LAPORAN RESMI

MODUL V

ABSTRACT CLASS &ABSTRACR

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	**: 17 MEI 2025**

Disetujui : 28 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

Dalam pemrograman berorientasi objek, konsep abstrak *class* dan abstrak muncul sebagai solusi untuk menciptakan kerangka dasar dari sebuah sistem yang kompleks. Abstrak *class* merupakan *class* yang tidak bisa diinstansiasi secara langsung, namun dirancang untuk menjadi fondasi bagi *class-class* turunannya. Dengan menyediakan struktur umum—seperti atribut dan yang dapat diwariskan—abstrak *class* membantu pengembang menyusun *blueprint* yang konsisten dalam sebuah hierarki objek. Konsep ini sangat berguna dalam sistem besar yang memiliki banyak entitas serupa namun memiliki perilaku yang sedikit berbeda.

Sementara itu, abstrak adalah yang dideklarasikan di dalam abstrak *class* namun tidak memiliki implementasi langsung. Tujuannya adalah untuk memaksa *class* turunan memberikan implementasi spesifik terhadap tersebut. Dengan adanya abstrak , *programmer* dapat menjamin bahwa semua *subclass* akan memiliki fungsi-fungsi penting tertentu yang telah didefinisikan secara abstrak di level atas. Hal ini memberikan kontrol dan konsistensi dalam desain sistem, sekaligus fleksibilitas dalam implementasi. Oleh karena itu, penggunaan abstrak dan abstrak *class* menjadi elemen penting dalam membangun sistem perangkat lunak yang modular, skalabel, dan mudah dikembangkan ke depannya.

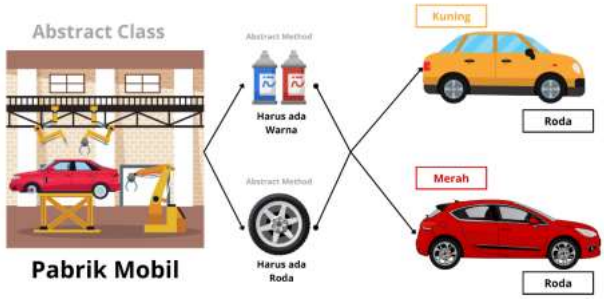
1.2 Tujuan

- Mahasiswa mampu memahami konsep *abstract class* dan *abstract* dalam OOP.
- Mahasiswa dapat mengimplementasikan *abstract class* menggunakan modul abc di Python.
- Mahasiswa dapat menerapkan konsep tersebut dalam studi kasus sederhana.

BAB II
DASAR TEORI

2.1 *ABSTRACT CLASS*

Abstract Class, merupakan suatu *Class* yang dideklarasikan sebagai *abstract*, dan merupakan kelas yang berisi satu atau lebih perilaku yang diabstraksi. Yang berarti bahwa objek atau kelas tersebut dapat diringkas menjadi karakteristik yang relevan. Cara kerja *abstract class* itu ketika di-*subclass*-kan, *subclass* tersebut biasanya menyediakan implementasi untuk semua metode *abstract* di *parent class*. Namun, jika tidak, maka *subclass*-nya juga harus dideklarasikan *abstract*..



Analogi sederhananya seperti, semua kendaraan harus bisa "bergerak", tapi cara bergeraknya bisa berbeda-beda tergantung jenisnya. *Blueprint* ini tidak bisa digunakan langsung (tidak bisa dibuat kendaraan dari *blueprint* saja), tapi digunakan sebagai dasar untuk membuat mobil, motor, atau sepeda yang masing-masing memiliki cara bergerak tersendiri.

2.1.1 Ciri – Ciri dan Tujuan *Abstract Class* pada Python

a. *Abstract Class* di Python mewarisi kelas ABC dari modul abc

b. Tidak dapat membuat objek dari *abstract class*. Jika dicoba, akan menghasilkan *error*.

Code:

```python
from abc import ABC, abstractmethod

class Kendaraan(ABC):

    @abstractmethod
    def makan(self):
        pass

objek = Kendaraan()
```

Output:

```
  File "d:\semester 4\NGAJAR 2025\PBO\modul5\contonh.py", line 9, in <module>
    objek = Kendaraan()
TypeError: Can't instantiate abstract class Kendaraan without an implementation for abstract method 'makan'
```

c. Digunakan untuk mendefinisikan struktur dasar yang akan diturunkan ke *subclass*.

d. *Abstract class* bisa berisi kombinasi antara biasa (dengan isi) dan *abstract* (tanpa isi).

```python
from abc import ABC, abstractmethod

class Kendaraan(ABC):

    def nyalakan_lampu(self):
        print("Lampu dinyalakan.")  # Ini method biasa

    @abstractmethod
    def jalan(self):
        pass  # Ini harus diimplementasi di subclass

```

e. Memastikan semua turunan memiliki struktur dan yang sama meskipun implementasinya berbeda.

2.2 *Abstract*

Abstract adalah (fungsi dalam *class*) yang hanya dideklarasikan tetapi tidak memiliki isi (implementasi) di dalam *class* induknya. ini dibuat menggunakan *decorator* `@abstract`, dan hanya bisa digunakan dalam *abstract class*. Tujuannya adalah untuk memaksa setiap *subclass* agar mengisi atau mengimplementasikan sendiri isi dari tersebut. Dengan begitu, *abstract* membantu menjaga agar semua *subclass* memiliki struktur yang sama, meskipun perilakunya bisa berbeda-beda sesuai kebutuhan.

Analoginya seperti kontrak kerja: Misalnya seorang manajer menyusun daftar tugas yang harus dikerjakan oleh semua karyawan baru, tapi tidak memberitahu detail cara mengerjakannya—itu terserah karyawan. Yang penting, setiap karyawan harus menyelesaikan tugas yang sama, meskipun dengan cara yang berbeda. Dalam *OOP*, manajer itu adalah *abstract class*, daftar tugas adalah *abstract* , dan karyawan adalah *subclass* yang mengisi detail cara menyelesaikan tugas tersebut.

2.2.1 Ciri – Ciri *Abstract* pada Python

a. Didekorasi dengan *@abstract*

Abstract ditandai dengan *decorator @abstract* yang berasal dari modul abc. Ini menandakan bahwa ini belum memiliki implementasi.

b. Tidak memiliki implementasi (hanya deklarasi)

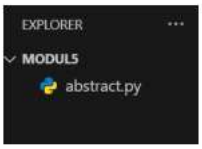
Abstract hanya dideklarasikan dengan nama dan parameter, tapi tidak ada logika atau isi di dalamnya. Biasanya menggunakan pass sebagai *placeholder*.

c. Wajib diimplementasikan oleh *subclass*

d. Tidak dapat dipanggil langsung dari *abstract class*

2.3 Implementasi *Abstract Class* dan *Abstract* pada Program Python

a. Buat folder baru untuk menyimpan project python dan buat file didalamnya.



b. Selanjutnya buat *Abstract Class* dengan nama Mobil.

```python
from abc import ABC, abstractmethod

# Abstract class
class Mobil(ABC):

    @abstractmethod
    def nyalakan_mesin(self):
        pass

    @abstractmethod
    def matikan_mesin(self):
        pass
```

c. Kemudian buat *abstract* pada *class* mobil tadi, dengan nama nyalakan_mesin() dan matikan_mesin(). Pastikan pada *abstract* diberi decorator *@abstract*.

```python
from abc import ABC, abstractmethod

# Abstract class
class Mobil(ABC):

    @abstractmethod
    def nyalakan_mesin(self):
        pass

    @abstractmethod
    def matikan_mesin(self):
        pass
```

d. Kemudian buat *subclass*nya, sebagai contoh buat *subclass* dengan nama MobilSport().

```python
class MobilSport(Mobil):

    def nyalakan_mesin(self):
        print("Mesin mobil sport dinyalakan: Vroom Vroom!")

    def matikan_mesin(self):
        print("Mesin mobil sport dimatikan.")
```

e. Setelah berhasil membuat *class* MobilSport() yang merupakan *subclass* dari *class* Mobil tadi. Tambahkan untuk mengoverride pada *abstract class*. (pastikan semua *abstract* sudah diterapkan pada *subclass*nya).

```python
class MobilSport(Mobil):

    def nyalakan_mesin(self):
        print("Mesin mobil sport dinyalakan: Vroom Vroom!")

    def matikan_mesin(self):
        print("Mesin mobil sport dimatikan.")
```

f. Tambahkan aksi dalam nyalakan_mesin() dan juga matikan_mesin().

```python
class MobilSport(Mobil):

    def nyalakan_mesin(self):
        print("Mesin mobil sport dinyalakan: Vroom Vroom!")

    def matikan_mesin(self):
        print("Mesin mobil sport dimatikan.")
```

g. Langkah terakhir, buat objek dari MobilSport() yang merupkan *subclass* dari *abstract class*.

```python
mobil_1 = MobilSport()

mobil_1.nyalakan_mesin()
mobil_1.matikan_mesin()
```

h. Nanti outputnya akan:

```
NGAJAR 2025/PBO/modul5/abstract.py"
Mesin mobil sport dinyalakan: Vroom Vroom!
Mesin mobil sport dimatikan.
PS D:\semester 4\NGAJAR 2025\PBO\modul5>
```

BAB III

TUGAS PENDAHULUAN

3.1 Soal

1. Jelaskan pengertian class, object, dan Constructor dalam python serta hubungannya dengan konsep abstract class!
2. Apa yang dimaksud dengan abstract class dan abstract method dalam Python. Jelaskan menggunakan bahasamu sendiri!
3. Mengapa abstract class tidak dapat diinstunisasi secara langsung?
4. Berikan sebuah studi kasus sederhana (misalnya: Sistem kendaraan, hewan, atau alat musik) yang membutuhkan penggunaan abstract class.
5. Buatlah potongan code Python yang berisi:
   a. sebuah abstract class dengan dua abstract method.
   b. Sebuah Subclass yang mengImplementasikan kedua method tersebut.
   c. Pembuatan objek dari subclass dan pemanggilan method-nya.
   d. Jelaskan fungsi dari setiap bagian kode yang kamu buat.

3.2 Jawaban

1. Class adalah blueprint membuat objek. Objek adalah instance nyata dari class tersebut. Constructor adalah metode khusus dalam class. Hubungan nya, abstrak class tetap menggunakan konsep Class dan Constructor tetapi hanya sebagai kerangka.
2. Abstract class adalah class yang berfungsi sebagai kerangka dasar dan tidak bisa dibuat objeknya secara langsung karena biasanya berisi abstract method. Abstract method ini wajib di implementasikan oleh Class turunannya sehingga abstract class digunakan untuk memastikan bahwa semua class turunan memiliki struktur sesuai ketentuan.
3. Karena masih mengandung abstract method yang belum memiliki implementasi. Abstract Class hanya boleh digunakan sebagai dasar untuk Class lain yang akan melengkapi semua methodnya.
4. Ketu punya abstract class kendaraan yang memiliki abstract method Jalan(). Karena setiap Jenis kendaraan (mobil, motor) cara jalannya berbeda, maka setiap Class turunan wajib mengimplementasikan method Jalan() sesuai karakteristiknya.

```
from abc import ABC, abstractmethod
class Kendaraan (ABC):
    @abstractmethod
    def Jalan (self):
        pass
class Mobil (Kendaraan):
    def Jalan (self):
        print ("Mobil berjalan dengan menggunakan mesin")
class Motor (Kendaraan):
    def Jalan (self):
        Print ("Sepeda motor berjalan dengan cara menghidupkan
               mesin dan menggerakan roda")
Mobil = Mobil()
Mobil. Jalan()

motor = Motor()
motor. Jalan()
```

5.
```
from abc import ABC, abstract method
# a. abstract class dengan dua abstract method
  class Kendaraan (ABC):
      @abstractmethod
      def nyalakan_mesin (self):
          pass
      @abstract method
      def matikan_mesin (self):
          pass
  # b. Subclass yang mengimplementasikan kedua method
    class Mobil (Kendaraan):
        def nyalakan_mesin (self):
            Print("Mesin mobil dinyalakan")
        def matikan_mesin (self):
            Print("Mesin mobil dimatikan")
  # c. Pembuatan objek dan pemanggilan method
    mobil_saya = mobil()
    mobil_saya = nyalakan_mesin()
    mobil_saya = matikan_mesin()
```

d. from abc import , abstractmethod

- mengimpor ABC (Abstract Base Class) dan abstractmethod dari modul abc untuk membuat kelas abstrak

Class Kendaraan(ABC):

- mendefinisikan kelas abstrak bernama kendaraan, yang tidak bisa langsung di instansiasi.

@ abstractmethod def nyalakan-mesin(self): dan matikan_mesin(self):

- Dua method abstrak yang harus diimplementasikan oleh Subclass

Class Mobil (kendaraan):

- mobil wajib mengimplementasikan semua method abstrak

mobil_saya = Mobil()

- membuat objek mobil_saya dari Class Mobil

mobil_saya.nyalakan_mesin() dan mobil_saya.matikan_mesin()

- memanggil method yang sudah diimplementasikan dalam Class mobil.

BAB IV
IMPLEMENTASI

4.1 Tugas Praktikum

4.1.1 Tugas Praktikum No. 1

Buatlah sebuah program Python dengan ketentuan sebagai berikut:

1. Buat *abstract class* bernama Manusia yang memiliki tiga *abstract* :
   a. berbicara()
   b. bekerja()
   c. makan()
2. Buat empat *subclass* dari Manusia, yaitu: Joko, Beni, Fani, dan Jani. Masing-masing *subclass* mengimplementasikan semua abstrak tersebut dengan pesan berbeda sesuai karakter mereka.
3. Buat objek dari masing-masing *subclass* dan panggil setiap -nya satu per satu (tanpa menggunakan list atau perulangan).

4.1.2 Tugas Praktikum No. 2

1. Buatlah sebuah *abstract class* bernama PerangkatElektronik dengan:
   a. Tiga *abstract* :
      - nyalakan() – untuk menyalakan perangkat.
      - matikan() – untuk mematikan perangkat.
      - gunakan(jam: int) – menerima parameter durasi pemakaian dalam jam.
2. Tambahkan properti:
   a. energi_tersisa bertipe integer.
   b. Set nilai awalnya di constructor sebagai 100 (maksimal 100%).
3. Buat dua *subclass* dari PerangkatElektronik:
   a. Laptop
      - Mengurangi energi_tersisa sebesar 10 jam setiap kali gunakan() dipanggil.
      - Jika energi_tersisa kurang dari 0, set ke 0 dan cetak pesan bahwa baterai habis.
   b. Kulkas

- Mengurangi energi_tersisa sebesar 5 jam.
- Jika energi turun di bawah 20, cetak peringatan “Energi rendah, kulkas butuh daya tambahan!”

4. Tambahkan status() yang dapat digunakan untuk menampilkan:
   a. Tipe perangkat
   b. Energi tersisa
5. Buat objek dari masing-masing *subclass*, panggil semua metodenya dengan nilai jam berbeda (tanpa list atau perulangan).

4.2 Source Code

4.2.1 Source Code Soal 1

```
from abc import ABC, abstractmethod

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

    @abstractmethod
    def berbicara(self):
        pass

    @abstractmethod
    def bekerja(self):
        pass

    @abstractmethod
    def makan(self):
        pass

class Orang(Manusia):
    def __init__(self, nama, pekerjaan, makanan):
        super().__init__(nama)
```

```
        self.pekerjaan = pekerjaan
        self.makanan = makanan

    def berbicara(self):
        print(f"Halo, perkenalkan nama saya {self.nama}")

    def bekerja(self):
        print(f"Saya saat ini sedang {self.pekerjaan}")

    def makan(self):
        print(f"Saya sedang makan {self.makanan}")

def main():
    nama = input("Masukkan Nama: ")
    pekerjaan = input("Masukkan Pekerjaan: ")
    makanan = input("Sedang makan apa?: ")

    manusia = Orang(nama, pekerjaan, makanan)

    manusia.berbicara()
    manusia.bekerja()
    manusia.makan()

if __name__ == "__main__":
    main()
```

4.2.2 Source Code Soal 2

```
from abc import ABC, abstractmethod


class PerangkatElektronik(ABC):
    def __init__(self, tipe: str):  # <-- perbaikan: __init__ bukan _init_
```

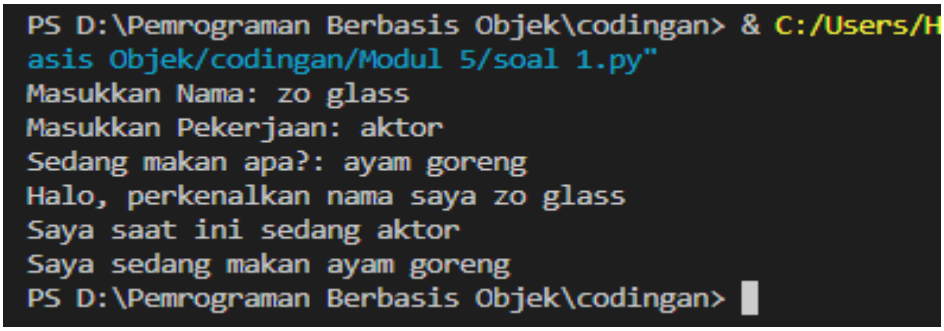
```
        self.tipe = tipe
        self.energi_tersisa = 100

    @abstractmethod
    def nyalakan(self):
        pass

    @abstractmethod
    def matikan(self):
        pass

    @abstractmethod
    def gunakan(self, jam: int):
        pass

    def status(self):
        print(f"Tipe perangkat: {self.tipe}")
        print(f"Energi tersisa: {self.energi_tersisa}%")

class Perangkat(PerangkatElektronik):
    def __init__(self, tipe: str):
        super().__init__(tipe)

    def nyalakan(self):
        print(f"{self.tipe} dinyalakan.")

    def matikan(self):
        print(f"{self.tipe} dimatikan.")

    def gunakan(self, jam: int):
        if self.tipe == "Laptop":
            pemakaian = 10  jam
```

```
            self.energi_tersisa -= pemakaian
            if self.energi_tersisa <= 0:
                self.energi_tersisa = 0
                print("Baterai habis. Laptop mati.")
            else:
                print(f"Laptop digunakan selama {jam} jam.")
        elif self.tipe == "Kulkas":
            pemakaian = 5  jam
            self.energi_tersisa -= pemakaian
            if self.energi_tersisa <= 0:
                self.energi_tersisa = 0
                print("Energi habis. Kulkas mati.")
            elif self.energi_tersisa < 20:
                print("Energi rendah, kulkas butuh daya tambahan!")
            print(f"Kulkas beroperasi selama {jam} jam.")
        else:
            print("Tipe perangkat tidak dikenali.")

def main():
    print("Pilih tipe perangkat:")
    print("1. Laptop")
    print("2. Kulkas")
    pilihan = input("Masukkan pilihan (1/2): ")

    if pilihan == "1":
        tipe = "Laptop"
    elif pilihan == "2":
        tipe = "Kulkas"
    else:
        print("Pilihan tidak valid. Harus 1 atau 2.")
        return
```

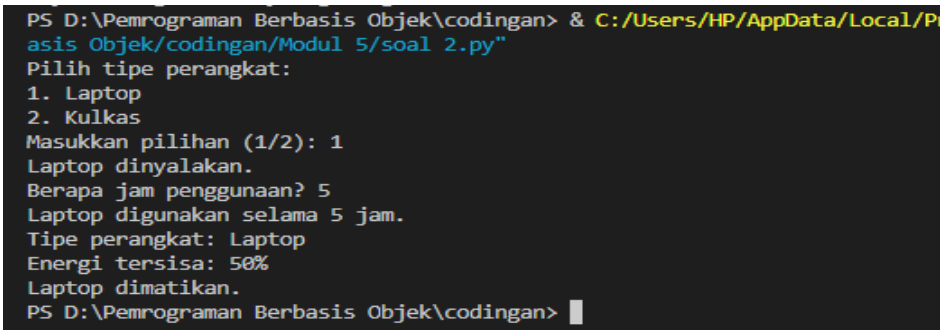
```
    perangkat = Perangkat(tipe)
    perangkat.nyalakan()

    jam_str = input("Berapa jam penggunaan? ")
    if not jam_str.isdigit():
        print("Jam harus berupa angka.")
        return

    jam = int(jam_str)
    perangkat.gunakan(jam)
    perangkat.status()
    perangkat.matikan()

if __name__ == "__main__":
    main()
```

4.3 Hasil

4.3.1 Hasil Soal 1



4.3.2 Hasil Soal 2



4.4 Penjelasan

4.4.1 Penjelasan Soal 1

Codingan yang saya buat di atas menggunakan bahasa Python. Kelas abstrak `Manusia` didefinisikan sebagai *blueprint* dengan tiga metode abstrak, yaitu `berbicara`, `bekerja`, dan `makan`, yang harus diimplementasikan oleh kelas turunannya. Kelas `Orang` merupakan turunan dari `Manusia` yang mengimplementasikan ketiga metode tersebut sesuai dengan atribut `nama`, `pekerjaan`, dan `makanan` yang diberikan saat objek dibuat. Fungsi `main()` berfungsi untuk menerima input dari pengguna dan mencetak hasil implementasi metode-metode yang telah diisi melalui objek `Orang`. Program ini menunjukkan bagaimana pewarisan dan abstraksi dapat digunakan untuk membuat struktur kode yang terorganisir dan fleksibel.

.

4.4.2 Penjelasan Soal 2

Codingan yang saya buat di atas merupakan Python untuk mensimulasikan perilaku perangkat elektronik. Kelas abstrak PerangkatElektronik berfungsi sebagai kerangka dasar dengan atribut tipe dan energi_tersisa, serta tiga metode abstrak: nyalakan, matikan, dan gunakan, yang wajib diimplementasikan oleh kelas turunannya. Kelas Perangkat mengimplementasikan ketiga metode tersebut dengan perilaku berbeda tergantung tipe perangkat, yaitu "Laptop" atau "Kulkas", yang mempengaruhi konsumsi energi per jam. Pada fungsi main(), pengguna diminta memilih jenis perangkat dan durasi penggunaan dalam jam, lalu program menampilkan aktivitas perangkat tersebut seperti dinyalakan, digunakan, status energi, dan dimatikan. Kode ini menunjukkan bagaimana pewarisan dan polymorphism dapat digunakan untuk menangani berbagai jenis perangkat dengan logika penggunaan yang berbeda, sekaligus memberikan interaksi sederhana dengan pengguna melalui *input* dan *output* yang intuitif. Selain itu, kondisi seperti energi habis atau rendah juga ditangani dengan baik untuk menambah *realisme* dalam simulasi.

BAB V
PENUTUP

5.1 Analisa

Abstract class berperan sebagai kerangka dasar yang tidak dapat diinstansiasi langsung, melainkan diturunkan oleh *subclass*. Dengan adanya *abstract* di dalamnya, *abstract class* memastikan bahwa setiap *subclass* memiliki struktur perilaku yang sama meskipun implementasinya berbeda.

Konsep ini sangat penting dalam *object-oriented programming* karena membantu menjaga konsistensi dan memperkuat prinsip pewarisan. Dengan menggunakan *abstract class* dan *abstract* , pengembang dapat membuat sistem yang lebih terstruktur, fleksibel, dan mudah dikembangkan.

5.2 Kesimpulan

Dalam pengembangan perangkat lunak berbasis *object-oriented programming*, penggunaan *abstract class* dan *abstract* menjadi sangat penting untuk menciptakan arsitektur kode yang rapi, konsisten, dan mudah dikembangkan. *Abstract class* berfungsi sebagai cetak biru yang mendefinisikan struktur dasar tanpa memberikan implementasi penuh, sementara *abstract* menjadi kontrak yang mewajibkan setiap *subclass* untuk mengisi sendiri perilaku yang diharapkan. Dengan penerapan konsep ini, pengembang dapat membangun sistem yang fleksibel, menjaga standar struktur antar kelas, serta mempermudah proses pemeliharaan dan pengembangan di masa depan.

1. *Abstract class* tidak dapat diinstansiasi langsung — hanya bisa diwariskan oleh *subclass*.
2. *Abstract* wajib diimplementasikan oleh *subclass* — karena tidak memiliki isi di kelas induk.
3. Menjaga konsistensi dan struktur dalam pewarisan kode — sehingga semua turunan mengikuti pola yang sama.