Pertemuan-2

Inheritance (Pewarisan)

A. Inheritance

Inheritance adalah sebuah konsep di dalam pemrograman berorientasi objek (OOP) di mana sebuah class mewarisi atribut dan method dari class lain yang disebut sebagai superclass atau parent class. Class yang menerima warisan disebut sebagai subclass atau child class. Dalam konsep inheritance, subclass bisa mengakses dan memodifikasi atribut dan method yang sudah ada di superclass. Ini memungkinkan kita untuk menghindari pengulangan kode dan membuat kode lebih mudah dipelihara dan dimodifikasi.

Contoh:

Mari kita lihat contoh berikut di mana kita ingin membuat sebuah program untuk menghitung luas dan keliling sebuah bangun datar. Ada beberapa jenis bangun datar seperti persegi, persegi panjang, lingkaran, segitiga, dan lain-lain. Semua jenis bangun datar ini memiliki atribut dan method yang sama, seperti panjang sisi dan luas, tetapi mereka memiliki formula dan method yang berbeda untuk menghitung keliling.

Untuk menghindari pengulangan kode, kita bisa membuat superclass BangunDatar yang berisi atribut dan method yang umum untuk semua jenis bangun datar. Kemudian, kita bisa membuat subclass untuk setiap jenis bangun datar yang mewarisi dari BangunDatar.

```
class BangunDatar:
    def __init__(self, sisi):
        self.sisi = sisi

    def luasArea(self):
        pass

class Persegi(BangunDatar):
    def __init__(self, sisi):
        super().__init__(sisi)

    def luasArea(self):
        return self.sisi * self.sisi

    def Nilaikeliling(self):
        return 4 * self.sisi


class Lingkaran(BangunDatar):
```

```
    def __init__(self, jari_jari):
        super().__init__(jari_jari)

    def luasArea(self):
        return 3.14 * self.sisi * self.sisi

    def Nilaikeliling(self):
        return 2 * 3.14 * self.sisi

persegi = Persegi(5)
lingkaran = Lingkaran(7)

print("Luas persegi:", persegi.luasArea())
print("Keliling persegi:", persegi.Nilaikeliling())

print("Luas lingkaran:", lingkaran.luasArea())
print("Keliling lingkaran:", lingkaran.Nilaikeliling())
```

Di dalam contoh ini, **BangunDatar** adalah superclass yang memiliki atribut sisi dan method luasArea(). Subclass Persegi dan Lingkaran mewarisi dari BangunDatar dan menambahkan method khusus Nilaikeliling() yang berbeda untuk menghitung keliling dari masing-masing bangun datar.

Kita bisa membuat objek untuk setiap subclass dan memanggil method luasArea() dan Nilaikeliling(), seperti berikut:

```
persegi = Persegi(5)
lingkaran = Lingkaran(7)

print("Luas persegi:", persegi.luasArea())
print("Keliling persegi:", persegi.Nilaikeliling())

print("Luas lingkaran:", lingkaran.luasArea())
print("Keliling lingkaran:", lingkaran.Nilaikeliling())
```

Hasilnya:

```
Luas persegi: 25

Keliling persegi: 20
Luas lingkaran: 153.86
Keliling lingkaran: 43.96
```

Jenis Inheritance

Dalam pemrograman berorientasi objek ada beberapa jenis inheritance yang mengacu pada cara pewarisan sifat-sifat dari sebuah class ke class lainnya. Dalam Python, ada beberapa jenis inheritance yang dapat digunakan, yaitu:

1. Single Inheritance (Pewarisan Tunggal)
2. Multiple inheritance (Pewarisan Ganda)
3. Hierarchical inheritance (Pewarisan Hirarki)
4. Multi-level Inheritance (Pewarisan Bertingkat)
5. Hybrid Inheritance (Pewarisan Campuran)

1. Single inheritance:

Single inheritance adalah tipe inheritance di mana sebuah class mewarisi sifat-sifat dari satu class induk saja. Contohnya adalah sebagai berikut:

Contoh 1:

```
class Hewan:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def bergerak(self):
        print(self.nama, "bergerak")

class Kucing(Hewan):
    def __init__(self, nama, umur, jenis_bulu):
        super().__init__(nama, umur)
        self.jenis_bulu = jenis_bulu

    def bersuara(self):
        print("Meow!")

kucingA = Kucing("Kitty", 2, "Persia")
kucingA.bergerak() # output: Kitty bergerak
kucingA.bersuara() # output: Meow!
```

Pada contoh di atas, class Kucing mewarisi sifat-sifat dari class Hewan melalui Hewan sebagai class induk atau parent class. Dalam class Kucing, sifat-sifat yang ditambahkan adalah jenis_bulu dan method bersuara.

Contoh 2:

```
class Manusia:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def berbicara(self):
        print(f"{self.nama} sedang berbicara.")

class Dosen(Manusia):
    def __init__(self, nama, umur, nip):
```

```
        super().__init__(nama, umur)
        self.nip = nip

    def mengajar(self):
        print(f"{self.nama} dengan NIP {self.nip} sedang mengajar.")

dosenA = Dosen("Budi", 35, "123456")
dosenA.berbicara() # Output: Budi sedang berbicara.
dosenA.mengajar() # Output: Budi dengan NIP 123456 sedang mengajar.
```

Dalam contoh ini, Dosen mewarisi sifat-sifat dari Manusia, karena Dosen sebenarnya juga adalah Manusia. Sebagai tambahan, Dosen memiliki atribut nip dan method mengajar yang tidak ada di class Manusia.

Contoh 3:

```
class Kendaraan:
    def __init__(self, jenis, merk, warna):
        self.jenis = jenis
        self.merk = merk
        self.warna = warna

    def berkendara(self):
        print("Kendaraan ini sedang berjalan.")

class SepedaMotor(Kendaraan):
    def __init__(self, jenis, merk, warna, cc):
        super().__init__(jenis, merk, warna)
        self.cc = cc

    def belok(self):
        print("Sepeda motor ini sedang belok.")

motorA = SepedaMotor("Sepeda Motor", "Honda", "Merah", 150)
motorA.berkendara() # Output: Kendaraan ini sedang berjalan.
motorA.belok() # Output: Sepeda motor ini sedang belok.
```

Dalam contoh ini, SepedaMotor mewarisi sifat-sifat dari Kendaraan. SepedaMotor memiliki atribut tambahan yaitu cc dan method tambahan belok.

2. Multiple Inheritance

Multiple inheritance adalah tipe inheritance di mana sebuah class mewarisi sifat-sifat dari dua atau lebih class induk. Contohnya adalah sebagai berikut:

Contoh 1:

```
class Mahasiswa:
    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim

    def belajar(self):
        print(self.nama, "sedang belajar")

class Pekerja:
    def __init__(self, nama, pekerjaan):
        self.nama = nama
        self.pekerjaan = pekerjaan

    def bekerja(self):
        print(self.nama, "sedang bekerja")

class MahasiswaPekerja(Mahasiswa, Pekerja):
    def __init__(self, nama, nim, pekerjaan):
        Mahasiswa.__init__(self, nama, nim)
        Pekerja.__init__(self, nama, pekerjaan)

    def bersosialisasi(self):
        print(self.nama, "sedang bersosialisasi")

mhs_pekerja = MahasiswaPekerja("Rahma", "190001", "Programmer")
mhs_pekerja.belajar() # output: Rahma sedang belajar
mhs_pekerja.bekerja() # output: Rahma sedang bekerja
mhs_pekerja.bersosialisasi() # output: Rahma sedang bersosialisasi
```

Contoh 2:

```
class Hewan:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def display_info(self):
        print(f"Nama: {self.nama}")
        print(f"Umur: {self.umur}")

class Reptil:
    def __init__(self, jenis, habitat):
        self.jenis = jenis
        self.habitat = habitat

    def display_info(self):
        print(f"Jenis: {self.jenis}")
        print(f"Habitat: {self.habitat}")
```

```
class Amphibi:
    def __init__(self, metamorfosis, habitat):
        self.metamorfosis = metamorfosis
        self.habitat = habitat

    def display_info(self):
        print(f"Metamorfosis: {self.metamorfosis}")
        print(f"Habitat: {self.habitat}")

class Katak(Reptil, Amphibi):
    def __init__(self, nama, umur, jenis, habitat, metamorfosis):
        Hewan.__init__(self, nama, umur)
        Reptil.__init__(self, jenis, habitat)
        Amphibi.__init__(self, metamorfosis, habitat)

    def display_info(self):
        super().display_info()
        print(f"Nama: {self.nama}")
        print(f"Umur: {self.umur}")
        print(f"Jenis: {self.jenis}")
        print(f"Habitat: {self.habitat}")
        print(f"Metamorfosis: {self.metamorfosis}")

# contoh penggunaan
katakA = Katak("Katak Hijau", 2, "Reptil-Amphibi", "Air", "Telur")
katakA.display_info()
```

Pada contoh di atas, class Hewan, Reptil, dan Amphibi masing-masing memiliki method display_info() yang sama. Kemudian, class Katak diturunkan dari kedua class tersebut, sehingga method display_info() yang ada pada class Reptil dan Amphibi hanya dipanggil sekali dengan menggunakan method super().display_info(). Method display_info() pada class Katak menambahkan informasi nama, umur, dan metamorfosis. Sehingga, ketika method display_info() dari class Katak dipanggil, maka akan menghasilkan informasi jenis, habitat, nama, umur, dan metamorfosis.

Contoh 3:

```
class Orang:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def display_info(self):
        print(f"Nama: {self.nama}")
        print(f"Umur: {self.umur}")

class Pekerja:
    def __init__(self, pekerjaan, gaji):
```

```
        self.pekerjaan = pekerjaan
        self.gaji = gaji

    def display_info(self):
        print(f"Pekerjaan: {self.pekerjaan}")
        print(f"Gaji: {self.gaji}")

class Penulis:
    def __init__(self, buku, genre):
        self.buku = buku
        self.genre = genre

    def display_info(self):
        print(f"Buku: {self.buku}")
        print(f"Genre: {self.genre}")

class PenulisPekerja(Orang, Pekerja, Penulis):
    def __init__(self, nama, umur, pekerjaan, gaji, buku, genre):
        Orang.__init__(self, nama, umur)
        Pekerja.__init__(self, pekerjaan, gaji)
        Penulis.__init__(self, buku, genre)

    def display_info(self):
        super().display_info()
        print(f"Pekerjaan: {self.pekerjaan}")
        print(f"Gaji: {self.gaji}")
        print(f"Buku: {self.buku}")
        print(f"Genre: {self.genre}")

# contoh penggunaan
penulis_pekerjaC = PenulisPekerja("Jane Doe", 30, "Penulis", "$5000", "The
Book", "Fiksi")
penulis_pekerjaC.display_info()
```

Pada contoh di atas, class Orang, Pekerja, dan Penulis masing-masing memiliki method display_info() yang sama. Kemudian, class PenulisPekerja diturunkan dari kedua class tersebut, sehingga method display_info() yang ada pada class Orang, Pekerja, dan Penulis hanya dipanggil sekali dengan menggunakan method super().display_info(). Method display_info() pada class PenulisPekerja menambahkan informasi pekerjaan, gaji, buku, dan genre. Sehingga, ketika method display_info() dari class PenulisPekerja dipanggil, maka akan menghasilkan informasi nama, umur, pekerjaan, gaji, buku, dan genre.

Contoh 4:

```
class Hewan:
    def __init__(self, jenis):
        self.jenis = jenis

    def display_info(self):
```

```
        print(f"Jenis hewan: {self.jenis}")

class Mamalia(Hewan):
    def __init__(self, jenis, nama):
        super().__init__(jenis)
        self.nama = nama

    def display_info(self):
        super().display_info()
        print(f"Nama mamalia: {self.nama}")

class Karnivora(Hewan):
    def __init__(self, jenis, makanan):
        super().__init__(jenis)
        self.makanan = makanan

    def display_info(self):
        super().display_info()
        print(f"Jenis makanan: {self.makanan}")

class Harimau(Mamalia, Karnivora):
    def __init__(self, jenis, nama, makanan):
        Mamalia.__init__(self, jenis, nama)
        Karnivora.__init__(self, jenis, makanan)

    def display_info(self):
        super().display_info()
        print(f"Jenis hewan: {self.jenis}")

# contoh penggunaan
harimauA = Harimau("Mamalia", "Harimau Sumatera", "Daging")
harimauA.display_info()
```

Pada contoh di atas, class Mamalia dan Karnivora masing-masing memiliki method display_info() yang sama dengan class Hewan. Namun, pada saat class Harimau diturunkan dari kedua class tersebut, maka method display_info() yang ada pada class Hewan hanya dipanggil sekali dengan menggunakan method super().display_info() dari class Mamalia dan Karnivora. Kemudian, method display_info() dari class Harimau menambahkan informasi jenis hewan. Sehingga, ketika method display_info() dari class Harimau dipanggil, maka akan menghasilkan informasi jenis hewan, nama mamalia, jenis makanan, dan jenis hewan kembali.

Contoh 5:

```
class Person:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur
```

```
    def display_info(self):
        print(f"Nama: {self.nama}")
        print(f"Umur: {self.umur}")

class Mahasiswa(Person):
    def __init__(self, nama, umur, jurusan):
        super().__init__(nama, umur)
        self.jurusan = jurusan

    def display_info(self):
        super().display_info()
        print(f"Jurusan: {self.jurusan}")

class Alumni(Person):
    def __init__(self, nama, umur, tahun_lulus):
        super().__init__(nama, umur)
        self.tahun_lulus = tahun_lulus

    def display_info(self):
        super().display_info()
        print(f"Tahun lulus: {self.tahun_lulus}")

class MahasiswaAlumni(Mahasiswa, Alumni):
    def __init__(self, nama, umur, jurusan, tahun_lulus):
        Mahasiswa.__init__(self, nama, umur, jurusan)
        Alumni.__init__(self, nama, umur, tahun_lulus)

    def display_info(self):
        super().display_info()
        print(f"Tahun lulus: {self.tahun_lulus}")

# contoh penggunaan
mahasiswa_alumniA = MahasiswaAlumni("Budi", 20, "Informatika", 2022)
mahasiswa_alumniA.display_info()
```

Pada contoh di atas, class Mahasiswa dan Alumni masing-masing memiliki method display_info() yang sama dengan class Person. Namun, pada saat class MahasiswaAlumni diturunkan dari kedua class tersebut, maka method display_info() yang ada pada class Person hanya dipanggil sekali dengan menggunakan method super().display_info() dari class Mahasiswa dan Alumni. Kemudian, method display_info() dari class MahasiswaAlumni menambahkan informasi tahun lulus. Sehingga, ketika method display_info() dari class MahasiswaAlumni dipanggil, maka akan menghasilkan informasi nama, umur, jurusan, dan tahun lulus.

3. Hierarchical inheritance

Hierarchical Inheritance adalah salah satu tipe pewarisan kelas (inheritance) dalam bahasa pemrograman Python. Konsep dasarnya adalah untuk membuat sebuah kelas turunan (child class) yang mewarisi (inherit) sifat-sifat (atribut dan method) dari kelas induk (parent class), namun dengan satu atau lebih kelas turunan yang berasal dari kelas induk yang sama.

Dalam Hierarchical Inheritance, sebuah kelas turunan dapat memiliki satu atau lebih kelas turunan lainnya yang juga berasal dari kelas induk yang sama. Ini berarti bahwa setiap kelas turunan memiliki semua sifat yang diturunkan dari kelas induk dan juga dapat menambahkan sifat-sifat tambahan yang spesifik untuk kelas turunan tersebut.

Contoh 1:

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

    def get_name(self):
        return self.name

    def get_color(self):
        return self.color

class Mammal(Animal):
    def __init__(self, name, color, fur):
        super().__init__(name, color)
        self.fur = fur

    def get_fur(self):
        return self.fur

class Bird(Animal):
    def __init__(self, name, color, wingspan):
        super().__init__(name, color)
        self.wingspan = wingspan

    def get_wingspan(self):
        return self.wingspan

# Hierarchical Inheritance
class Reptile(Mammal):
    def __init__(self, name, color, fur, scale):
        super().__init__(name, color, fur)
        self.scale = scale

    def get_scale(self):
        return self.scale
```

Dalam contoh ini, kita memiliki kelas induk "Animal", dengan dua kelas turunan "Mammal" dan "Bird". Kemudian, kita memiliki turunan Hierarchical Inheritance "Reptile", yang mewarisi dari "Mammal". "Reptile" memiliki sifat tambahan "scale", yang tidak ada pada "Animal" dan "Bird".

Contoh 2:

```
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age

    def get_salary(self):
        return self.salary

class Manager(Employee):
    def __init__(self, name, age, salary, department):
        super().__init__(name, age, salary)
        self.department = department

    def get_department(self):
        return self.department

class Programmer(Employee):
    def __init__(self, name, age, salary, language):
        super().__init__(name, age, salary)
        self.language = language

    def get_language(self):
        return self.language

# Hierarchical Inheritance
class SeniorProgrammer(Programmer):
    def __init__(self, name, age, salary, language, level):
        super().__init__(name, age, salary, language)
        self.level = level

    def get_level(self):
        return self.level
```

Dalam contoh ini, kita memiliki kelas induk "Employee", dengan dua kelas turunan "Manager" dan "Programmer". Kemudian, kita memiliki turunan Hierarchical Inheritance "SeniorProgrammer", yang mewarisi dari "Programmer". "SeniorProgrammer" memiliki sifat tambahan "level", yang tidak ada pada "Employee", "Manager", dan "Programmer".

Contoh 3:

```
class Kendaraan:
    def __init__(self, nama):
        self.nama = nama

    def get_nama(self):
        return self.nama

class Mobil(Kendaraan):
    def __init__(self, nama, merek):
        super().__init__(nama)
        self.merek = merek

    def get_merek(self):
        return self.merek

class SepedaMotor(Kendaraan):
    def __init__(self, nama, tipe):
        super().__init__(nama)
        self.tipe = tipe

    def get_tipe(self):
        return self.tipe

# turunan Hierarchical Inheritance
class Truk(Mobil):
    def __init__(self, nama, merek, kapasitas):
        super().__init__(nama, merek)
        self.kapasitas = kapasitas

    def get_kapasitas(self):
        return self.kapasitas

# turunan Hierarchical Inheritance
class MotorListrik(SepedaMotor):
    def __init__(self, nama, tipe, daya):
        super().__init__(nama, tipe)
        self.daya = daya

    def get_daya(self):
        return self.daya
```

Dalam contoh kode di atas, kita memiliki kelas induk Kendaraan, yang memiliki dua kelas turunan: Mobil dan SepedaMotor. Setiap kelas turunan memiliki sifat-sifat khususnya sendiri (merek dan tipe), namun juga mewarisi sifat-sifat dari kelas induk, seperti nama.

Kita juga memiliki dua turunan Hierarchical Inheritance lainnya: Truk, yang mewarisi dari Mobil, dan MotorListrik, yang mewarisi dari SepedaMotor. Dalam contoh ini, setiap kelas turunan memiliki sifat-sifat tambahan yang spesifik untuk masing-masing kelas. Truk memiliki sifat kapasitas, sementara MotorListrik memiliki sifat daya.

Contoh 4:

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

    def get_name(self):
        return self.name

    def get_color(self):
        return self.color

class TwoDimensional(Shape):
    def __init__(self, name, color, sides):
        super().__init__(name, color)
        self.sides = sides

    def get_sides(self):
        return self.sides

class ThreeDimensional(Shape):
    def __init__(self, name, color, faces):
        super().__init__(name, color)
        self.faces = faces

    def get_faces(self):
        return self.faces

# Hierarchical Inheritance
class Sphere(ThreeDimensional):
    def __init__(self, name, color, faces, radius):
        super().__init__(name, color, faces)
        self.radius = radius

    def get_radius(self):
        return self.radius
```

Dalam contoh ini, kita memiliki kelas induk "Shape", dengan dua kelas turunan "TwoDimensional" dan "ThreeDimensional". Kemudian, kita memiliki turunan Hierarchical

Inheritance "Sphere", yang mewarisi dari "ThreeDimensional". "Sphere" memiliki sifat tambahan "radius", yang tidak ada pada "Shape", "TwoDimensional", dan "ThreeDimensional".

Contoh 5:

```
class AkunBank:
    def __init__(self, nomor_akun, saldo):
        self.nomor_akun = nomor_akun
        self.saldo = saldo

    def get_nomor_akun(self):
        return self.nomor_akun

    def get_saldo(self):
        return self.saldo

class AkunTabungan(AkunBank):
    def __init__(self, nomor_akun, saldo, persentase_bunga):
        super().__init__(nomor_akun, saldo)
        self.persentase_bunga = persentase_bunga

    def get_persentase_bunga(self):
        return self.persentase_bunga

class CekAkun(AkunBank):
    def __init__(self, nomor_akun, saldo, overdraft_limit):
        super().__init__(nomor_akun, saldo)
        self.overdraft_limit = overdraft_limit

    def get_overdraft_limit(self):
        return self.overdraft_limit

# Hierarchical Inheritance
class JointAccount(AkunTabungan):
    def __init__(self, nomor_akun, saldo, persentase_bunga, owners):
        super().__init__(nomor_akun, saldo, persentase_bunga)
        self.owners = owners

    def get_owners(self):
        return self.owners
```

Dalam contoh ini, kita memiliki kelas induk "AkunBank", dengan dua kelas turunan "AkunTabungan" dan "CekAkun". Kemudian, kita memiliki turunan Hierarchical Inheritance "JointAccount", yang mewarisi dari "AkunTabungan". "JointAccount" memiliki sifat tambahan "owners", yang tidak ada pada "AkunBank", "AkunTabungan", dan "CekAkun".

4. Multi-level Inheritance (Pewarisan Bertingkat)

Multi-level Inheritance di dalam Python merujuk pada suatu konsep di mana sebuah kelas mewarisi sifat dan metode dari kelas induk (parent class) dan kelas induk itu sendiri merupakan turunan dari kelas yang lebih tinggi lagi. Dalam hal ini, kita bisa membayangkan struktur kelas sebagai sebuah piramida, dengan kelas yang lebih tinggi berada di puncak dan kelas yang lebih rendah di bawahnya.

Contoh 1:

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

    def speak(self):
        print("The animal speaks")

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def speak(self):
        print("The dog barks")

class Bulldog(Dog):
    def __init__(self, name, breed, origin):
        super().__init__(name, breed)
        self.origin = origin

    def speak(self):
        print("The bulldog growls")
```

Pada contoh di atas, kelas Animal adalah kelas induk (parent class) yang memiliki atribut dan metode dasar yang berkaitan dengan hewan. Kelas Dog mewarisi sifat dan metode dari kelas Animal, serta menambahkan atribut tambahan seperti ras (breed). Terakhir, kelas Bulldog mewarisi sifat dan metode dari kelas Dog, dan menambahkan atribut lain seperti asal (origin).

Dalam kasus ini, ketika kita membuat objek dari kelas Bulldog dan memanggil metode speak(), maka output yang dihasilkan adalah "The bulldog growls", karena metode speak() pada kelas Bulldog akan mengambil prioritas paling tinggi dalam urutan pewarisan.

Dengan konsep Multi-level Inheritance, kita bisa membuat struktur kelas yang lebih kompleks dan memungkinkan kita untuk membagi kelas-kelas dalam hirarki yang lebih terorganisir. Namun, perlu diingat bahwa terlalu banyak tingkat pewarisan bisa membuat kode menjadi sulit dibaca dan dipahami, sehingga disarankan untuk tidak terlalu banyak menggunakan pewarisan dalam struktur kelas.

Contoh 2:

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

    def get_details(self):
        print(f"Name: {self.name}, Age: {self.age}")

class Employee(Person):
    def __init__(self, name, age, id, salary):
        super().__init__(name, age)
        self.id = id
        self.salary = salary

    def get_details(self):
        super().get_details()
        print(f"ID: {self.id}, Salary: {self.salary}")

class Manager(Employee):
    def __init__(self, name, age, id, salary, department):
        super().__init__(name, age, id, salary)
        self.department = department

    def get_details(self):
        super().get_details()
        print(f"Department: {self.department}")
```

Pada contoh di atas, kelas Person adalah kelas induk (parent class) yang memiliki atribut dan metode dasar yang berkaitan dengan seseorang. Kelas Employee mewarisi sifat dan metode dari kelas Person, serta menambahkan atribut tambahan seperti id dan salary. Terakhir, kelas Manager mewarisi sifat dan metode dari kelas Employee, dan menambahkan atribut lain seperti department.

Contoh 3:

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

    def speak(self):
        print(f"{self.name} speaks")

class Bird(Animal):
    def __init__(self, name, wingspan):
        super().__init__(name)
        self.wingspan = wingspan
```

```
    def fly(self):
        print(f"{self.name} is flying with a wingspan of {self.wingspan}")

class Parrot(Bird):
    def __init__(self, name, wingspan, color):
        super().__init__(name, wingspan)
        self.color = color

    def speak(self):
        print(f"{self.name} is a {self.color} parrot that talks")

parrot = Parrot("Rio", 12, "blue")
parrot.speak() # Output: Rio is a blue parrot that talks
parrot.fly()   # Output: Rio is flying with a wingspan of 12
```

Pada contoh di atas, kelas Animal adalah kelas induk (parent class) yang memiliki metode speak() yang menghasilkan output berupa suara hewan. Kelas Bird mewarisi sifat dan metode dari kelas Animal, serta menambahkan atribut tambahan seperti wingspan dan metode fly() yang menghasilkan output berupa burung terbang dengan lebar sayap tertentu. Terakhir, kelas Parrot mewarisi sifat dan metode dari kelas Bird, dan menambahkan atribut lain seperti color dan metode speak() yang menghasilkan output berupa suara yang diucapkan oleh burung Parrot dengan warna tertentu.

Dalam contoh ini, objek parrot adalah sebuah instance dari kelas Parrot yang memanggil metode speak() dan fly() yang diwarisi dari kelas Bird dan Animal. Dengan demikian, parrot bisa memanggil metode fly() untuk terbang dan metode speak() untuk berbicara dengan warna spesifik.

5. Hybrid Inheritance

Hybrid Inheritance merupakan jenis pewarisan atau inheritance yang terjadi ketika suatu kelas mewarisi dari dua atau lebih kelas induk atau parent class. Dalam Hybrid Inheritance, kelas anak dapat mewarisi sifat dan metode dari kelas induk yang diwarisi secara langsung atau tidak langsung.

Dalam Python, Hybrid Inheritance dapat dicapai dengan menggabungkan dua atau lebih jenis pewarisan, yaitu **Multiple Inheritance** dan **Single Inheritance**. Dalam Multiple Inheritance, suatu kelas mewarisi sifat dan metode dari dua atau lebih kelas induk secara langsung. Sedangkan dalam Hierarchical Inheritance, suatu kelas induk memiliki dua atau lebih kelas anak.

Sebagai contoh, kita dapat membuat kelas A dan B sebagai kelas induk atau parent class, dan kelas C dan D sebagai kelas anak atau child class yang mewarisi dari kelas A dan B, seperti berikut:

Contoh 1:

```python
# Single Inheritance
class GameObject:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Single Inheritance
class Drawable:
    def draw(self):
        print("Drawing object at: ", self.x, self.y)

# Single Inheritance
class Moveable:
    def move(self, dx, dy):
        self.x += dx
        self.y += dy

# Multiple Inheritance
class Player(GameObject, Drawable, Moveable):
    def __init__(self, x, y):
        super().__init__(x, y)

    def update(self):
        self.move(1, 1)
        self.draw()
```

Contoh 2:

```python
class Seseorang:
    def __init__(self, name, age, address):
        self.name = name
        self.age = age
        self.address = address

    def get_info(self):
        print("Name:", self.name)
        print("Age:", self.age)
        print("Address:", self.address)

# Single Inheritance
class Mahasiswa(Seseorang):
    def __init__(self, name, age, address, student_id):
        super().__init__(name, age, address)
        self.student_id = student_id

    def get_info(self):
        super().get_info()
        print("Student ID:", self.student_id)
```

```
# Single Inheritance
class Employee(Seseorang):
    def __init__(self, name, age, address, employee_id, salary):
        super().__init__(name, age, address)
        self.employee_id = employee_id
        self.salary = salary

    def get_info(self):
        super().get_info()
        print("Employee ID:", self.employee_id)
        print("Salary:", self.salary)

# Multiple Inheritance
class Penulis(Employee, Mahasiswa):
    def __init__(self, name, age, address, employee_id, salary, student_id,
published_books):
        Employee.__init__(self, name, age, address, employee_id, salary)
        Mahasiswa.__init__(self, name, age, address, student_id)
        self.published_books = published_books

    def get_info(self):
        super().get_info()
        print("Student ID:", self.student_id)
        print("Published Books:", self.published_books)
```

Pada contoh di atas, kelas Seseorang merupakan kelas induk atau super class yang memiliki atribut name, age, dan address, serta metode get_info untuk mencetak informasi diri seseorang. Kelas Mahasiswa dan Employee merupakan kelas turunan atau sub class dari Seseorang. Kelas Mahasiswa memiliki tambahan atribut student_id, sementara kelas Employee memiliki tambahan atribut employee_id dan salary.

Kelas Penulis mewarisi sifat dan perilaku dari kedua kelas turunan tersebut menggunakan multiple inheritance. Kelas Penulis memiliki tambahan atribut published_books. Dalam konstruktor, kelas Penulis memanggil konstruktor dari kedua kelas turunan menggunakan fungsi super() dan menyediakan argumen yang sesuai.

Metode get_info pada kelas Penulis menggunakan metode get_info dari kelas Employee dan Mahasiswa menggunakan fungsi super(). Sehingga, metode get_info pada kelas Penulis mencetak informasi lengkap mengenai seorang penulis, termasuk informasi mengenai mahasiswa dan pegawai.

Praktikum

1. Buatlah masing-masing 2 contoh jenis pewarisan di luar dari contoh yang telah diberikan, beri nama:

single1.py, single2.py,

multiple1.py, multiple2.py,

hierarchical1.py, hierarchical2.py,

multilevel1.py, multilevel2,

hybrid1.py, hybrid2.py

Tugas: Buatlah design tabel sebuah aplikasi database yang menerapkan Inheritance