MODUL PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

Minggu 3

ABSTRAKSI DAN ENKAPSULASI (VISIBILITAS FUNGSI DAN VARIABEL, RELASI ANTAR KELAS)

Disusun Oleh :

Andhika Putra Pratama	119140224
Andhika Wibawa B.	119140218
Nurul Aulia Larasati	119140088
Ihza Fajrur Rachman H.	119140130
Enrico Johanes S.	119140021
M. Ammar Fadhila R.	119140029

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK ELEKTRO, INFORMATIKA DAN SISTEM FISIS

INSTITUT TEKNOLOGI SUMATERA

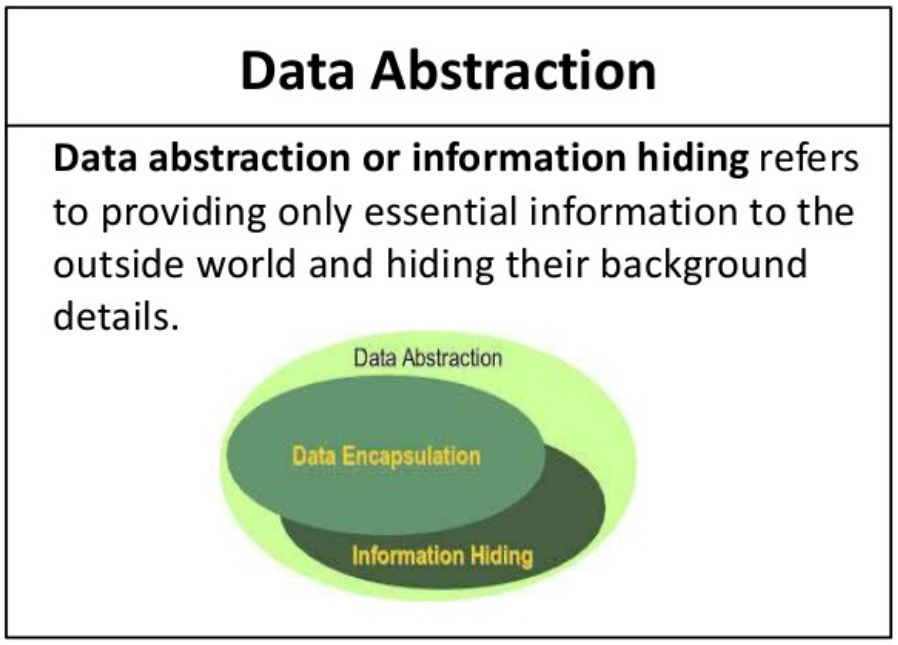
2022

1. Abstraksi

Abstraksi adalah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user. gunanya untuk mengurangi kompleksitas.

User mengetahui apa yang objek lakukan, tapi tidak tau mekanisme yang terjadi di belakang layar bagaimana. Contohnya ketika mengendarai mobil, user mengetahui bagaimana menyalakan mobil, menjalankan, menghentikan, dll, tetapi tidak mengetahui mekanisme apa yang terjadi pada mobil ketika mendapat perintah di atas.

Ketika mendefinisikan kelas, sebenarnya sedang membuat Abstrak dari suatu Objek. Kelas merupakan bentuk abstrak atau cetak biru (blue print) dari suatu objek nyata. Wujud nyata suatu kelas dinamakan instance (Objek). Contoh : Str adalah kelas, sedangkan teks “Python” adalah objek.



2. Enkapsulasi (*Encapsulation*)

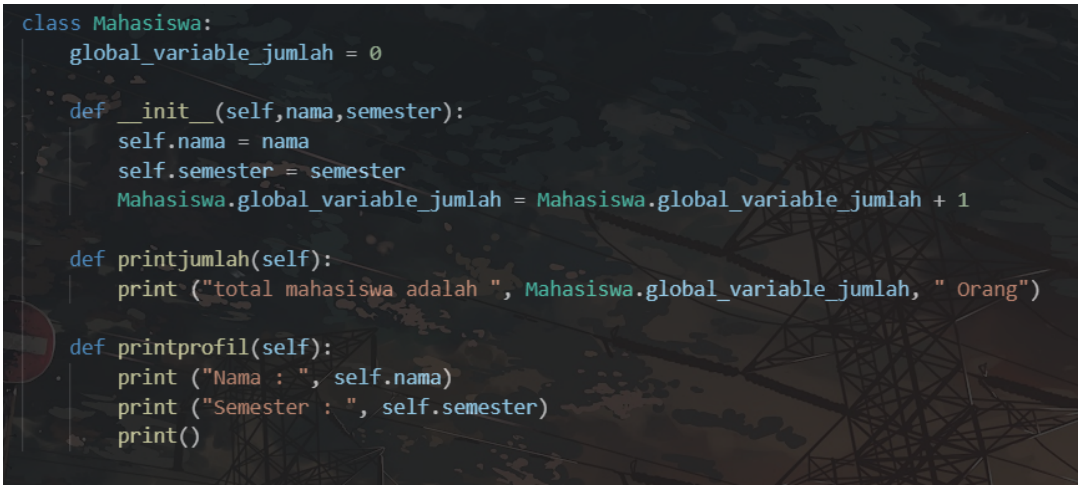
Enkapsulasi adalah sebuah metode yang digunakan untuk mengatur struktur kelas dengan cara menyembunyikan alur kerja dari kelas tersebut. Yang dimaksud dengan struktur kelas tersebut adalah property dan method. Dengan konsep ini, kita dapat "menyembunyikan" property dan method dari suatu kelas agar hanya property dan method tertentu saja yang dapat diakses dari luar kelas. Dengan menghalangi akses dari luar kelas tersebut, elemen penting yang terdapat dalam kelas dapat lebih terjaga, dan menghindari kesalahan jika elemen tersebut diubah secara tidak sengaja.

Abstraksi adalah cara untuk menyembunyikan informasi yang tidak dibutuhkan, sedangkan enkapsulasi adalah cara menyembunyikan atribut suatu entitas serta metode untuk melindungi informasi dari luar. Untuk membatasi hak akses terhadap property dan method dalam suatu kelas, terdapat 3 jenis access modifier yang terdapat dalam python, yaitu public access, protected access, dan private access.

- Public Access Modifier

  Pada umumnya ketika kita mendeklarasikan suatu variabel atau method, maka itulah public access modifier. Setiap class, variable dan method yang dibuat secara default merupakan public.

  Contoh :



```
class Mahasiswa:
    global_variable_jumlah = 0

    def __init__(self,nama,semester):
        self.nama = nama
        self.semester = semester
        Mahasiswa.global_variable_jumlah = Mahasiswa.global_variable_jumlah + 1

    def printjumlah(self):
        print ("total mahasiswa adalah ", Mahasiswa.global_variable_jumlah, " Orang")

    def printprofil(self):
        print ("Nama : ", self.nama)
        print ("Semester : ", self.semester)
        print()
```

- Protected Access Modifier

  Jika suatu variabel dan method dideklarasikan secara protected, maka variable dan method tersebut hanya dapat diakses oleh kelas turunan darinya. Cara mendeklarasikannya dengan menambahkan 1 underscore (_) sebelum variable atau method.

  Contoh :

```
class Mobil:
    #protected variable
    _merk = None
    _warna = None

    #constructor
    def __init__(self,merk,warna):
        self._merk = merk
        self._warna = warna

    #protected method
    def _tampilMobil(self):
        print("Merk Mobil : ", self._merk)
        print("Warna Mobil : ", self._warna)
```

- Private Access Modifier

  Jika suatu variabel dan method dideklarasikan secara private, maka variable dan method tersebut hanya dapat diakses di dalam kelas itu sendiri, private access merupakan yang paling aman. Dalam mendeklarasikannya, hanya perlu menambahkan double underscore (__) sebelum nama variable dan methodnya.

  Contoh :

```python
class Mobil:
    #private variable
    __merk = None
    __warna = None

    #constructor
    def __init__(self,merk,warna):
        self.__merk = merk
        self.__warna = warna

    #private method
    def _tampilMobil(self):
        print("Merk Mobil : ", self.__merk)
        print("Warna Mobil : ", self.__warna)
```

- Setter dan Getter

Seperti yang sudah dijelaskan pada minggu yang lalu, setter adalah sebuah method yang digunakan untuk mengatur sebuah property yang ada di dalam suatu kelas/objek. Sedangkan getter adalah sebuah method yang digunakan untuk mengambil nilai dari suatu property. Dikarenakan property dengan access modifier private hanya dapat diakses dari dalam kelas tersebut, dengan metode inilah kita dapat mengaksesnya dari luar kelas. Terdapat 2 cara untuk membuat setter dan getter, yaitu dengan tanpa decorator dan dengan decorator.

- Tanpa Decorator

```python
class siswa:
    def __init__(self, umur = 0):
        self._umur = umur

    # getter method
    def get_umur(self):
        return self._umur

    # setter method
    def set_umur(self, x):
        self._umur=x

raj = siswa()

# setting the umur using setter
raj.set_umur(19)
# retrieving umur using getter
print(raj.get_umur())
print(raj._umur)
```

Output :

```
PS D:\itera\Asprak PBO 2022>
Asprak PBO 2022/mg 2.py"
19
19
PS D:\itera\Asprak PBO 2022>
```

- Dengan Decorator

```python
class Siswa:
    def __init__(self, nama, umur = 15):
        self.__nama = nama
        self.__umur = umur

    @property
    def umur(self):
        print("Fungsi getter umur dipanggil")
        return self.__umur

    @umur.setter
    def umur(self, x):
        print("Fungsi setter umur dipanggil")
        self.__umur = x

Bambang = Siswa("Bambang")

# Akan error karena bersifat private
#print(Bambang.__umur)

# Gunakan fungsi property decorator
print(Bambang.umur)
Bambang.umur += 5
print(Bambang.umur)
```

Output:

```
Fungsi getter umur dipanggil
15
Fungsi getter umur dipanggil
Fungsi setter umur dipanggil
Fungsi getter umur dipanggil
20
```

Perlu diketahui bahwa penggunaan setter getter dengan decorator hanya dapat dilakukan terhadap properti dengan access modifier protected dan private. Jika dilakukan terhadap properti yang bersifat public (dengan nama yang sama), maka akan mengakibatkan error

Object

Membuat Instance Objects

Disini terdapat suatu kelas mahasiswa seperti berikut :

```
class Mahasiswa:
    global_variable_jumlah = 0

    def __init__(self,nama,semester):
        self.nama = nama
        self.semester = semester
        Mahasiswa.global_variable_jumlah = Mahasiswa.global_variable_jumlah + 1

    def printjumlah(self):
        print ("total mahasiswa adalah ", Mahasiswa.global_variable_jumlah, " Orang")

    def printprofil(self):
        print ("Nama : ", self.nama)
        print ("Semester : ", self.semester)
        print()
```

Untuk membuat instance dari kelas yang telah dibuat dapat dilakukan dengan menggunakan nama dari class kemudian argumen diterima oleh metode init.

Bentuk umum dari instansiasi objek yaitu

```
nama_objek = Mahasiswa(name,smster)
```

Berikut ini adalah contoh membuat objek dari kelas mahasiswa

```
maha1 = Mahasiswa("Tama",6)
maha2 = Mahasiswa("Lucia",4)
```

Mengakses Atribut Objek

Dari objek yang telah dibuat dapat dilakukan pengaksesan atribut dari objek dengan menggunakkan operator dot(titik)

Sintaks umumnya yaitu

```
namaObjek.atribut
```

Berikut contoh untuk mengakses atribut dari kelas Mahasiswa

```
maha1.printprofil()
maha2.printprofil()

print ("Jumlah total mahasiswa : ", Mahasiswa.global_variable_jumlah)
```

Output yang dihasilkan :

```
Nama :  Tama
Semester :  6

Nama :  Lucia
Semester :  4

Jumlah total mahasiswa :  2
```

Menambah, menghapus, dan Mengubah atribut object

Objek yang sudah dibuat dapat dimodifikasi seperti ditambah, dihapus, ataupun diubah atributnya. Misal ingin mengubah semester dari mahasiswa 3, karena ternyata mahasiswa 3 sudah semester 6. Maka dapat dilakukan seperti berikut:

```
maha3 = Mahasiswa("Inu",3)

print("Sebelum diubah")
maha3.printprofil()

maha3.semester = 6

print("Setelah diubah")
maha3.printprofil()
```

Maka output yang dihasilkan

```
Sebelum diubah
Nama :  Inu
Semester :  3

Setelah diubah
Nama :  Inu
Semester :  6
```

Atau jika kita ingin menghapus objek dari mahasiswa 3 karena mahasiswa 3 sudah DO, maka seperti berikut :

```
8   print ("Mahasiswa 3 di-DO")
9   del maha3.semester
0   maha3.printprofil()
```

del merupakan fungsi yang menghapus atribut semester dari maha3. Sehingga pada output terlihat bahwa saat pemanggilan method tampil_profil kedua setelah atribut dihapus terjadi error.

Cara yang lebih baik untuk memodifikasi atribut dari suatu objek dapat menggunakkan fungsi berikut :

- **getattr(obj, name[, default])** – Mengakses atribut dari objek
- **hasattr(obj, name)** – Memeriksa apakah suatu objek memiliki atribut tertentu
- **setattr(obj, name, value)** – Mengatur nilai atribut. Jika ternyata atribut tidak ada, maka atribut tersebut akan dibuat.
- **delattr(obj, name)** – Menghapus atribut dari suatu objek.

Contoh :

```
#mendapatkan value semester dari mahasiswa 1
print ("Dapatkan nilai atribut semester mahasiswa 1 = ", getattr(maha1,'semester'))
#memastikan apakah mahasiswa 2 memiliki atribut nama
print ("Apakah mahasiswa 2 memiliki atribut nama = ",hasattr(maha2,'nama'))
#mengubah value atribut dari mahasiswa 3
print ("Nama mahasiswa 3 sebelum diubah = ",maha3.nama)
setattr(maha3,'nama','Taku')
print ("Mahasiswa 3 setelah diganti namanya = ",maha3.nama)
#menghapus atribut semester mahasiswa 3
delattr(maha3,'semester')
```

Output yang dihasilkan

```
Dapatkan nilai atribut semester mahasiswa 1 =  6
Apakah mahasiswa 2 memiliki atribut nama =  True
Nama mahasiswa 3 sebelum diubah =  Inu
Mahasiswa 3 setelah diganti namanya =  Taku
```

Referensi

https://www.askpython.com/python/oops/abstraction-in-python

https://www.geeksforgeeks.org/abstract-classes-in-python/

https://ichi.pro/id/python-untuk-pemula-pemrograman-berorientasi-objek-65021974666569