Pertemuan-3

Polymorphism

A. Polymorphism

Polymorphism adalah kemampuan suatu objek untuk mengambil banyak bentuk atau tampilan yang berbeda. Dalam pemrograman berorientasi objek, polymorphism mengacu pada kemampuan suatu objek untuk memiliki banyak bentuk atau perilaku yang berbeda tergantung pada konteks di mana objek tersebut digunakan.

Ada dua jenis polymorphism:

1. polymorphism statis (overload)
2. polymorphism dinamis (overriding).

1. Polymorphism Statis (Overload)

Polymorphism statis (Static Polymorphism) adalah suatu jenis polymorphism dimana pemilihan metode yang dieksekusi dilakukan pada saat kompilasi, bukan saat runtime seperti pada polymorphism dinamis.

Polymorphism statis sering juga disebut dengan nama lain yaitu "overloading" atau "compile-time polymorphism". Hal ini karena pada saat kompilasi, kompiler akan menentukan metode mana yang harus dieksekusi berdasarkan jumlah dan tipe parameter yang digunakan dalam pemanggilan metode.

Contoh sederhana polymorphism statis adalah ketika kita memiliki suatu kelas yang memiliki beberapa metode dengan nama yang sama, namun dengan jumlah dan tipe parameter yang berbeda. Pada saat pemanggilan metode, kompiler akan menentukan metode mana yang harus dieksekusi berdasarkan jumlah dan tipe parameter yang digunakan.

Berikut adalah contoh sederhana penggunaan polymorphism statis pada bahasa pemrograman Python:

Contoh 1:

Penggunaan fungsi “len” pada berbagai tipe data:

```
print(len("Hello")) # output: 5
print(len([1, 2, 3])) # output: 3
print(len((1, 2, 3))) # output: 3
```

Contoh 2:

Penggunaan operator “+” pada berbagai tipe data:

```
print(2 + 3) # output: 5
print("Hello" + " World") # output: Hello World
print([1, 2] + [3, 4]) # output: [1, 2, 3, 4]
```

Contoh 3:

Penggunaan fungsi “max” pada berbagai tipe data:

```
print(max(2, 5)) # output: 5
print(max([1, 2, 3])) # output: 3
print(max("Hello")) # output: "o"
```

Contoh 4:

Penggunaan metode “sort” pada berbagai tipe data:

```
a = [3, 1, 2]
a.sort()
print(a) # output: [1, 2, 3]

b = ["c", "a", "b"]
b.sort()
print(b) # output: ["a", "b", "c"]
```

2. Polymorphism Dinamis (Overriding)

Polymorphism dinamis adalah konsep dalam pemrograman berorientasi objek yang memungkinkan objek dari kelas yang berbeda untuk menunjukkan perilaku yang sama saat menggunakan metode yang sama.

Polymorphism dinamis dapat membantu membuat kode lebih fleksibel dan mudah diubah pada masa depan. Dalam pemrograman berorientasi objek, konsep ini sering digunakan untuk memungkinkan beberapa objek untuk berinteraksi dengan cara yang sama, bahkan jika objek-objek tersebut memiliki perilaku yang berbeda.

Contoh:

```
class Matematika:
    def add(self, a, b):
        return a + b

    def add(self, a, b, c=0):
        return a + b + c

mat = Matematika()
B = mat.add(5, 3, 4)
print(B)

mut = Matematika()
C = mut.add(7,3)
print(C)
```

Pada contoh di atas, kelas Matematika memiliki dua metode "add" dengan jumlah dan tipe parameter yang berbeda. Pada saat pemanggilan metode "add", kompiler akan menentukan metode mana yang harus dieksekusi berdasarkan jumlah dan tipe parameter yang digunakan. Dalam contoh tersebut, karena pemanggilan metode "add" menggunakan tiga parameter, maka yang dieksekusi adalah metode "add" yang menerima tiga parameter. Oleh karena itu, output yang dihasilkan adalah 12 (5 + 3 + 4) dan 10 (7 + 3)

Implementasi Polymorphism pada perilaku objek saat runtime

Dalam polymorphism dinamis, perilaku objek dapat diubah pada saat runtime. Ini juga dikenal sebagai late binding, dynamic binding, atau runtime polymorphism.

Contoh:

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

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

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

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

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

shapes = [Rectangle(3, 4), Circle(5)]
for shape in shapes:
    print(shape.area())
```

Pada contoh di atas, terdapat kelas Shape sebagai kelas induk, serta dua kelas turunan Rectangle dan Circle yang masing-masing memiliki metode area() yang berbeda. Kemudian, objek-objek dari kelas-kelas tersebut dimasukkan ke dalam sebuah list shapes.

Pada saat dilakukan iterasi di dalam list shapes dan dipanggil metode area(), Python secara dinamis menentukan metode area() mana yang harus dipanggil berdasarkan tipe objek yang sedang diiterasi. Sebagai contoh, pada iterasi pertama dengan objek Rectangle, Python akan memanggil metode area() pada kelas Rectangle, sedangkan pada iterasi kedua dengan objek Circle, Python akan memanggil metode area() pada kelas Circle. Inilah yang disebut dengan dynamic binding.

Implementasi Polymorphism Dinamis pada pewarisan dan pengikatan dinamis

Contoh konsep polymorphism dinamis yang lain adalah dengan menggunakan pewarisan dan pengikatan dinamis. Misalnya, sebuah kelas yang mewarisi sebuah metode dari kelas induknya dapat mengubah perilaku metode tersebut pada saat runtime, baik dengan cara menambahkan fungsi tambahan atau dengan cara mengubah logika di dalam metode tersebut.

Contoh:

```
class Animal:
    def make_sound(self):
        print("The animal makes a sound")

class Dog(Animal):
    def make_sound(self):
        print("The dog barks")

class Cat(Animal):
    def make_sound(self):
        print("The cat meows")

class Chihuahua(Dog):
    def make_sound(self):
        print("The chihuahua yaps")

class Siamese(Cat):
    def make_sound(self):
        print("The Siamese purrs")

def animal_sound(animal):
    animal.make_sound()

# Instantiate objects of each class
animal = Animal()
dog = Dog()
cat = Cat()
chihuahua = Chihuahua()
siamese = Siamese()

# Call the animal_sound function for each object
animal_sound(animal)    # Output: The animal makes a sound
animal_sound(dog)       # Output: The dog barks
animal_sound(cat)       # Output: The cat meows
animal_sound(chihuahua) # Output: The chihuahua yaps
animal_sound(siamese)   # Output: The Siamese purrs
```

Dalam contoh di atas, kelas Animal memiliki metode make_sound yang dicontoh oleh kelas-kelas turunannya seperti Dog dan Cat. Kemudian, kelas Chihuahua dan Siamese mewarisi metode make_sound dari kelas induknya tetapi mengubah perilaku metode tersebut dengan menambahkan suara yang lebih spesifik untuk jenis binatang tersebut. Pada saat runtime, ketika metode make_sound dipanggil untuk objek chihuahua atau siamese, versi yang diperbarui dari metode tersebut akan dipanggil.

Implementasi Polymorphism Dinamis pada Abstract Class:

Salah satu contoh penerapan polymorphism dinamis adalah saat menggunakan pewarisan dan abstraksi. Ketika sebuah kelas turunan mengimplementasikan metode yang telah didefinisikan pada kelas induknya, ia dapat menyesuaikan perilaku metode tersebut dengan menambahkan atau mengubah perilaku yang dimiliki.

Contoh:

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass

class Car(Vehicle):
    def start(self):
        print("Starting car...")

class Motorcycle(Vehicle):
    def start(self):
        print("Starting motorcycle...")

class Bus(Vehicle):
    def start(self):
        print("Starting bus...")

vehicles = [Car(), Motorcycle(), Bus()]

for vehicle in vehicles:
    vehicle.start()
```

Pada contoh kode di atas, terdapat class abstract Vehicle yang memiliki method abstract start(). Kemudian, class Car, Motorcycle, dan Bus masing-masing mengimplementasikan method start() sesuai dengan kebutuhan mereka.

Pada bagian akhir kode, terdapat list vehicles yang berisi objek-objek dari class Car, Motorcycle, dan Bus. Kemudian, dengan menggunakan loop for, setiap objek di dalam list tersebut dipanggil method start(). Karena masing-masing objek memiliki implementasi yang berbeda pada method start(), maka akan terjadi polymorphism dinamis di mana setiap objek akan memanggil implementasi method start() sesuai dengan class-nya masing-masing.

Implementasi Polymorphism Dinamis dalam Duck Typing

Duck typing adalah konsep dalam pemrograman berorientasi objek yang digunakan dalam bahasa pemrograman dinamis seperti Python. Konsep ini mengizinkan programmer untuk memeriksa karakteristik objek, bukan jenis objek, untuk memutuskan apakah objek tersebut dapat digunakan untuk operasi tertentu.

Istilah "duck typing" berasal dari pepatah "Jika berjalan seperti bebek, berbunyi seperti bebek, dan terlihat seperti bebek, maka itu pasti bebek". Artinya, jika sebuah objek memiliki karakteristik atau perilaku yang diperlukan oleh suatu operasi, maka objek tersebut dapat digunakan untuk operasi tersebut.

Contohnya, mari kita asumsikan kita memiliki dua kelas yang berbeda:

```
class Kucing:
    def bersuara(self):
        print("Meow")

class Anjing:
    def bersuara(self):
        print("Guk guk")
```

Kedua kelas ini memiliki metode yang sama dengan nama bersuara(). Dalam duck typing, kita tidak perlu memeriksa tipe objek secara eksplisit, namun kita cukup memeriksa apakah objek tersebut memiliki metode bersuara().

Sebagai contoh, jika kita ingin mencetak suara dari suatu objek, kita bisa menuliskan kode sebagai berikut:

```
class Kucing:
    def bersuara(self):
        print("Meow")

class Anjing:
    def bersuara(self):
        print("Guk guk")

def cetak_suara(objek):
    objek.bersuara()

kucing = Kucing()
ajing = Anjing()

cetak_suara(kucing) # Output: Meow
cetak_suara(ajing)  # Output: Guk guk
```

Dalam contoh di atas, kita tidak memeriksa apakah objek adalah Kucing atau Anjing. Sebaliknya, kita hanya memeriksa apakah objek tersebut memiliki metode bersuara(), dan

memanggil metode tersebut. Oleh karena itu, Python akan mencetak suara dari kucing dan anjing dengan benar. Itulah yang disebut **duck typing**.

Praktikum:

Buatlah masing-masing 2 contoh polymorphism statis (overload) dan polymorphism dinamis (overriding). Beri nama overload1.py, overload2, overriding1.py, overriding2.py

Tugas:

Berdasarkan contoh berikut:

```
class Animal:
    def make_sound(self):
        print("The animal makes a sound")

class Dog(Animal):
    def make_sound(self):
        print("The dog barks")

class Cat(Animal):
    def make_sound(self):
        print("The cat meows")


def animal_sound(animal):
    animal.make_sound()

# Instantiate objects of each class
animal = Animal()
dog = Dog()
cat = Cat()

# Call the animal_sound function for each object
animal_sound(animal)    # Output: The animal makes a sound
animal_sound(dog)       # Output: The dog barks
animal_sound(cat)       # Output: The cat meows
```

Dalam contoh diatas suara dituliskan dalam kata-kata.

Soal Tugas:

Buatlah sebuah aplikasi sederhana untuk memperdengarkan suara 10 hewan yang berbeda. Format suara bisa .wav atau mp3