Resume Praktikum PBO Pertemuan 5

Enkapsulasi

Sama seperti abstraksi, enkapsulasi juga merupakan konsep yang penting dalam pemrograman berbasis objek (bahasa pemrograman python). Pada intinya, Enkapsulasi membantu kita untuk menyembunyikan sebagian kode/data yang tidak butuh untuk ditujukan ke pengguna (dengan alasan pengamanan dan penampilan program) dan juga menampilkan kode/data yang butuh untuk ditampilkan kepada pengguna. Untuk implementasinya sendiri, enkapsulasi dilakukan dengan suatu proses dimana kita menciptakan suatu kelas yang isinya merupakan atribut dan metode. Beda dengan abstraksi yang mengoperasikan masalah-masalah di tingkat desain, enkapsulasi membantu kita dalam mengoperasikan masalah-masalah di tingkat implementasi.

Untuk mengimplementasikan konsep enkapsulasi, kita cukup menempatkan variabel dan metode dalam suatu kelas.

Contoh kode enkapsulasi:

```
class Mobil:
      def __init__(self, gears_ratio, machine):
            self.nama = "Mitsubishi Evo Lancer 2020"
            self._gears_ratio = gears_ratio #protected variables (_<variableName>)
            self.__machine = machine #private variables (__<variableName>)

      def __str__(self):
        return "Mobil ini merupakan mobil pabrik, bukan mobil second"
```

Selanjutnya, terdapat Getters dan Setters yang membantu programmer dalam menentukan data-data yang terenkapsulasi. Getter merupakan metode dalam suatu kelas yang dapat menambahkan validasi suatu nilai, sedangkan setter dapat menetapkan suatu nilai.

Contoh kode dari Getters dan Setters:

```
def contoh_setter(self):
      return self.contoh_variabel = variabel

    def contoh_getter(self):
      return self.contoh_variabel
```

Abstraksi

Abstraksi merupakan konsep yang sering digunakan dalam pemrograman berbasis objek (terkhususnya untuk python). Pada dasarnya, abstraksi merupakan suatu teknik untuk mengekstrak atau mengambil data penting tentang suatu item/objek. Selain itu, abstraksi juga dapat membantu kita dalam menyembunyikan data yang tidak perlu ditunjukan kepada pengguna/pelanggan sewaktu mereka menggunakan perangkat lunak yang kita kembangkan. Kita juga dapat menampilkan data yang relevan dan juga mengoperasikan masalah-masalah di tingkat desain (tahap planning untuk suatu perangkat lunak).

Untuk membuat suatu atribut atau metode kelas bersifat private dengan menempatkan dua underscore sebelum atribut atau metode tersebut (contoh : __gears_ratio). Jika ingin membuat atribut atau metode dengan sifat protected, maka tempatkan satu underscore sebelum metode dan atribut (contoh : _gears_ratio). Apa perbedaan antara protected dan private? Objek-objek yang bersifat protected hanya bisa digunakan dalam kelas tersebut dan turunannya, sedangkan objek-objek yang bersifat private hanya bisa digunakan dalam sub-kelas atau kelas tersebut sendiri.

Contoh kode enkapsulasi :

```
class Mobil:
      def __init__(self, gears_ratio, machine):
            self.nama = "Mitsubishi Evo Lancer 2020"
            self._gears_ratio = gears_ratio #protected variables (_<variableName>)
            self.__machine = machine #private variables (__<variableName>)
```

Inheritance

Secara garis besar, inheritance merupakan teknik yang digunakan untuk menurunkan ciri-ciri dari suatu kelas kepada kelas lain (dimana relasinya berupa kelas *parent* dan kelas *child*). Dengan ini kita dapat menurunkan metode-metode (fungsi) dan atribut suatu kelas kepada kelas-kelas lain yang memiliki ciri khas sama. Di sisi lain, kelas anak (*child*) juga dapat memiliki metode ataupun atributnya sendiri, berbeda dengan kelas *parent*nya.

Contoh kode yang menggunakan konsep inheritance :

```
class Komputer:
    def __init__(self,nama,jenis,harga,merk) -> None:
        self.nama = nama
        self.jenis = jenis
        self.harga = harga
        self.merk = merk

class Processor(Komputer):
    def __init__(self, nama, jenis, harga, merk,jumlah_core,kecepatan_processor)
```

```
        super().__init__(nama, jenis, harga, merk)
        self.nama = "Processor"
        self.jumlah_core = jumlah_core
        self.kecepatan_processor = kecepatan_processor
    def __str__(self):
        return f"{self.nama} ini merupakan tipe {self.jenis} yang dibuat oleh
{self.merk}"

class RAM(Komputer):
    def __init__(self, nama, jenis, harga, merk,ram_capacity) -> None:
        super().__init__(nama, jenis, harga, merk)
        self.nama = "RAM"
        self.ram_capacity = ram_capacity
    def __str__(self):
        return f"{self.nama} ini memiliki kapasitas {self.jenis} yang dibuat oleh
{self.merk}"
```

Pada contoh program yang dapat dilihat di atas, kita dapat melihat bahwa kelas Processor dan Kelas RAM merupakan kelas turunan dari kelas Komputer. Melalui inheritance, metode dan atribut yang terdapat pada kelas Komputer dapat diturunkan ke kelas-kelas lain (yang kita akan sebut sebagai kelas child/ kelas turunan). Selain itu, kita juga dapat mengoptimasi kode dari program kita dengan menghilangkan bagian-bagian yang *redundant* layaknya seperti membuat fungsi untuk melakukan suatu aksi namun untuk objek.

Polymorphism

Polymorphism merupakan kemampuan dimana sekumpulan objek dalam suatu program dapat melakukan hal yang berbeda walaupun memiliki nama yang sama. Layaknya suatu pensil yang dapat digunakan untuk menulis kabar baik, namun pada situasi yang berbeda pensil tersebut juga dapat digunakan untuk menyebarkan berita palsu atau negatif. Dalam kata lain, dengan *polymorphism,* kita dapat memodifikasi cara perilaku dari suatu objek dalam program kita.

Contoh kode *polymorphism* :

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof"
```

```
class Cat(Animal):
    def speak(self):
        return "Meow"

class Cow(Animal):
    def speak(self):
        return "Moo"

def animal_speak(animal):
    print(animal.speak())

dog = Dog("Buddy")
cat = Cat("Whiskers")
cow = Cow("Betsy")

animal_speak(dog)
animal_speak(cat)
animal_speak(cow)
```

Pada contoh program di atas, kita dapat melihat bahwa kelas Dog, Cat, dan Cow merupakan kelas turunan dari kelas Animal yang memiliki metode “speak”. Perhatikan bahwa tiap kelas turunan memiliki output yang berbeda untuk metode “speak” masing-masing. Cat sewaktu memanggil metode “speak’ akan memberi keluaran “Meow”, Cow akan memberikan keluaran “Moo”, dan Dog akan memberi keluaran “Woof”. Pada kasus ini, kita dapat melihat bahwa metode “speak” merupakan contoh dari konsep *polymorphism* dalam pemrograman berbasis objek.