Nama : Winnerson Laia

NIM : 121140121

Kelas PBO siakad : RC

Kelas Praktikum PBO : RB

RESUME DASAR PEMROGRAMAN BERORIENTASI OBJEK

Pemrograman Berorinetasi Objek (PBO) adalah salah salah satu paradigma pemrograman berdasarkan konsep "objek" yang dapat berisi data atau yang kita kenal sebagai atribut. Terdapat beberapa bahasa pemrograman mendukung konsep PBO seperti c++, java, python, dan lain-lain. Namun ada juga bahasa pemrograman yang tidak mendukung konsep PBO, biasanya bahasa pemrograman tidak rendah seperti bahasa pemrograman c dan assembly. Dalam PBO terdapat empat dasar konsep yaitu enkapsulasi, abstraksi, inheritance, dan polimorfisme. Berikut penjelasannya ke empat konsep tersebut :

a. ENKAPSULASI

Enkapsulasi adalah suatu cara untuk melindungi atau menyembunyikan atribut tertentu dari suatu objek dengan memanfaatkan access modifier. Tujuan umum dari enkapsulasi adalah untuk menghindari akses atau pengubahan nilai atribut yang tidak disengaja.

Access modifier adalah satu suatu kata kunci dalam PBO yang digunakan untuk mengatur aksesibilitas suatu atribut kelas. Access modifier ada 3 yaitu public, privat, dan protected. Jika kita menjadikan suatu atiribut public maka atribut tersebut dapat diakses dari luar kelas. Jika kita menjadikan suatu atribut privat maka atribut tersebut hanya dapat diakses di dalam kelas itu saja. Jika kita menjadikan suatu atibut protected maka atribut tersebut hanya dapat diakses dari dalam kelas itu sendiri dan kelas turunanya.

Semisal kita ingin membuat suatu kelas *Mahasiswa* yang dimana memiliki atiribut nama serta umur. Kita tidak ingin terjadi perubahan secara tidak sengaja terhadap atribut nama dan umur, maka kita dapat menggunakan konsep enkapsulasi.

Contoh penerapan enkapsulasi :

```python
class Mahasiswa :
    def __init__(self, nama, nim) -> None:
        self.__nama = nama
        self.__nim  = nim

    def tampilkan_nama(self) :
        return self.__nama

    def ganti_nama(self, nama_baru) :
        self.__nama = nama_baru
        print("Nama mahasiswa telah diganti !")

    def tampilkan_nim(self) :
        return self.__nim

    def ganti_nim(self, nim_baru) :
        self.__nim = nim_baru
        print("NIM mahasiswa telah diganti !")


# Main Program

mahasiswa = Mahasiswa("Jedy", 123456789)

print(f"\nData Mahasiswa")
print(f"Nama    : {mahasiswa.tampilkan_nama()}")
print(f"NIM     : {mahasiswa.tampilkan_nim()} \n")

mahasiswa.ganti_nama("Risa")
mahasiswa.ganti_nim(123456000)

print(f"\nData Mahasiswa")
print(f"Nama    : {mahasiswa.tampilkan_nama()}")
print(f"NIM     : {mahasiswa.tampilkan_nim()} \n")
```

b. ABSTRAKSI

Pada suatu waktu, kita ingin membuat beberapa class yang hampir mirip dan memiliki fungsi/method yang sama kegunaanya namun memiliki definisi yang berbeda disetiap kelasnya. Jika membuat class satu persatu dan memdefinsikan fungsi/methodnya satu persatu maka akan membuat kode program terlihat tidak terstuktur. Solusinya kita dapat menggunakan kelas yang bersifat abstrak (kelas abstrak).

Kelas abstrak sendiri adalah kelas yang fungsi/methodnya belum memiliki implementasi, namun implemtasinya baru dilakukan pada kelas turunan dari kelas abstrak. Kelas abstrak juga dapat memiliki konstruktor namun tidak dapat dibuat objeknya. Dalam bahasa pemrograman python, kita dapat menggunakan abstraksi dengan menggunakan modul abc (abstract base class).

Contoh penerapan abstraksi :

```python
from abc import ABC
from abc import abstractmethod

class BangunDatar:
    def __init__(self, nama_bangun) -> None:
        self.nama_bangun = nama_bangun

    @abstractmethod
    def luas(self):
        pass

    @abstractmethod
    def keliling(self):
        pass

class Persegi(BangunDatar) :
    def __init__(self, panjang, lebar) -> None:
        super().__init__("Persegi")
        self.panjang = panjang
        self.lebar = lebar

    def luas(self):
        return self.panjang * self.lebar

    def keliling(self):
        return 2 * (self.panjang + self.lebar)

    def __str__(self) -> str:
        return f"\nPersegi \nLuas = {self.luas()} \nKeliling = {self.keliling()} \n"

class Lingkaran(BangunDatar) :
    def __init__(self, jari_jari) -> None:
        super().__init__("Lingkara")
        self.jari_jari = jari_jari

    def luas(self):
        return 3.14 * self.jari_jari * self.jari_jari

    def keliling(self):
        return 2 * 3.14 * self.jari_jari

    def __str__(self) -> str:
        return f"\nLingkaran \nLuas = {self.luas()} \nKeliling = {self.keliling()} \n"


# Main Program

persegi = Persegi(4, 3)
lingkaran  = Lingkaran(7)

print(persegi)
print(lingkaran)
```

c. INHERITENCE

Inheritance sendiri sendiri pewarisan sifat dari kelas induk ke kelas tururan. Dalam PBO inheritance berarti seluruh atribut dan fungsi yang bersifat public dan protected yang dimiliki oleh kelas induk akan dimiliki oleh kelas turunannya. Contoh inheritance/pewarisan paling sederhana dalam kehidupan sehari-hari adalah hubungan antar orangtua dan anak, dimana anak memiliki sifat-sifat tertentu dari orangtuanya.

Contoh penerapan inheritance :

```python
class Manusia : # kelas induk
    def __init__(self, nama, umur) -> None:
        self.nama = nama
        self.umur = umur

class Dosen(Manusia) : # kelas turunan
    def __init__(self, nama, umur, nidn) -> None:
        super().__init__(nama, umur)
        self.nidn = nidn

    def tampilkan_data_dosen(self) :
        return f"\nData Dosen \nNama : {self.nama} \nNIDN : {self.nidn} \nUmur : {self.umur} \n"

class Mahasiswa(Manusia) : # kelas turunan
    def __init__(self, nama, umur, nim) -> None:
        super().__init__(nama, umur)
        self.nim = nim

    def tampilkan_data_mahasiswa(self) :
        return f"\nData Mahasisiwa \nNama : {self.nama} \nNIM : {self.nim} \nUmur : {self.umur} \n"


# Main Program
dosen = Dosen("Kira", 29, 100100100)
mahasiswa = Mahasiswa("Budi", 19, 123456789)

print(dosen.tampilkan_data_dosen())
print(mahasiswa.tampilkan_data_mahasiswa())
```

Dari contoh diatas dapat kita lihat bahwa kelas *Dosen* dan *Mahasiswa* merupakan kelas turuan dari kelas *Manusia sehingga* kelas *Dosen* dan *Mahasiswa* juga memiliki atribut yang dimiliki kelas *Manusia* yaitu atribut nama dan umur.

d. POLIMORFISME

Pada suatu waktu, kita ingin membuat beberapa class yang hampir mirip dan memiliki fungsi/method yang sama kegunaanya namun memiliki definisi yang berbeda disetiap kelasnya. Jika membuat class satu persatu dan memdefinsikan fungsi/methodnya satu persatu maka akan membuat kode program terlihat tidak terstuktur.

Perbedaan antara konsep polimorfisme dan konsep abstraksi adalah polimorfisme bersifat opsional untuk didefinisikan kembali, sedangkan konsep abtraksi bersifat kontrak yang berarti semua fungsi yang yang merupakan *abstrac method* harus didefiniskan ulang di kelas turunan.

Dalam PBO,fungsi/ method yang terdapat pada kelas induk terdapat implementasi. Implementasi fungsi tersebut baru dilakukan dikelas turunannya. Polimorfisme dapat dilakukan dengan dua cara yaitu overloading, overriding.

Contoh penerapan polimorfisme :

```python
class Hewan:
    def __init__(self,hewan) -> None:
        self.hewan = hewan

    def bersuara(self) :
        pass

class Kucing(Hewan) :
    def __init__(self) -> None:
        super().__init__("Kucing")

    def bersuara(self):
        return "meow-meow"

class Sapi(Hewan):
    def __init__(self) -> None:
        super().__init__("Sapi")

    def bersuara(self):
        return "moooo"


# Main Program
hewan1 = Kucing()
hewan2 = Sapi()

print(f"Hewan 1 bersuara : {hewan1.bersuara()}")
print(f"Hewan 2 bersuara : {hewan2.bersuara()}")
```

Dari contoh diatas dapat kita lihat bahwa kelas *Kucing* dan *Sapi* merupakan kelas turunan dari kelas *Hewan sehingga* kelas *Kucing* dan *Sapi* juga memiliki fungsi yang dimiliki kelas *Hewan* yaitu fungsi *bersuara()*. Fungsi *bersuara()* didefinisikan ulang disetiap kelas turunannya.