BAB 3

Encapsulation

3.1 Tujuan

1. Dapat memahami konsep encapsulation dan abstraction pada Python
2. Dapat mengimplementasikan encapsulation dan abstraction dengan menggunakan bahasa pemrograman Python

3.2 Pengantar

3.2.1 Encapsulation

Encapsulation merupakan salah satu konsep penting dalam pemrograman berorientasi objek (OOP) yang berfungsi untuk menyembunyikan detail implementasi suatu objek dari pengguna, dan hanya mengekspos metode publik atau antarmuka publik yang telah ditentukan. Dengan menggunakan encapsulation, pengguna hanya dapat berinteraksi dengan objek melalui metode publik yang telah didefinisikan, sehingga tidak dapat secara langsung memodifikasi atau mengakses data atau metode privat yang tersembunyi dari objek tersebut. Dengan cara ini, encapsulation dapat membantu memperkuat keamanan dan memudahkan penggunaan objek dalam kode yang lebih bersih dan terstruktur. Perhatikan contoh kode berikut

```
1. class Person():
2.     def __init__(self, name, salary):
3.         self.name = name
4.         self.salary = salary
5.
6.
7. oPerson1 = Person('Joe Schmoe', 90000)
8. oPerson2 = Person('Jane Smith', 99000)
9.
10.#akses nilai variabel salary secara langsung
11.print(oPerson1.salary)
12.print(oPerson2.salary)
13.
14.#mengubah nilai variabel salary secara langsung
15.oPerson1.salary = 100000
16.oPerson2.salary = 111111
17.
18.print(oPerson1.salary)
19.print(oPerson2.salary)
```

Kode diatas merupakan sebuah class Person dengan atribut name dan salary. Baris 7 dan 8 merupakan pembuatan object dengan nama oPerson1 dan oPerson2. Perhatikan Baris 11, 12 dan Baris 15,16, baris tersebut merupakan penulisan kode untuk mengakses nilai variable salary dan mengubah nilai dari variable salary.

Sebenarnya cara tersebut dapat berjalan pada python, namun penulisan kode tersebut menyalahi kaidah encapsulation dimana kita tidak diperbolehkan mengakses data atau mengubah data secara langsung. Praktek tersebut harus dihindari karena menimbulkan beberapa masalah seperti :

- Mengurangi keamanan data: Jika variabel OOP yang bersifat private diakses secara langsung, maka data tersebut tidak lagi terlindungi dari modifikasi yang tidak sah atau akses yang tidak diizinkan. Hal ini dapat mengurangi keamanan data, terutama jika data tersebut penting atau rahasia.
- Memperburuk modularitas: Jika variabel OOP yang bersifat private diakses secara langsung, maka hal tersebut dapat memperburuk modularitas kode, karena memungkinkan satu kelas untuk bergantung pada detail implementasi kelas lain. Hal ini dapat memperumit pengembangan kode, pemeliharaan, dan pengujian.
- Meningkatkan risiko kesalahan: Akses langsung pada variabel OOP yang bersifat private dapat meningkatkan risiko kesalahan, karena memungkinkan penggunaan yang tidak sah atau modifikasi yang tidak diinginkan. Hal ini dapat menyebabkan bug yang sulit ditemukan dan memperburuk kualitas kode.

3.2.2 Private Variable

Untuk melindungi variable dari akses secara langsung maka kita dapat menggunakan variable private. Variabel Private adalah variabel yang didefinisikan dengan dua tanda underscore (__) di depan nama variabel. Variabel private hanya dapat diakses dan dimodifikasi di dalam kelas tempat variabel tersebut didefinisikan.

```
1. class Person():
2.     def __init__(self, name, salary):
3.         self.__name = name #variabel private
4.         self.__salary = salary #variabel private
5.
6.
7. oPerson1 = Person('Joe Schmoe', 90000)
8. oPerson2 = Person('Jane Smith', 99000)
```

```
9.
10.#akses nilai variabel salary secara langsung
11.print(oPerson1.salary)
```

Perhatikan baris ke 3 dan 4 potongan kode diatas. Kita mengubah variable name dan salary menjadi private dengan menambahkan dua tanda underscore. Sehingga Ketika kode tersebut dijalankan maka akan memunculkan pesan error seperti berikut

```
AttributeError: 'Person' object has no attribute 'salary'
```

3.2.3 Getter dan Setter

Lalu bagaimana cara mengakses dan memodifikasi variable private tersebut? Salah satu caranya kita dapat membuat method baru yang berfungsi untuk melakukan aksi pada variable tersebut. Nama method ini dinamakan dengan getter dan setter. Getter digunakan untuk mengambil nilai dari variabel, sedangkan setter digunakan untuk mengatur nilai variabel.

Getter biasanya memiliki nama yang sama dengan variabel yang ingin diambil nilainya, dengan tambahan kata "get" di depannya, sedangkan setter biasanya memiliki nama yang sama dengan variabel yang ingin diatur nilainya, dengan tambahan kata "set" di depannya. Contohnya, jika kita memiliki variabel "nama" pada suatu kelas, maka getter untuk variabel tersebut dapat diberi nama "getNama", sedangkan setter dapat diberi nama "setNama".

```
1. class Person():
2.     def __init__(self, name, salary):
3.         self.__name = name
4.         self.__salary = salary
5.
6.     def getName(self):
7.         return self.__name
8.
9.     def setName(self, name):
10.        self.__name = name
```

```
11.
12.     def getSalary(self):
13.         return self.__salary
14.
15.     def setSalary(self, salary):
16.         self.__salary = salary
17.
18.
19.#Membuat objek oPerson1
20.oPerson1 = Person('Joe Schmoe', 90000)
21.
22.#Mengakses variabel salary dengan method getSalary()
23.print(oPerson1.getSalary())
24.
25.#mengubah nilai variabel name dengan method setName()
26.oPerson1.setName('Joe Taslim')
27.
28.#mengubah nilai variabel salary dengan method setSalary()
29.oPerson1.setSalary(100000)
30.
31.#mengakses variabel name dan salary
32.print(oPerson1.getName())
33.print(oPerson1.getSalary())
```

Ada beberapa alasan mengapa kita harus menggunakan konsep encapsulation dalam pemrograman berorientasi objek:

- Membantu memperkuat keamanan: Encapsulation membuat atribut privat tidak dapat diakses secara langsung dari luar Class, sehingga memperkuat keamanan data dan mencegah penggunaan yang tidak sah.
- Memudahkan perubahan dan pemeliharaan kode: Dengan menggunakan encapsulation, kita dapat mengubah implementasi dari suatu class tanpa memengaruhi penggunaannya di luar class. Hal ini akan memudahkan pemeliharaan kode dan memperkuat struktur kode.
- Meningkatkan modularitas: Encapsulation memungkinkan class-class untuk saling berinteraksi melalui antarmuka publik tanpa harus mengetahui detail

implementasi satu sama lain. Hal ini memperkuat modularitas kode dan memungkinkan untuk pengembangan dan pengujian yang lebih efisien.

- Membantu mempermudah debugging: Dengan menggunakan encapsulation, kita dapat membatasi akses ke bagian kode tertentu, sehingga membantu mempermudah debugging dan memperkuat kesalahan yang terjadi di dalam class.

3.2.4 Abstraction

Abstraction merupakan konsep OOP yang masih berhubungan dengan encapsulation. Abstraction merupakan cara untuk mengelola kompleksitas dengan cara menyembunyikan hal-hal yang detail. Abstraction sering diimplementasikan dalam kehidupan sehari-hari. Sebagai contoh saat kita mengendarai mobil, jika kita ingin berjalan maju maka kita cukup menginjak pedal gas. Kita tidak perlu mengetahui bagaimana hubungan pedal gas dengan cara kerja mesin sehingga mampu menggerakkan roda untuk bergerak. Yang ada pada pikiran kita selama kita injak pedal gas, maka mobil harus maju. Hal-hal detail yang bekerja di belakang tidak perlu kita pikirkan.

Perbedaan antara encapsulation dan abstraction adalah sebagai berikut

- Encapsulation (enkapsulasi) adalah konsep untuk menyembunyikan rincian implementasi suatu objek dari pengguna objek tersebut dan hanya mengekspos metode publik atau antarmuka publik. Dengan cara ini, pengguna objek hanya dapat berinteraksi dengan objek melalui metode publik yang telah ditentukan, dan tidak dapat memodifikasi atau mengakses data atau metode privat yang tersembunyi dari mereka.
- Abstraction (abstraksi) adalah konsep untuk memusatkan perhatian pada informasi yang penting dan mengabaikan rincian yang tidak penting atau tidak perlu diketahui oleh pengguna

Konsep abstraction dapat kita lihat pada kode implementasi stack pada python

```
#inisialisasi variabel mystack
myStack = []

#insert data menggunakan fungsi append
myStack.append('a')
myStack.append('b')
myStack.append('c')

#print isi variabel mystack
print(myStack)

#hapus data menggunakan fungsi pop
myStack.pop()
print(myStack)

#hapus data menggunakan fungsi pop
myStack.pop()
print(myStack)

#hapus data menggunakan fungsi pop
myStack.pop()
print(myStack)
```

pada kode diatas kita menggunakan fungsi append dan pop. Append digunakan untuk menambah data pada stack sedangkan pop digunakan untuk menghapus data pada stack. Kita tidak perlu memikirkan bagaimana detail fungsi tersebut bekerja. Yang perlu kita pikirkan inputan dan bagaimana hasil dari fungsi tersebut Ketika dijalankan.

3.3 Kegiatan Praktikum

3.3.1 Kegiatan 1 : Implementasi Private Variabel

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
class Car:
    def __init__(self, make, model, year):
        self.__make = make
        self.__model = model
        self.__year = year
```

```
6.
7.      def get_make(self):
8.          return self.__make
9.
10.     def get_model(self):
11.         return self.__model
12.
13.     def get_year(self):
14.         return self.__year
15.
16.car = Car("Honda", "Civic", 2022)
17.print(car.get_make())
18.print(car.get_model())
19.print(car.get_year())
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

3.3.2 Kegiatan 2 : Implementasi Getter dan Setter

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
1. class Person:
2.     def __init__(self, name, age):
3.         self.__name = name
4.         self.__age = age
5.
6.     def get_name(self):
7.         return self.__name
8.
9.     def set_name(self, name):
10.        self.__name = name
11.
12.    def get_age(self):
13.        return self.__age
14.
15.    def set_age(self, age):
16.        self.__age = age
17.
18.person = Person("Abdul Hakim", 25)
19.print(person.get_name())
20.print(person.get_age())
21.
22.person.set_name("Ryan Hakim")
23.person.set_age(30)
```

```
24.
25.print(person.get_name())
26.print(person.get_age())
27.
28.
29.print(person.__name)
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

3.3.3 Kegiatan 3 : Abstraction pada OOP

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
1. class Stack:
2.     def __init__(self):
3.         self.items = []
4.
5.     def is_empty(self):
6.         return self.items == []
7.
8.     def push(self, item):
9.         self.items.append(item)
10.
11.    def pop(self):
12.        return self.items.pop()
13.
14.    def peek(self):
15.        return self.items[len(self.items) - 1]
16.
17.    def size(self):
18.        return len(self.items)
19.
20.
21.s = Stack()
22.print(s.is_empty())
23.
24.s.push(1)
25.s.push(2)
26.s.push(3)
27.print(s.peek())
28.print(s.size())
29.
30.s.pop()
```

```
31.print(s.peek())
32.print(s.size())
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

3.4 Tugas

1. Buatlah sebuah class Employee dengan atribut private *__name, __age,* dan *__salary*. Class tersebut harus memiliki metode *set_name(), set_age(), set_salary(), get_name(), get_age(),* dan *get_salary()* yang memungkinkan pengguna untuk mengakses dan memodifikasi nilai dari atribut private tersebut. Buatlah satu object berdasar class tersebut dan implementasikan semua method yang ada.