Kelas Kolaborasi Pemrograman Python

Program Praktisi Mengajar Batch 4 2024 Universitas Amikom Yogyakarta
Muhammad Oriza Nurfajri S.Kom., M.IT.
Sesi 5 10 Juni 2024 07.00 - 08.30 Durasi 90 Menit Per Sesi

OBJECT ORIENTED PROGRAMMING

HEADS UP

Pemrograman Berorientasi Objek adalah paradigma pemrograman yang menggunakan "objek" - yang bisa berupa data dan kode, dalam bentuk data fields (sering disebut sebagai atribut atau properti), dan kode, dalam bentuk prosedur (sering dikenal sebagai metode). Python mendukung paradigma ini dengan sangat baik dan merupakan salah satu bahasa yang paling banyak digunakan untuk OOP.

Konsep Dasar OOP

1. Class dan Object
    - Class adalah blueprint untuk objek. Ia mendefinisikan atribut dan metode yang harus dimiliki oleh objek yang dibuat dari class tersebut.
    - Object adalah instance dari sebuah class. Ia adalah representasi nyata dari class dengan nilai konkret dari atribut yang didefinisikan oleh class.
2. Encapsulation
    - Proses menyembunyikan detail implementasi dari suatu objek dan hanya menunjukkan fungsionalitas yang dibutuhkan.
3. Inheritance
    - Mekanisme di mana sebuah class dapat mewarisi atribut dan metode dari class lain.
4. Polymorphism
    - Kemampuan untuk menggunakan interface yang sama untuk objek yang berbeda, dengan masing-masing objek dapat melakukan implementasi yang berbeda terhadap interface tersebut.
5. Abstraction
    - Proses menyembunyikan detail yang tidak relevan dari pengguna dan menunjukkan hanya fungsionalitas yang diperlukan.

Implementasi Sederhana Kelas dan Object

```python
# Definisikan kelas
class Mobil:
    # Konstruktor untuk inisialisasi objek
    def __init__(self, merk, model, tahun):
        self.merk = merk
        self.model = model
        self.tahun = tahun

    # Metode untuk menampilkan informasi mobil
    def info(self):
        return f"{self.merk} {self.model} ({self.tahun})"

    # Metode untuk menyalakan mesin mobil
    def nyalakan_mesin(self):
        return f"Mesin {self.merk} {self.model} menyala!"

    # Metode untuk mematikan mesin mobil
    def matikan_mesin(self):
        return f"Mesin {self.merk} {self.model} mati."
```

```python
# Buat objek dari kelas Mobil
mobil1 = Mobil("Toyota", "Corolla", 2020)
mobil2 = Mobil("Honda", "Civic", 2018)

# Panggil metode info untuk melihat informasi mobil
print(mobil1.info())
print(mobil2.info())

# Panggil metode nyalakan_mesin dan matikan_mesin
print(mobil1.nyalakan_mesin())
print(mobil2.matikan_mesin())
```

Penjelasan

1. Definisi Kelas: Mobil adalah kelas yang memiliki atribut merk, model, dan tahun.
2. Konstruktor: Metode __init__ digunakan untuk menginisialisasi objek dengan atribut yang diberikan.
3. Metode info: Metode ini mengembalikan informasi tentang mobil dalam format string.
4. Metode nyalakan_mesin dan matikan_mesin: Metode ini untuk menyalakan dan mematikan mesin mobil, masing-masing mengembalikan pesan yang sesuai.
5. Instansiasi Objek: mobil1 dan mobil2 adalah objek yang dibuat dari kelas Mobil dengan nilai atribut yang berbeda.
6. Pemanggilan Metode: Menampilkan informasi mobil serta status mesin yang menyala atau mati melalui metode yang telah didefinisikan.

Enkapsulasi (Encapsulation)

Enkapsulasi adalah teknik untuk menyembunyikan data internal sebuah objek dan hanya menyediakan metode untuk mengakses dan memodifikasi data tersebut.

```
# Definisikan kelas dengan enkapsulasi
class Mobil:
    # Konstruktor untuk inisialisasi objek
    def __init__(self, merk, model, tahun):
        self.__merk = merk  # Atribut private
        self.__model = model  # Atribut private
        self.__tahun = tahun  # Atribut private

    # Metode untuk mendapatkan informasi merk
    def get_merk(self):
        return self.__merk

    # Metode untuk mengatur informasi merk
    def set_merk(self, merk):
        self.__merk = merk

    # Metode untuk mendapatkan informasi model
    def get_model(self):
        return self.__model
```

```python
# Metode untuk mengatur informasi model
def set_model(self, model):
    self.__model = model

# Metode untuk mendapatkan informasi tahun
def get_tahun(self):
    return self.__tahun

# Metode untuk mengatur informasi tahun
def set_tahun(self, tahun):
    self.__tahun = tahun

# Metode untuk menampilkan informasi mobil
def info(self):
    return f"{self.__merk} {self.__model} ({self.__tahun})"

# Metode untuk menyalakan mesin mobil
def nyalakan_mesin(self):
    return f"Mesin {self.__merk} {self.__model} menyala!"

# Metode untuk mematikan mesin mobil
def matikan_mesin(self):
    return f"Mesin {self.__merk} {self.__model} mati."
```

```python
# Buat objek dari kelas Mobil
mobil1 = Mobil("Toyota", "Corolla", 2020)
mobil2 = Mobil("Honda", "Civic", 2018)

# Panggil metode info untuk melihat informasi mobil
print(mobil1.info())
print(mobil2.info())

# Panggil metode nyalakan_mesin dan matikan_mesin
print(mobil1.nyalakan_mesin())
print(mobil2.matikan_mesin())

# Menggunakan setter untuk mengubah nilai atribut
mobil1.set_merk("Nissan")
mobil1.set_model("Altima")
mobil1.set_tahun(2021)

# Melihat perubahan yang telah dibuat menggunakan getter
print(mobil1.info())
```

Buat kode berikut pada file lain:

```python
# Impor kelas Mobil dari file mobil.py
from mobil import Mobil

# Buat objek dari kelas Mobil
mobil1 = Mobil("Toyota", "Corolla", 2020)


try:
    print(mobil1.__merk)
except AttributeError as e:
    print(f"Error: {e}")

# Cara yang benar untuk mengakses atribut private menggunakan getter
print(mobil1.get_merk())

# Cara yang benar untuk mengubah atribut private menggunakan setter
mobil1.set_merk("Nissan")
mobil1.set_model("Altima")
mobil1.set_tahun(2021)

# Melihat perubahan yang telah dibuat menggunakan getter
print(mobil1.info())
```

Pewarisan (Inheritance)

Pewarisan adalah salah satu pilar utama dalam Pemrograman Berorientasi Objek (OOP), di mana sebuah kelas (subclass/child) dapat mewarisi atribut dan metode dari kelas lain (superclass/parent). (dalam file mobil.py)

```python
# Definisikan kelas Turunan yang mewarisi dari Mobil
class MobilSport(Mobil):
    def __init__(self, merk, model, tahun, top_speed):
        super().__init__(merk, model, tahun)  # Panggil konstruktor kelas induk
        self.__top_speed = top_speed  # Atribut khusus untuk MobilSport

    # Metode untuk mendapatkan informasi top speed
    def get_top_speed(self):
        return self.__top_speed

    # Metode untuk mengatur informasi top speed
    def set_top_speed(self, top_speed):
        self.__top_speed = top_speed

    # Override metode info untuk menyertakan top speed
    def info(self):
        return f"{super().info()} - Top Speed: {self.__top_speed} km/h"
```

Buat kode berikut pada file lain:

```python
# Impor kelas Mobil dan MobilSport dari file mobil.py
from mobil import Mobil, MobilSport

# Buat objek dari kelas Mobil
mobil1 = Mobil("Toyota", "Corolla", 2020)
print(mobil1.info())
print(mobil1.nyalakan_mesin())

# Buat objek dari kelas MobilSport
mobil_sport1 = MobilSport("Ferrari", "488", 2021, 330)
print(mobil_sport1.info())
print(mobil_sport1.nyalakan_mesin())


try:
    print(mobil_sport1.__top_speed)
except AttributeError as e:
    print(f"Error: {e}")

# Cara yang benar untuk mengakses atribut private menggunakan getter
print(mobil_sport1.get_top_speed())

# Menggunakan setter untuk mengubah nilai atribut
mobil_sport1.set_top_speed(340)
print(mobil_sport1.info())
```

Polimorfisme (Polimorphism)

Polimorfisme adalah konsep dalam Pemrograman Berorientasi Objek (OOP) di mana objek dari kelas yang berbeda dapat diperlakukan sebagai objek dari kelas yang sama melalui interface yang sama. Ini memungkinkan penggunaan metode yang sama pada objek yang berbeda. (dalam file mobil.py)

```python
# mobil.py

class Mobil:
    def __init__(self, merk, model, tahun):
        self.__merk = merk  # Atribut private
        self.__model = model  # Atribut private
        self.__tahun = tahun  # Atribut private
```

```
    # Metode untuk mendapatkan informasi merk
    def get_merk(self):
        return self.__merk

    # Metode untuk mengatur informasi merk
    def set_merk(self, merk):
        self.__merk = merk

    # Metode untuk mendapatkan informasi model
    def get_model(self):
        return self.__model

    # Metode untuk mengatur informasi model
    def set_model(self, model):
        self.__model = model

    # Metode untuk mendapatkan informasi tahun
    def get_tahun(self):
        return self.__tahun

    # Metode untuk mengatur informasi tahun
    def set_tahun(self, tahun):
        self.__tahun = tahun

    # Metode untuk menampilkan informasi mobil
    def info(self):
        return f"{self.__merk} {self.__model} ({self.__tahun})"

    # Metode untuk menyalakan mesin mobil
    def nyalakan_mesin(self):
        return f"Mesin {self.__merk} {self.__model} menyala!"

    # Metode untuk mematikan mesin mobil
    def matikan_mesin(self):
        return f"Mesin {self.__merk} {self.__model} mati."

# Definisikan kelas Turunan yang mewarisi dari Mobil
class MobilSport(Mobil):
    def __init__(self, merk, model, tahun, top_speed):
        super().__init__(merk, model, tahun)  # Panggil konstruktor kelas
induk
        self.__top_speed = top_speed  # Atribut khusus untuk MobilSport

    # Metode untuk mendapatkan informasi top speed
    def get_top_speed(self):
        return self.__top_speed

    # Metode untuk mengatur informasi top speed
```

```
    def set_top_speed(self, top_speed):
        self.__top_speed = top_speed

    # Override metode info untuk menyertakan top speed
    def info(self):
        return f"{super().info()} - Top Speed: {self.__top_speed} km/h"

    # Override metode nyalakan_mesin untuk tambahan fitur sport
    def nyalakan_mesin(self):
        return f"Mesin sport {self.get_merk()} {self.get_model()} menyala
dengan top speed {self.__top_speed} km/h!"

# Definisikan kelas Turunan lain yang mewarisi dari Mobil
class MobilListrik(Mobil):
    def __init__(self, merk, model, tahun, kapasitas_baterai):
        super().__init__(merk, model, tahun)  # Panggil konstruktor kelas
induk
        self.__kapasitas_baterai = kapasitas_baterai  # Atribut khusus untuk
MobilListrik

    # Metode untuk mendapatkan informasi kapasitas baterai
    def get_kapasitas_baterai(self):
        return self.__kapasitas_baterai

    # Metode untuk mengatur informasi kapasitas baterai
    def set_kapasitas_baterai(self, kapasitas_baterai):
        self.__kapasitas_baterai = kapasitas_baterai

    # Override metode info untuk menyertakan kapasitas baterai
    def info(self):
        return f"{super().info()} - Kapasitas Baterai:
{self.__kapasitas_baterai} kWh"

    # Override metode nyalakan_mesin untuk tambahan fitur listrik
    def nyalakan_mesin(self):
        return f"Mesin listrik {self.get_merk()} {self.get_model()} menyala
dengan kapasitas baterai {self.__kapasitas_baterai} kWh!"
```

Tulis kode berikut pada file eksekutor (main.py)

```python
# Impor kelas Mobil, MobilSport, dan MobilListrik dari file mobil.py
from MobilPoli import Mobil, MobilSport, MobilListrik

# Fungsi untuk menampilkan informasi dan menyalakan mesin mobil
def tampilkan_info_dan_nyalakan_mesin(mobil):
    print(mobil.info())
    print(mobil.nyalakan_mesin())

# Buat objek dari kelas Mobil
mobil1 = Mobil("Toyota", "Corolla", 2020)
mobil_sport1 = MobilSport("Ferrari", "488", 2021, 330)
mobil_listrik1 = MobilListrik("Tesla", "Model S", 2022, 100)

# # Menggunakan fungsi yang sama untuk objek yang berbeda (Polimorfisme)
tampilkan_info_dan_nyalakan_mesin(mobil1)
tampilkan_info_dan_nyalakan_mesin(mobil_sport1)
tampilkan_info_dan_nyalakan_mesin(mobil_listrik1)
```

Abstraksi (Abstract)

Abstraksi (Abstract) dalam OOP adalah konsep di mana kelas dasar (superclass) mendefinisikan metode yang harus diimplementasikan oleh kelas turunannya (subclass). Kelas abstrak tidak dapat diinstansiasi secara langsung dan biasanya digunakan untuk mendefinisikan antarmuka untuk kelas yang lain.

Python menyediakan modul abc (Abstract Base Classes) yang memungkinkan pembuatan kelas dan metode abstrak.

```python
# mobil.py

from abc import ABC, abstractmethod

class Mobil(ABC):
    def __init__(self, merk, model, tahun):
        self.__merk = merk  # Atribut private
        self.__model = model  # Atribut private
        self.__tahun = tahun  # Atribut private

    # Metode untuk mendapatkan informasi merk
    def get_merk(self):
        return self.__merk

    # Metode untuk mengatur informasi merk
```

```
    def set_merk(self, merk):
        self.__merk = merk

    # Metode untuk mendapatkan informasi model
    def get_model(self):
        return self.__model

    # Metode untuk mengatur informasi model
    def set_model(self, model):
        self.__model = model

    # Metode untuk mendapatkan informasi tahun
    def get_tahun(self):
        return self.__tahun

    # Metode untuk mengatur informasi tahun
    def set_tahun(self, tahun):
        self.__tahun = tahun

    # Metode untuk menampilkan informasi mobil
    def info(self):
        return f"{self.__merk} {self.__model} ({self.__tahun})"

    # Metode abstrak untuk menyalakan mesin mobil
    @abstractmethod
    def nyalakan_mesin(self):
        pass

    # Metode abstrak untuk mematikan mesin mobil
    @abstractmethod
    def matikan_mesin(self):
        pass

# Definisikan kelas Turunan yang mewarisi dari Mobil
class MobilSport(Mobil):
    def __init__(self, merk, model, tahun, top_speed):
        super().__init__(merk, model, tahun)  # Panggil konstruktor kelas
induk
        self.__top_speed = top_speed  # Atribut khusus untuk MobilSport

    # Metode untuk mendapatkan informasi top speed
    def get_top_speed(self):
        return self.__top_speed

    # Metode untuk mengatur informasi top speed
    def set_top_speed(self, top_speed):
        self.__top_speed = top_speed
```

```
    # Override metode info untuk menyertakan top speed
    def info(self):
        return f"{super().info()} - Top Speed: {self.__top_speed} km/h"

    # Implementasi metode abstrak nyalakan_mesin
    def nyalakan_mesin(self):
        return f"Mesin sport {self.get_merk()} {self.get_model()} menyala
dengan top speed {self.__top_speed} km/h!"

    # Implementasi metode abstrak matikan_mesin
    def matikan_mesin(self):
        return f"Mesin sport {self.get_merk()} {self.get_model()} mati."

# Definisikan kelas Turunan lain yang mewarisi dari Mobil
class MobilListrik(Mobil):
    def __init__(self, merk, model, tahun, kapasitas_baterai):
        super().__init__(merk, model, tahun)  # Panggil konstruktor kelas
induk
        self.__kapasitas_baterai = kapasitas_baterai  # Atribut khusus untuk
MobilListrik

    # Metode untuk mendapatkan informasi kapasitas baterai
    def get_kapasitas_baterai(self):
        return self.__kapasitas_baterai

    # Metode untuk mengatur informasi kapasitas baterai
    def set_kapasitas_baterai(self, kapasitas_baterai):
        self.__kapasitas_baterai = kapasitas_baterai

    # Override metode info untuk menyertakan kapasitas baterai
    def info(self):
        return f"{super().info()} - Kapasitas Baterai:
{self.__kapasitas_baterai} kWh"

    # Implementasi metode abstrak nyalakan_mesin
    def nyalakan_mesin(self):
        return f"Mesin listrik {self.get_merk()} {self.get_model()} menyala
dengan kapasitas baterai {self.__kapasitas_baterai} kWh!"

    # Implementasi metode abstrak matikan_mesin
    def matikan_mesin(self):
        return f"Mesin listrik {self.get_merk()} {self.get_model()} mati."
```

Pada File Lain

```
# Fungsi untuk menampilkan informasi dan menyalakan mesin mobil
def tampilkan_info_dan_nyalakan_mesin(mobil):
    print(mobil.info())
    print(mobil.nyalakan_mesin())

# Buat objek dari kelas MobilSport
mobil_sport1 = MobilSport("Ferrari", "488", 2021, 330)
# Buat objek dari kelas MobilListrik
mobil_listrik1 = MobilListrik("Tesla", "Model S", 2022, 100)

# Menggunakan fungsi yang sama untuk objek yang berbeda (Polimorfisme)
tampilkan_info_dan_nyalakan_mesin(mobil_sport1)
tampilkan_info_dan_nyalakan_mesin(mobil_listrik1)
```

TUGAS

Tugas kalian hanya mengulangi semua kode diatas dengan 2 file untuk setiap pilar dari OOP.

Contoh format: untuk kodingan pertama (contoh simple implementasi OOP)

1. MobileOOP.py
2. MainOOP.py

Pastikan semua code telah dirun dan menghasilkan outpun sempurna, untuk file dengan output yang salah atau error, tidak akan dihitung telah mengerjakan pilar tersebut.