipun ada banyak jenis kendaraan, seperti mobil, sepeda, motor, kereta, dan lain sebagainya, namun mereka memiliki beberapa karakteristik yang sama, seperti roda, mesin, dan kendali. Dengan begitu maka manusia dapat naik dan mengendarai berbagai jenis kendaraan, meskipun setiap jenis kendaraan memiliki cara pengoperasian yang berbeda.

- Peralatan rumah tangga

  Peralatan rumah tangga seperti blender, mixer, oven, dan mesin cuci, juga dapat menjadi contoh polymorphism. Meskipun setiap peralatan rumah tangga memiliki fungsi yang berbeda-beda, namun mereka memiliki fitur yang sama, seperti tombol on/off, pengaturan kecepatan, dan lain sebagainya.

- Alat musik

  Alat musik juga merupakan contoh nyata dari polymorphism. Meskipun ada banyak jenis alat musik, seperti gitar, drum, biola, piano, dan lain sebagainya, namun alat-alat tersebut memiliki beberapa karakteristik yang sama, seperti nada, ritme, dan tempo. Sehingga kita dapat bermain dan memainkan

berbagai jenis alat musik, meskipun setiap jenis alat musik memiliki teknik dan cara bermain yang berbeda.

4.2.2 Polymorphism pada Python

Konsep polymorphism pada Python dapat ditemukan di dalam library bawaan Python. Sebagai contoh, kita dapat menggunakan fungsi len() untuk mendapatkan panjang dari objek yang berbeda-beda, seperti string, list, tuple, set, dan dictionary. Meskipun objek-objek tersebut memiliki tipe data yang berbeda, namun kita dapat memanggil fungsi len() pada setiap objek tersebut dengan cara yang sama, sehingga fungsi tersebut bersifat polymorphic.

Berikut ini adalah contoh penggunaan fungsi len() pada beberapa tipe data yang berbeda:

```
1. my_string = "Hello, World!"
2. my_list = [1, 2, 3, 4, 5]
3. my_tuple = (6, 7, 8, 9, 10)
4. my_set = {11, 12, 13, 14, 15}
5. my_dict = {"name": "John", "age": 30, "city": "New York"}
6.
7. print(len(my_string))   # output: 13
8. print(len(my_list))     # output: 5
9. print(len(my_tuple))    # output: 5
10.print(len(my_set))      # output: 5
11.print(len(my_dict))     # output: 3
```

Polymorphism dalam OOP dapat diimplementasikan melalui dua cara, yaitu:

- Overloading: mengimplementasikan fungsi atau method dengan nama yang sama, tetapi dengan parameter yang berbeda. Dengan demikian, saat pemanggilan fungsi atau method, program akan memilih implementasi yang sesuai dengan parameter yang diberikan.

- Overriding: mengimplementasikan method yang sama di dalam class yang berbeda, Dengan demikian, saat pemanggilan method pada objek, program akan memilih implementasi method pada setiap classnya.

4.2.3 Overloading Polymorphism

Pada contoh kode berikut, class Calculator memiliki dua method yang bernama sama yaitu add. Namun, keduanya memiliki jumlah parameter yang berbeda. Saat pemanggilan method, program akan memilih method yang sesuai dengan jumlah parameter yang diberikan. Jika diberikan dua parameter, program akan memilih method add dengan dua parameter, dan jika diberikan tiga parameter, program akan memilih method add dengan tiga parameter. Dengan demikian, kita dapat menggunakan class Calculator dengan cara yang berbeda-beda, tergantung pada kebutuhan penggunaannya.

```
1. class Calculator:
2.     def add(self, a, b):
3.         return a + b
4.
5.     def add(self, a, b, c):
6.         return a + b + c
7.
8. # membuat objek calculator
9. my_calculator = Calculator()
10.
11.# memanggil method add dengan dua parameter
12.result1 = my_calculator.add(2, 3)
13.print(result1)   # output: 5
14.
15.# memanggil method add dengan tiga parameter
16.result2 = my_calculator.add(2, 3, 4)
17.print(result2)   # output: 9
```

4.2.4 Overriding Polymorphism

Pada contoh kode berikut, class Cow memiliki method sound yang menghasilkan output "Moo". Namun, class Dog dan Cat yang memiliki method sound yang sama, namun dengan implementasi yang berbeda di masing-masing class. Ketika method sound dipanggil pada objek Dog, program akan memilih implementasi method pada class Dog, yaitu "Bark Bark". Demikian juga ketika method sound dipanggil pada objek Cat, program akan memilih implementasi method pada class Cat, yaitu "Meooow". Dengan demikian, kita dapat menggunakan class Cow, Dog, dan Cat dengan cara yang berbeda-beda, tergantung pada kebutuhan penggunaannya..

```
1. class Cow:
2.     def __init__(self, name):
3.         self.name = name
4.
5.     def sound(self):
6.         print("Moooo")
7.
8. class Dog:
9.     def __init__(self, name):
10.        self.name = name
11.
12.    def sound(self):
13.        print("Bark Bark")
14.
15.class Cat:
16.    def __init__(self, name):
17.        self.name = name
18.
19.    def sound(self):
20.        print("Meoooww")
21.
22.my_cow = Cow("Sapi")
23.my_cow.sound()
24.my_dog = Dog("Anjing")
25.my_dog.sound()
26.my_cat = Cat("Kucing")
27.my_cat.sound()
```

4.3 Kegiatan Praktikum

4.3.1 Kegiatan 1 : Konsep Polymorphism

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
# contoh penggunaan duck typing
def print_all(items):
    for item in items:
        print(item)

# list
my_list = [1, 2, 3]

# tuple
my_tuple = (4, 5, 6)

# string
my_string = "Hello, world!"

# memanggil fungsi print_all dengan parameter my_list, my_tuple, dan my_string
print_all(my_list)    # output: 1 2 3
print_all(my_tuple)   # output: 4 5 6
print_all(my_string)  # output: H e l l o ,   w o r l d !
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.3.2 Kegiatan 2 : Polymorphism pada OOP

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country.")

```

```
class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
    country
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.3.3 Kegiatan 3 : Polymorphism pada OOP

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
class Shape:
    def area(self):
        pass

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

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

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

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

# membuat objek rectangle
```

```
my_rectangle = Rectangle(5, 6)

# memanggil method area pada objek rectangle
result1 = my_rectangle.area()
print(result1)   # output: 30

# membuat objek circle
my_circle = Circle(7)

# memanggil method area pada objek circle
result2 = my_circle.area()
print(result2)   # output: 153.86
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.3.4 Kegiatan 4 : Polymorphism pada OOP

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

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

    def speak(self):
        print(self.name, 'says bark, bark, bark!')

class Cat():
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(self.name, 'says meeeoooow')

class Bird():
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(self.name, 'says tweet')

oDog1 = Dog('Rover')
oDog2 = Dog('Fido')
oCat1 = Cat('Fluffy')
oCat2 = Cat('Spike')
```

```
26.oBird = Bird('Big Bird')
27.
28.petsList = [oDog1, oDog2, oCat1, oCat2, oBird]
29.
30.for oPet in petsList:
31.    oPet.speak()
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.4 Tugas

1. Buatlah 3 buah class yaitu Mobil, Pesawat dan Kapal, setiap class memiliki method yang sama bernama move(). Di dalam Method move() terdapat fungsi print yang memiliki output berbeda tergantung dengan class dimana method tersebut berada.

   Output move() pada class Mobil = Berjalan di jalan raya

   Output move() pada class Pesawat = Terbang di udara

   Output move() pada class Kapal = Berlayar di laut