LAPORAN RESMI

MODUL III

INHERITANCE & OVERRIDING

PEMROGRAMAN BERBASIS OBJEK

NAMA	**: ADYTTA PUTRA TARIGAN**
N.R.P	**: 240441100139**
DOSEN	**: YUDHA DWI PUTRA NEGARA, S.KOM., M.KOM.**
ASISTEN	**: AHMAD RIKHAN ARBA'I**
TGL PRAKTIKUM	**: 26 APRIL 2025**

Disetujui : 1 MEI 2025
Asisten

AHMAD RIKHAN ARBA'I
23.04.411.00192

LABORATORIUM TEKNOLOGI INFORMASI

PRODI SISTEM INFORMASI

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS TRUNOJOYO MADURA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Inheritance atau pewarisan dalam pemrograman berorientasi objek adalah konsep di mana sebuah kelas (anak) bisa mewarisi properti dan metode dari kelas lain (induk). Tujuannya adalah untuk menghindari penulisan ulang kode yang sama, sehingga kode menjadi lebih efisien dan terstruktur. Misalnya, kalau kita punya kelas Hewan yang punya metode bergerak(), maka kelas Kucing atau Burung bisa mewarisi metode tersebut tanpa perlu menulis ulang, karena pada dasarnya mereka juga termasuk hewan yang bisa bergerak.

Namun dalam praktiknya, tidak semua metode dari kelas induk selalu cocok 100% untuk kelas anak. Di sinilah konsep *Overriding* menjadi penting. *Overriding* berarti kelas anak menulis ulang (menimpa) metode yang diwarisi dari kelas induk supaya lebih sesuai dengan kebutuhan spesifik. Misalnya, metode bergerak() di kelas Burung bisa ditimpa agar mencerminkan bahwa burung bergerak dengan cara terbang, bukan berjalan seperti kucing. Ini memberi fleksibilitas dan kendali lebih besar terhadap perilaku masing-masing objek.

Dengan kombinasi *inheritance* dan *overriding*, kita bisa membuat sistem program yang rapi, modular, dan mudah diperluas. Bayangkan jika setiap kelas harus ditulis dari nol tanpa bisa mewarisi apa pun—hal itu akan membuat program cepat menjadi rumit dan susah dipelihara. *Inheritance* menghemat waktu dan memperjelas hubungan antar objek, sedangkan *overriding* memberikan keleluasaan dalam menyesuaikan perilaku sesuai konteks masing-masing. Keduanya adalah fondasi penting dalam membuat program yang cerdas dan adaptif.

1.2 Tujuan

- Mahasiswa mampu memahami konsep *Inheritance* dan *Overriding* dalam Pemrograman Berorientasi Objek serta mampu mengimplementasikannya.
- Mahasiswa mampu memahami Mengidentifikasi struktur hubungan antara *superclass* (*parent class*) dan subclass (*child class*).

BAB II
DASAR TEORI

2.1 Pengertian Inheritance

Inheritance adalah salah satu konsep fundamental dalam Pemrograman berorientasi Objek. Konsep ini memungkinkan sebuah kelas (class) untuk mewarisi properti (atribut) dan perilaku (metode) dari kelas lain. Dengan kata lain, *inheritance* memungkinkan kita membuat kelas baru yang merupakan turunan dari kelas yang sudah ada, sehingga kelas baru tersebut dapat menggunakan kembali kode yang sudah ada tanpa harus menulis ulang.

2.1.1 Mengapa Inheritance Penting

A. Menghemat Waktu dan Tenaga (Reusabilitas Kode)

   Dengan *inheritance*, kita bisa menggunakan kembali kode yang sudah dibuat di kelas sebelumnya. Jadi, kita tidak perlu menulis ulang kode yang sama berulang-ulang. Ini membuat pekerjaan kita jadi lebih cepat dan efisien.

B. Membuat Kode Lebih Rapi dan Teratur (Organisasi Kode)

   Dengan *inheritance*, kita bisa mengelompokkan kode berdasarkan hubungan yang mirip seperti di dunia nyata. Misalnya, semua hewan punya sifat dasar yang sama, jadi kita buat satu kelas hewan, lalu hewan-hewan spesifik seperti anjing atau kucing dibuat sebagai turunan dari kelas hewan. Ini membuat kode kita lebih mudah dimengerti dan diatur.

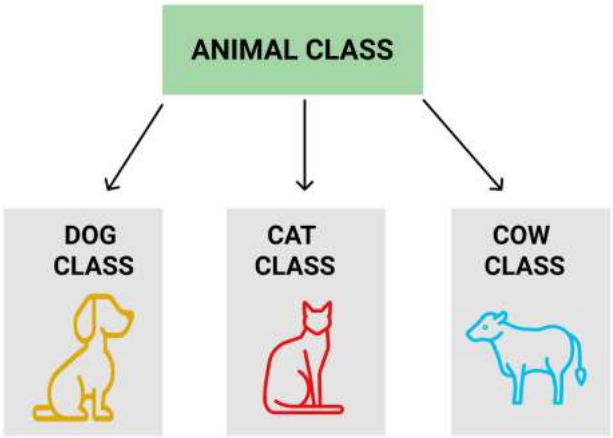
C. Mempermudah Perubahan dan Perbaikan (Pemeliharaan Kode)

   Kalau kita ingin mengubah sesuatu yang umum, kita cukup ubah di kelas induk saja. Perubahan ini otomatis akan berlaku ke semua kelas turunan yang mewarisinya. Jadi, kita tidak perlu mengubah kode di banyak tempat sekaligus.

D. Membuat Program Lebih Fleksibel (*Polimorfisme*)

   Dengan *inheritance*, kita bisa menggunakan objek dari kelas turunan seolah-olah itu objek dari kelas induk. Ini membantu kita membuat program yang lebih fleksibel dan mudah dikembangkan, karena bisa

menggunakan berbagai jenis objek yang berbeda tapi dengan cara yang sama.



Penjelasan :

A. ***Animal Class*:** Merupakan kelas induk (*parent class*).yang dapat berisikan method yang akan diwariskan.
B. ***Dog Class, Cat Class, Cow Class*:** Merupakan kelas anak (*child class*) yang mewarisi atribut dan method dari Animal Class.

Contoh Kode:

A. ***Parent Class***

```
class Animal:
    def __init__(self, name):
        self.name = name
        self.health = 100

    def Makan (self):
        print(f"{self.name} Sedang Makan")

    def Jalan (self):
        print(f"{self.name} Sedang Jalan")
```

Class *Animal* merupakan *parent class* (kelas induk) yang berfungsi sebagai dasar bagi kelas-kelas lain yang akan dibuat sebagai child class (kelas turunan). Class ini memiliki dua method (fungsi) yaitu Makan dan Jalan. yang mendefinisikan perilaku umum yang dapat dilakukan oleh semua objek hewan.

B. Child Class

```
class Dog (Animal):
    pass
```

Class *Dog* merupakan child class atau kelas anak dari class *Animal*. Artinya, *Dog* mewarisi semua atribut dan method yang sudah didefinisikan di class *Animal*.

- **Cara Memanggil**

```
dog = Dog("Blacky")
print(dog.name)
print(dog.health)
dog.Makan()
dog.Jalan()
```

- **Hasil**

```
Blacky
100
Blacky Sedang Makan
Blacky Sedang Jalan
```

2.2 Pengertian *Overriding*

Overriding adalah proses di mana sebuah *child class* (kelas anak) menulis ulang atau mengganti implementasi sebuah metode yang diwarisi dari *parent class* (kelas induk). Tujuannya adalah agar child class dapat memiliki perilaku atau fungsi yang berbeda dan lebih spesifik sesuai dengan kebutuhan atau karakteristiknya sendiri, meskipun metode tersebut sudah ada di kelas induk.

Contoh Kode:

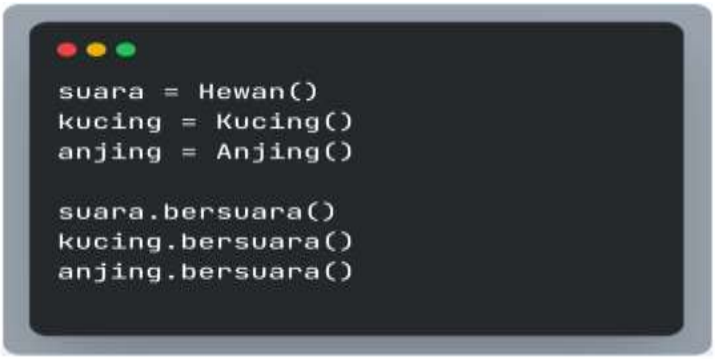
A. Parent Class

Class Hewan adalah *parent class* (kelas induk) yang memiliki method bersuara.

- **Overriding**

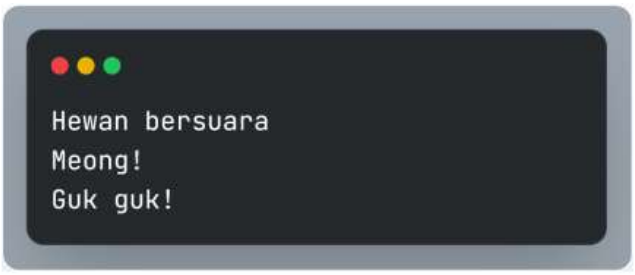
Class Kucing dan anjing merupakan *child class* dari kelas Hewan. Di sini, kita menulis ulang (*override*) method bersuara() yang ada di kelas Hewan. Jadi, meskipun Hewan punya method bersuara() yang menampilkan "Hewan bersuara", kelas Kucing menggantinya dengan method baru yang menampilkan "Meong!".Ini artinya, saat kita memanggil bersuara() pada objek Kucing, yang keluar bukan lagi "Hewan bersuara", tapi "Meong!". ini berlaku juga di Class Anjing.

B. Cara Memanggil



Untuk pemanggilannya kita membuat objek yang memanggil nama class setelah membuat objek kita memanggil methodnya dengan memanggil objek titik lalu nama method yang akan dijalankan.

C. Hasil



Nah, yang awalnya method suara yang berisikan “Hewan Bersuara” setelah di *overriding* hasilnya berubah menjadi “Meong!” dan “Guk guk!”.

2.3 Pengertian Super()

super() adalah fungsi bawaan yang digunakan untuk mengakses method atau constructor dari kelas induk (*parent class*) secara langsung, tanpa menyebutkan nama kelas induk tersebut secara eksplisit. Fungsi ini biasanya digunakan di dalam kelas turunan untuk melanjutkan atau memperluas perilaku yang telah didefinisikan oleh kelas induk. Salah satu penggunaan utama super() adalah untuk memanggil constructor induk (__init__) dari dalam constructor anak, agar atribut atau logika inisialisasi yang ditetapkan oleh kelas induk tetap dijalankan.

BAB III
TUGAS PENDAHULUAN

3.1 Soal

1. Buatlah class person dengan atribut name and age, serta sebuah method Introduce() yang mencetak perkenalan diri. Buat subkelas Student dan Teacher yang mewarisi Class person dan menambahkan atribut khusus. Implementasikan method Introduce() disetiap subkelas dan tampilkan hasilnya
2. Buat class Animal dengan method speak(). kemudian buat dua subkelas Dog dan Cat yang mewarisi class animal. Implementasikan method speak() disetiap subkelas untuk menghasilkan suara yang sesuai.
3. Implementasikan pewarisan bertingkat dengan class Shape. Rectangle. dan square. Class Rectangle mewarisi Shape dan menambahkan atribut width dan height. Class square mewarisi Rectangle dan memastikan bahwa panjang sisi sama. Implementasikan method untuk menghitung luas dimasing - masing class.
4. Buat class employee dengan atribut name and salary, dan buat dua subkelas manager dan developer. Tambahkan atribut tambahan pada setiap subkelas dan tampilkan informasi menggunakan method display_INFO() yang dioverride.
5. Buat class vehicle dengan method fuel_eFFiciency(). kemudian buat dua subkelas car dan truk yang mewarisi class vehicle. Implementasikan method fuel_efficiency() untuk mencetak informasi efesiensi bahan bakar pada setiap kendaraan.
6. Apa itu konsep pewarisan (inheritance) dalam oop? Jelaskan dan berikan contoh singkat dengan dua class. Satu kelas induk dan satu kelas turunan.
7. Jelaskan apa itu "method overriding" dalam inheritance dan buat contoh sederhana.
8. Apa yang dimaksud dengan "super()" dalam inheritance dan bagaimana cara kerjanya? Jelaskan dengan contoh.
9. Jelaskan perbedaan antara "Single Inheritance" dan "Multiple Inheritance" dengan contoh.
10. Apa yang dimaksud dengan "Constructor" dalam oop? Bagaimana cara menggunakan Constructor dalam Inheritance?

3.2 Jawaban

1.
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age  = age

    def introduce(self):
        print(f"Halo, saya {self.name}. Umur saya {self.age} tahun.")

class Student(person):
    def __init__(self, name, age, major):
        super().__init__(name, age)
        self.major = major

    def introduce(self):
        print(f"Halo, saya {self.name}, umur {self.age} tahun, dan
              saya mahasiswa Jurusan {self.major}.")

class Teacher(person):
    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject

    def introduce(self):
        print(f"Selamat datang, Saya {self.name}, umur {self.age}
              tahun, dan saya mengajar mapel {self.subject}.")

Person = Person("Budi", 40)
Student = Student("Adytta", 20, "Informatika")
Teacher = teacher("Bu Titir", 40, "Conversation")

Person.introduce()
Student.introduce()
Teacher.introduce()
```

Halo, saya Budi, umur Saya 40 tahun.
Halo, Saya Adytta, umur 20 tahun, dan saya mahasiswa Jurusan Informatika.
Selamat datang, Saya Bu Titir, umur 40 tahun, dan Saya mengajar mapel Conversation.

2. Class Animal:
    def speak(self):
        Print("Animal is making a sound.")

Class Dog(Animal):
    def speak(self):
        Print("Gut Gut")

Class cat(Animal):
    speak(self):
        Print("meong")

animal = Animal()
dog = Dog()
cat = Cat()

animal.speak()
dog.speak()
cat.speak()

3. Class Shape:
    def area(self):
        return 0

Class Rectangle(shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class square(Rectangle):
    def __init__(self, side):
        Super().__init__(side, side)
    def area(self):
        return Super().area()

Shape = Shape()
Rectangle = Rectangle(4,5)
Square = Square(6)

4. Class Employee :

```
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def display_info(self):
        print(f"Nama : {self.name}")
        print(f"Gaji : {self.salary}")
class manager(Employee):
    def __init__(self, name, salary, departement):
        super().__init__(name, salary)
        self.departement = departement
    def display_info(self):
        print(f"=== Info manager ===")
        print(f"Nama : {self.name}")
        print(f"Gaji : Rp {self.salary:,}")
        print(f"Departemen: {self.departement}")
class Developer(Employee):
    def __init__(self, name, salary, programming_language):
        super().__init__(name, salary)
        self.programming_language = programming_language
    def display_info(self):
        print(f"=== Info Developer ===")
        print(f"Nama : {self.name}")
        print(f"Gaji : Rp. {self.salary}")
        print(f"Bahasa pemrograman : {self.programming_language}")
employee = Employee("Rina", 5000000)
manager = Manager("Budi", 10000000, "Marketing")
developer = Developer("Adytia", 80000000, "Python")

employee.display_info()
print()
manager.display_info()
print()
developer.display_info()
```

5. Class Vehicle :

```
    def fuel_efficiency (self):
    Print ("Efisiensi bahan bakar kendaraan tidak diketahui")
Class Car (Vehicle):
    def _init_ (self, km_per_liter)
        self.km_per_liter
    def fuel_efficiency (self):
        Print (f"Mobil ini memiliki efisiensi bahan bakar sekitar
            {self.km_per_liter} km/liter.")
Class Truck (Vehicle):
    def _init_ (self, km_per_liter):
    self.km_per_liter = km_per_liter
    def fuel_efficiency (self):
        Print (f"Truk ini memiliki efesiensi bahan bakar sekitar
            {self.km_per_liter} km/liter.")
Vehicle = Vehicle ()
Car = Car (15)
truck = Truck (8)
Vehicle.fuel_efficiency()
Car.fuel_efficiency()
Truck.fuel_efficiency()
```

6. Pewarisan adalah konsep di OOP di mana sebuah class turunan bisa mengambil atribut dari Class Induk.

```
Class Animal :
    def speak (self):
        Print ("Hewan ini bersuara")
Class Dog (Animal):
    def bark (self):
        Print ("Anjing menggonggong : Guk Guk!")
```

7. Method overriding terjadi saat Class turunan punya method dengan nama yang sama seperti di Class Induknya, tapi dengan isi yang berbeda.

```
Class Animal :
    def speak (self):
        Print ("Hewan ini bersuara.")
Class Cat (Animal):
    def speak (self):
        Print ("kucing: meong")
```

8. Sebuah fungsi built-in di python yang digunakan untuk mengakses method atau atribut dari class induk dalam sebuah class turunan. Cara berjungya, di dalam subclass, kita bisa menggunakan fungsi super() untuk memanggil method di Class induk.

```
Class Animal:
    def __init__ (self, name):
        self.name = name
    def speak (self):
        Print (f"{self.name} bersuara.")
Class Dog (Animal):
    def __init__ (self, name, breed):
        super().__init__ (name)
        self.breed = breed
    def speak (self):
        super().speak()
        Print (f"{self.name} menggonggong: Guk Guk!")
dog = Dog ("Buddy", "Golden Retriever")
dog.speak()
```

9. Single Inheritance hanya mewarisi dari satu Class Induk
multiple Inheritance mewarisi dari lebih dari satu class induk

Contoh Single Inheritance

```
class Animal:
    def speak (self):
        Print ("Hewan ini bersuara.")
Class dog (Animal):
    def bark (self):
```

```
        Print ("Anjing menggonggong: Guk Guk!")
dog = Dog()
dog.speak()
dog.bark()
```

multiple Inheritance

```
Class Animal:
    def speak(self):
        Print("Hewan ini bersuara.")

Class mammal:
    def has_fur(self):
        Print("Hewan ini memiliki bulu")

Class Dog(Animal, mammal):
    def bark(self):
        Print("Anjing menggonggong: Guk Guk")

dog = Dog()
dog.speak()
dog.has_fur()
dog.bark
```

10. Sebuah method khusus di dalam sebuah class yang secara otomatis dipanggil saat sebuah objek dari class tersebut dibuat.

Penggunaan Constructor dalam Inheritance:

1. Class induk memiliki Constructor untuk mengatur atribut
2. Class turunan bisa menggunakan Constructor induk dengan super().

BAB IV
IMPLEMENTASI

4.1 Tugas Praktikum

4.1.1 Tugas Praktikum No. 1

Buatlah class Karyawan dengan atribut :

- nama, gaji, dan departemen
- serta method info() yang mencetak nama, gaji, dan departemen. Buatlah class KaryawanTetap yang mewarisi Karyawan dengan atribut tambahan tunjangan dan method info() yang mencetak informasi karyawan tetap dengan tunjangan.

Buat class KaryawanHarian yang mewarisi Karyawan dengan atribut tambahan jam_kerja (misalnya: jam kerja per hari) dan method info() yang mencetak informasi karyawan harian.

Buat class ManajemenKaryawan yang memiliki atribut :

- daftar_karyawan, yang berupa list yang menyimpan objek KaryawanTetap dan KaryawanHarian.
- Tambahkan method tambah_karyawan() di class ManajemenKaryawan untuk menambahkan objek karyawan ke dalam list daftar_karyawan.
- Tambahkan method tampilkan_semua_karyawan() yang akan menampilkan informasi tentang semua karyawan dengan memanggil method info() untuk masing-masing objek karyawan di dalam list daftar_karyawan.

Buatlah objek dari class ManajemenKaryawan dan tambahkan beberapa karyawan tetap dan harian.

Tampilkan informasi semua karyawan yang ada dalam daftar dengan memanggil method tampilkan_semua_karyawan().

4.1.2 Tugas Praktikum No. 2

Terapkan

Buatlah class Pengiriman dengan:

- Atribut asal dan tujuan.

- Method estimasi_waktu() yang mengembalikan waktu estimasi pengiriman (misal: 5 hari).

Buatlah class PengirimanDarat yang mewarisi Pengiriman, dengan:

- Atribut tambahan jenis_kendaraan.
- Override method estimasi_waktu() berdasarkan jenis kendaraan

Buatlah class PengirimanUdara yang mewarisi Pengiriman, dengan:

- Atribut tambahan maskapai.
- Override method estimasi_waktu() berdasarkan maskapai.

Buatlah class PengirimanInternasional yang mewarisi PengirimanDarat dan PengirimanUdara, dan override estimasi_waktu() untuk menyesuaikan waktu pengiriman internasional berdasarkan asal dan tujuan (misal: jika tujuan luar negeri, tambahkan 3 hari dari estimasi sebelumnya).

Buatlah beberapa objek PengirimanInternasional, set atribut-atributnya, dan panggil method estimasi_waktu() untuk menghitung waktu pengiriman.

4.2 Source Code

4.2.1 Source Code Soal 1

```
class Karyawan:
    def __init__(self, nama, gaji, departemen):
        self.nama = nama
        self.gaji = gaji
        self.departemen = departemen

    def tampilkan_info(self):
        return f"Nama: {self.nama}, Gaji: {self.gaji}, Departemen:
{self.departemen}"

class KaryawanTetap(Karyawan):
    def __init__(self, nama, gaji, departemen, tunjangan):
        super().__init__(nama, gaji, departemen)
```

```
        self.tunjangan = tunjangan

    def tampilkan_info(self):
        return f"{super().tampilkan_info()}, Tunjangan:
{self.tunjangan}"

class KaryawanHarian(Karyawan):
    def __init__(self, nama, gaji, departemen, jam_kerja):
        super().__init__(nama, gaji, departemen)
        self.jam_kerja = jam_kerja

    def tampilkan_info(self):
        return f"{super().tampilkan_info()}, Jam Kerja:
{self.jam_kerja}"

class ManajemenKaryawan:
    def __init__(self):
        self.daftar_karyawan = []

    def tambah_karyawan(self, karyawan):
        self.daftar_karyawan.append(karyawan)

    def tampilkan_semua_karyawan(self):
        for karyawan in self.daftar_karyawan:
            print(karyawan.tampilkan_info())

manajemen = ManajemenKaryawan()

karyawan_tetap_1 = KaryawanTetap("Adytta", 5000000, "HR",
1000000)
karyawan_tetap_2 = KaryawanTetap("Fauzi", 6000000, "Finance",
1200000)
```

```
karyawan_harian_1 = KaryawanHarian("Teddy", 3000000,
"Marketing", 8)
karyawan_harian_2 = KaryawanHarian("Abdul", 3500000, "Sales", 7)


manajemen.tambah_karyawan(karyawan_tetap_1)
manajemen.tambah_karyawan(karyawan_tetap_2)
manajemen.tambah_karyawan(karyawan_harian_1)
manajemen.tambah_karyawan(karyawan_harian_2)


manajemen.tampilkan_semua_karyawan()
```

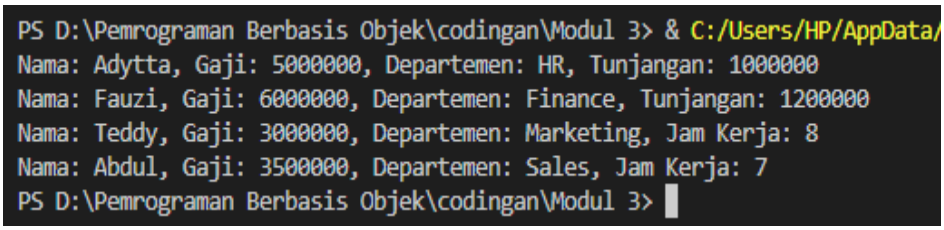
4.2.2 Source Code Soal 2

```
class Pengiriman:
    def __init__(self, asal, tujuan):
        self.asal = asal
        self.tujuan = tujuan

    def estimasi_waktu(self):
        return 6


class PengirimanDarat(Pengiriman):
    def __init__(self, asal, tujuan, jenis_kendaraan):
        super().__init__(asal, tujuan)
        self.jenis_kendaraan = jenis_kendaraan

    def estimasi_waktu(self):
        if self.jenis_kendaraan == "mobil":
            return 5
        elif self.jenis_kendaraan == "truk":
            return 7
        else:
            return 10

class PengirimanUdara(Pengiriman):
    def __init__(self, asal, tujuan, maskapai):
        super().__init__(asal, tujuan)
        self.maskapai = maskapai

    def estimasi_waktu(self):
        if self.maskapai == "Maskapai A":
            return 2
```

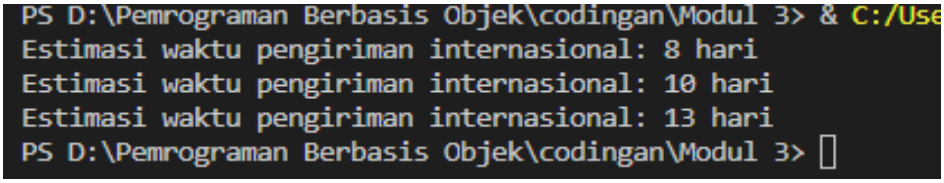
```
        elif self.maskapai == "Maskapai B":
            return 3
        else:
            return 4

class PengirimanInternasional(PengirimanDarat, PengirimanUdara):
    def __init__(self, asal, tujuan, jenis_kendaraan, maskapai):
        Pengiriman.__init__(self, asal, tujuan)
        self.jenis_kendaraan = jenis_kendaraan
        self.maskapai = maskapai

    def estimasi_waktu(self):
        waktu_darat = PengirimanDarat.estimasi_waktu(self)
        waktu_udara = PengirimanUdara.estimasi_waktu(self)
        estimasi_awal = max(waktu_darat, waktu_udara)

        if self.tujuan.lower not in ["jakarta", "bandung", "surabaya",
"medan"]:
            estimasi_awal += 3

        return f"Estimasi waktu pengiriman internasional:
{estimasi_awal} hari"

pengiriman1 = PengirimanInternasional("Jakarta", "Amerika",
"mobil", "Maskapai A")
pengiriman2 = PengirimanInternasional("Bandung", "New York",
"truk", "Maskapai B")
pengiriman3 = PengirimanInternasional("Surabaya", "London",
"motor", "Maskapai C")

print(pengiriman1.estimasi_waktu())
print(pengiriman2.estimasi_waktu())
print(pengiriman3.estimasi_waktu())
```

4.3 Hasil

4.3.1 Hasil Soal 1



4.3.2 Hasil Soal 2



4.4 Penjelasan

4.4.1 Penjelasan Soal 1

Kode yang saya buat, terdapat beberapa kelas yang digunakan untuk mengelola data karyawan. Kelas Karyawan merupakan kelas dasar dengan atribut nama, gaji, dan departemen, serta metode tampilkan_info untuk menampilkan informasi karyawan. Kelas KaryawanTetap dan KaryawanHarian adalah turunan dari Karyawan, dengan penambahan atribut spesifik masing-masing, yaitu tunjangan untuk karyawan tetap dan jam_kerja untuk karyawan harian. Kelas ManajemenKaryawan digunakan untuk mengelola daftar karyawan, dengan metode untuk menambahkan karyawan dan menampilkan seluruh data karyawan. Di akhir kode, beberapa objek karyawan dibuat dan ditambahkan ke dalam manajemen, lalu semua data karyawan ditampilkan menggunakan metode tampilkan_semua_karyawan.

4.4.2 Penjelasan Soal 2

Kode yang saya buat berisi beberapa kelas yang digunakan untuk menghitung estimasi waktu pengiriman barang berdasarkan moda transportasi dan rute pengirimannya. Kelas Pengiriman adalah kelas dasar yang memiliki atribut asal dan tujuan, serta metode estimasi_waktu yang mengembalikan nilai default 6 hari. Kelas PengirimanDarat dan PengirimanUdara adalah turunan dari kelas Pengiriman, masing-masing menghitung estimasi waktu berdasarkan jenis kendaraan darat (seperti mobil dan truk) atau maskapai penerbangan. Kelas PengirimanInternasional merupakan gabungan dari PengirimanDarat dan PengirimanUdara, yang menghitung estimasi waktu pengiriman internasional dengan mempertimbangkan waktu darat dan udara. Jika tujuan pengiriman tidak termasuk kota-kota tertentu (Jakarta, Bandung, Surabaya, Medan), waktu estimasi ditambah 3 hari. Kode ini kemudian membuat objek pengiriman internasional dan mencetak estimasi waktu pengiriman untuk setiap kasus berdasarkan data yang diberikan.

BAB V
PENUTUP

5.1 Analisa

Inheritance dan *Overriding* adalah dua prinsip dasar dalam pemrograman berorientasi objek yang sangat membantu dalam merancang sistem yang efisien dan terstruktur. *Inheritance* memungkinkan sebuah kelas untuk mewarisi atribut dan metode dari kelas induk, yang mengurangi *redundansi* dan memfasilitasi penggunaan kembali kode. Hal ini juga memungkinkan pengembangan kode yang lebih modular, di mana fitur-fitur dasar dapat ditangani dalam kelas induk, dan kelas turunan dapat menambahkan atau memodifikasi perilaku sesuai kebutuhan. Di sisi lain, *Overriding* memberikan fleksibilitas bagi kelas turunan untuk mengganti implementasi metode yang diwarisi dari kelas induk dengan versi yang lebih spesifik dan sesuai konteks.

5.2 Kesimpulan

Inheritance dan *Overriding* adalah dua konsep inti dalam pemrograman berorientasi objek yang memungkinkan pengembangan kode yang lebih efisien dan terstruktur. *Inheritance* memungkinkan penggunaan kembali kode dengan mewarisi atribut dan metode dari kelas induk, sementara *Overriding* memberikan fleksibilitas untuk mengganti implementasi metode di kelas turunan agar lebih spesifik sesuai kebutuhan. Kombinasi keduanya membuat sistem menjadi lebih modular, fleksibel, dan mudah untuk dikembangkan atau disesuaikan, tanpa perlu menulis ulang kode yang sudah ada.

1. *Inheritance* mengurangi duplikasi kode dengan mewarisi atribut dan metode dari kelas induk, serta memudahkan pemeliharaan kode.
2. *Overriding* memberi kesempatan pada kelas turunan untuk mengganti implementasi metode dari kelas induk, menyesuaikan perilaku sesuai kebutuhan spesifik.
3. Kombinasi *Inheritance* dan *Overriding* menciptakan sistem yang lebih modular, fleksibel, dan mudah dikembangkan tanpa menulis ulang fungsionalitas yang telah ada.