# TUGAS 5 PRAKTIKUM PBO

# SIFAT-SIFAT PBO


**OLEH:**

**MUHAMMAD RAIHAN PUTERANDA**

**121140089**

**RB**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNOLOGI PRODUKSI DAN INDUSTRI**

**INSTITUT TEKNOLOGI SUMATERA**

**LAMPUNG SELATAN**

**2023**

## A. Abstraksi

Abstraksi adalah suatu sifat untuk menyembunyikan informasi-informasi yang tidak penting atau tidak relevan terhadap program yang dibuat. Maksudnya adalah, user tidak perlu mengetahui detail dari fungsi-fungsi yang terdapat pada suatu program, mereka hanya tahu cara mengambil atau menggunakan fungsi tersebut sehingga dapat mengurangi kompleksitas dari kode program serta mempermudah dalam pengembangan dan pemeliharaannya. Sifat ini dapat diterapkan dengan menggunakan:

### 1. Interface

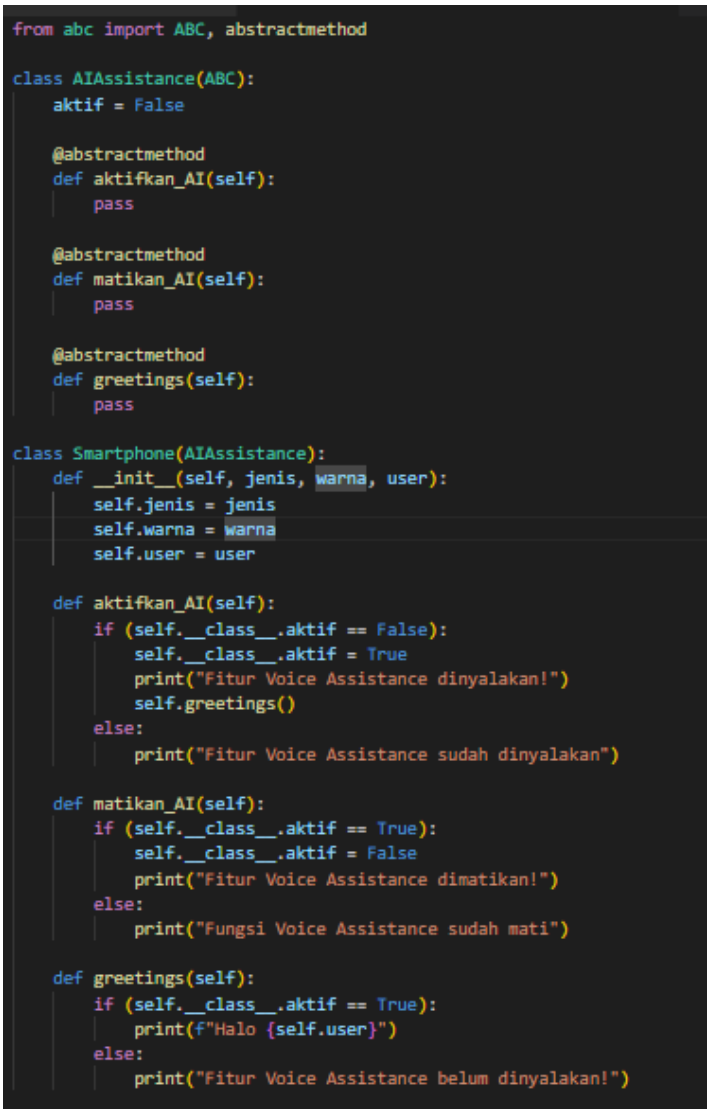
Semacam kontrak yang berisi konsep atau ide dasar dan memiliki fungsi-fungsi yang akan diterapkan pada suatu program nantinya, biasanya diterapkan bersamaan dengan Inheritance dimana fungsi tersebut akan dikembangkan dan harus ada pada kelas turunannya.

```python
from abc import ABC, abstractmethod

class AIAssistance(ABC):
    aktif = False

    @abstractmethod
    def aktifkan_AI(self):
        pass

    @abstractmethod
    def matikan_AI(self):
        pass

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

AI Assistance sebagai Interface dari perangkat

### 2. Abstract Class

Membuat kelas abstrak dimana terdapat satu atau lebih metode abstrak namun tidak ada implementasi yang dilakukan didalamnya dan dapat diwariskan ke subclassnya. Untuk membuat suatu metode abstrak dapat menggunakan decorator **@abstractmethod** yang kemudian detail fungsinya akan diimplementasikan di subclassnya. Abstract class juga menjadi bagian dari penerapan Interface.



```python
from abc import ABC, abstractmethod

class AIAssistance(ABC):
    aktif = False

    @abstractmethod
    def aktifkan_AI(self):
        pass

    @abstractmethod
    def matikan_AI(self):
        pass

    @abstractmethod
    def greetings(self):
        pass

class Smartphone(AIAssistance):
    def __init__(self, jenis, warna, user):
        self.jenis = jenis
        self.warna = warna
        self.user = user

    def aktifkan_AI(self):
        if (self.__class__.aktif == False):
            self.__class__.aktif = True
            print("Fitur Voice Assistance dinyalakan!")
            self.greetings()
        else:
            print("Fitur Voice Assistance sudah dinyalakan")

    def matikan_AI(self):
        if (self.__class__.aktif == True):
            self.__class__.aktif = False
            print("Fitur Voice Assistance dimatikan!")
        else:
            print("Fungsi Voice Assistance sudah mati")

    def greetings(self):
        if (self.__class__.aktif == True):
            print(f"Halo {self.user}")
        else:
            print("Fitur Voice Assistance belum dinyalakan!")
```

Metode dari kelas abstrak akan diimplementasikan pada subclassnya

### 3. Access Modifier

Menggunakan pembeda hak akses pada data yang ingin disembunyikan (Public, Protected, Private) yang kemudian akan diakses oleh suatu metode baru.

```python
class Perpustakaan:
    def __init__(self, nama_buku, penulis, penerbit):
        self.nama_buku = nama_buku
        self._penulis = penulis
        self.__penerbit = penerbit

    def get_penulis(self):
        return self._penulis

    def get_penerbit(self):
        return self.__penerbit
```

Metode “get” untuk mengambil atribut dengan akses protected & private

## B. Enkapsulasi

Hampir mirip seperti abstraksi, enkapsulasi berarti menyembunyikan informasi berupa kode yang bisa berupa atribut atau metode dalam satu kelas dengan cara membungkusnya. Maksudnya adalah, atribut atau metode disembunyikan agar terlindungi dari dunia luar, dalam hal ini kelas lain ataupun diluar kelas. Cara membungkusnya yaitu dengan menerapkan access modifier berupa Public, Protected, atau Private.

- **Public:** Atribut atau metode bisa diakses diseluruh kelas dan diluar kelas
- **Protected:** Atribut atau metode hanya bisa diakses oleh kelas dan subclassnya
- **Private:** Atribut atau metode hanya bisa diakses dikelasnya

```python
class Perpustakaan:
    def __init__(self, nama_buku, penulis, penerbit):
        self.nama_buku = nama_buku
        self._penulis = penulis
        self.__penerbit = penerbit

    def get_penerbit(self):
        print(f"Penerbit buku itu adalah {self.__penerbit}")

class Perpus_Mini(Perpustakaan):
    def get_penulis(self):
        print(f"Penulisnya bernama {self._penulis}")

    def get_penerbit(self):
        return self.__penerbit

# main

buku1 = Perpus_Mini("Buku1", "Penulis1", "Penerbi1")
print(buku1.nama_buku)
buku1.get_penerbit() # Error
print(buku1.__penerbit) # Error
```

Contoh penerapan Enkapsulasi

Untuk bisa mengakses dan mengubah data yang bersifat protected atau private dapat menggunakan decorator atau menerapkan setter & getter (metode publik).

```python
@property
def penerbit(self):
    return self.__penerbit

@penerbit.setter
def penerbit(self, penerbit):
    self.__penerbit = penerbit

# atau

# Getter
def get_penerbit(self):
    return self.__penerbit

# Setter
def set_penerbit(self, penerbit):
    self.__penerbit = penerbit
```

Mengakses atribut private

## C. Inheritance

Merupakan konsep yang memungkinkan penurunan data berupa atribut dan fungsi dari induk kepada anaknya tanpa perlu mengimplementasikannya lagi. Kelas anak atau subclass dapat memodifikasi sifat yang diwarisi dan menambahkan sifat baru. Inheritance diperlukan ketika suatu kelas memiliki jenis yang bermacam-macam sehingga membutuhkan data yang berbeda dan memerlukan data baru yang mungkin saja berbeda dengan jenis lainnya.

### 1. Single Inheritance

Subclass hanya diwarisi sifat dari satu induk saja, untuk konsep ini digunakan **super().__init__(*sifat yang diwariskan*)** untuk menetapkan apa saja yang diwariskan dari induknya

```python
# Parent class
class Komputer:
    def __init__(self, nama ,jenis, harga, merk):
        self.nama = nama
        self.jenis = jenis
        self.harga = harga
        self.merk = merk

# Child class
class Processor(Komputer):
    def __init__(self, nama, jenis, harga, merk, jumlah_core, kecepatan_processor):
        super().__init__(nama, jenis, harga, merk)
        self.jumlah_core = jumlah_core
        self.kecepatan_processor = kecepatan_processor
```

Processor mewarisi nama, jenis, harga, dan merk dari Komputer

Atau bisa juga langsung diterapkan nilainya pada **super** sehingga bisa dihapus nilai yang diwariskan pada konstruktor subclass.

```
# Parent class
class Komputer:
    def __init__(self, nama ,jenis, harga, merk):
        self.nama = nama
        self.jenis = jenis
        self.harga = harga
        self.merk = merk

# Child class
class Processor(Komputer):
    def __init__(self, jumlah_core, kecepatan_processor):
        super().__init__("Intel-i7", "Processor", 3000000, "Intel")
        self.jumlah_core = jumlah_core
        self.kecepatan_processor = kecepatan_processor
```

Nilai dari atribut yang diwariskan langsung ditulis

### 2. Multiple Inheritance

Subclass mewarisi fungsi dari banyak induk. Untuk kasus ini biasanya penggunaan **super** tidak cukup karena fungsi dari masing-masing induk bisa saja berbeda, oleh karena itu disarankan untuk mengubah **super** menjadi nama induk masing-masing seperti berikut:

```
# Child class
class Anak(Ayah, Ibu):
    def __init__(self, main, masak):
        Ayah.__init__(self, main)
        Ibu.__init__(self, masak)

    def kerjaan(self):
        return Ayah.pekerjaan(self) + Ibu.masakan(self)
```

Anak mewarisi sifat dari kedua orang tua

## D. Polymorphism

Konsep polimorfisme berarti kemampuan objek untuk menerima dan memberikan respon terhadap suatu fungsi yang sama dengan cara yang berbeda. Tujuannya yakni untuk membuat kelas yang lebih gampang dimodifikasi sehingga menjadi lebih fleksibel karena berbagai kelas dapat memiliki nama fungsi yang sama. Penerapan polimorfisme ada 4 cara antara lain:

### 1. Polymorphism dengan Operator

Operator seperti + dapat melakukan operasi yang berbeda untuk tipe data yang berbeda pula, misalkan untuk integer maka operasinya penjumlahan, sedangkan jika string maka akan menggabungkannya menjadi kalimat utuh.

```
a = 1
b = 2
c = "test"
d = "kode"
jumlah = a + b
gabung = c + " " + d
print(jumlah) # 3
print(gabung) # test kode
```

Polimorfisme Operator +

### 2. Polymorphism dengan Fungsi

Fungsi seperti **max()** dapat memberikan hasil yang berbeda untuk berbagai tipe data, misalkan pada sebuah list untuk integer maka akan mencari nilai max nya, sedangkan untuk yang berupa teks, akan dihitung dari nilai ASCII nya.

```
angka = [1, 2, 3, 4, 5]
teks = ["Halo", "dunia", "!"]

print(max(angka)) # 5
print(max(teks)) # dunia
```

Polimorfisme Fungsi max()

### 3. Polymorphism dengan Kelas

Banyak kelas dapat memiliki fungsi dengan nama yang sama dengan kelas lain namun memiliki isi yang berbeda. Fungsi yang sama dari berbagai kelas tersebut kemudian bisa dipanggil yang menyebabkan *Overloading*, yakni fungsi akan dijalankan sesuai objeknya masing-masing.

```python
class Robot1:
    def __init__(self, nama = "Antares", health = 4000, damage = 5000):
        self.nama = nama
        self.health = health
        self.damage = damage

    def info(self):
        print(f"Nama: {self.nama}")
        print(f"HP: {self.health}")
        print(f"DMG: {self.damage}")
        print("Boost damage: 1.5x pada turn kelipatan 3 jika menang\n")

class Robot2:
    def __init__(self, nama = "Alphasetia", health = 4500, damage = 4000):
        self.nama = nama
        self.health = health
        self.damage = damage

    def info(self):
        print(f"Nama: {self.nama}")
        print(f"HP: {self.health}")
        print(f"DMG: {self.damage}")
        print("Boost health: 4000 pada turn kelipatan 2 jika menang\n")

# Overloading
def info_robot(*robots):
    for robot in robots:
        robot.info()
```

Polimorfisme dengan Kelas (+ Overloading)

## 4. Polymorphism dengan Inheritance

Subclass mewarisi fungsi dari induk dan dapat memodifikasinya agar sesuai dengan kebutuhannya, hal ini disebut juga sebagai *Overriding*.

```python
# Parent class
class Robot:
    jumlah_turn = 0

    def __init__(self, nama, health, damage):
        self.nama = nama
        self.health = health
        self.damage = damage

    def lakukan_aksi(self, a, b, robot_lain):
        Robot.jumlah_turn += 1
        self.aksi(robot_lain)

    # Fungsi akan di override
    def aksi(self):
        pass

# Child class
class Antares(Robot):
    def __init__(self):
        super().__init__("Antares", 50000, 5000)

    # Overriding fungsi parent class
    def aksi(self, robot_lain):
        self.menang += 1
        robot_lain.health -= self.damage
        print(f"Antares menyerang sebanyak {self.damage} DMG")
```

Polimorfisme dengan Inheritance

**Referensi:** Modul Praktikum & Pembelajaran