MODUL PRAKTIKUM
PEMROGRMAAN BERORIENTASI OBJEK
MENGGUNAKAN PYTHON 3

Disusun oleh:

Muhamad Soleh, S.Si, M.Kom

Dino Hariatma Putra, S.T, M.Kom

INSTITUT TEKNOLOGI INDONESIA

TANGERANG SELATAN

JANUARI 2021

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Lab Komputer dan Lab Informatika IF ITI

Tata Tertib Praktikum Komputer

Penilaian Praktikum ini dilakukan terhadap 3 hal yaitu:

1. Penilaian hasil dari praktikum
2. Kehadiran
3. Sikap dan tingkah laku saat praktikum

Dalam praktikum ini peserta harus memperhatikan hal-hal sebagai berikut:

1. Kehadiran peserta praktikum harus 100%, semua modul praktikum harus dilaksanakan. Apabila peserta tidak hadir pada jadwal yang ditentukan, peserta harus dilaksanakan pada jam yang lain dengan konsekuensi membayar biaya perawatan sebesar Rp. 20.000,- (dua puluh ribu rupiah) per modul. Peserta yang tidak melaksanakan salah satu modul dan tidak mengganti pada jam yang lain, maka nilai praktikum modul tersebut adalah 0.
2. Bagi yang memerlukan praktikum tambahan, dapat menggunakan fasilitas laboratorium (bila ada yang kosong) dengan membayar biaya perawatan sebesar Rp. 1.500,- (seribu lima ratus rupiah) per jam atau bagian dari 1 jam.
3. Keterlambatan lebih dari 15 menit tidak diperkenankan mengikuti praktikum.
4. Setiap praktikum, Kartu Praktikum harus dibawa dan diserahakn pada asisten yang bertugas.
5. Duduklah pada tempat yang telah ditentukan sesuai dengan nomor yang tertera di Kartu Praktikum.
6. Tas, buku, flashdisk, dll; harus diletakkan pada tempat yang telah disediakan kecuali Buku Petunjuk Praktikum.
7. Peserta harus berpakaian rapih dan tidak diperkenankan memakai sandal.
8. Ruangan praktikum merupakan ruangan ber-AC. Di mana tidak seorang pun diperkenankan untuk merokok, membawa makanan, dan minuman.

9. Peralatan komputer yang ada adalah peralatan yang berharga. Kecerobohan peserta yang menyebabkan kerusakan alat harus ditanggung oleh peserta itu sendiri.
10. Praktikan dilarang mengganti atau mengubah perangkat lunak (software) atau kata sandi yang sudah ada.
11. Selama praktikum berlangsung, jaga kesopanan dan ketenangan supaya tidak mengurangi manfaat dari praktikum anda. Sangsi atas pelanggaran ini dapat mempengaruhi nilai anda.
12. Setelah praktikum selesai, harap bersihkan meja praktikum dari sampah-sampah dan membuang sampah tersebut pada tempat yang telah disediakan.
13. Sebelum meninggalkan ruangan, komputer dan monitor yang digunakan harus dalam keadaan mati.
14. Peserta harus meninggalkan ruangan bila ada aba-aba untuk selesainya waktu praktikum.
15. Nilai praktikum merupakan komponen penentu nilai akhir.

Tata Tertib Penggunaan Petunjuk Praktikum

Buku Petunjuk Praktikum adalah milik Institut, tidak diberikan tetapi dipinjamkan selama satu semester. Apabila rusak/hilang maka peserta dikenakan denda sebesar Rp. 10.000,- (sepuluh ribu rupiah) atau mengganti buku tersebut.

1. Pendahuluan Pemorgraman Berorientasi Objek

Pemrograman berorientasi objek atau dalam bahasa inggris disebut *Object Oriented Programming* (OOP) adalah paradigma atau teknik pemrograman di mana semua hal dalam program dimodelkan seperti objek dalam dunia nyata.
Objek di dunia nyata memiliki dua komponen utama berupa:

1. Ciri (*atribut / datafield / variable / data member*)
2. Aksi atau kelakuan (*fungsi / method / behavior*).

1.1 Membuat Template sebuah Objek dengan Kelas

Kelas merupakan sebuah template / Cetakan / blueprint / prototipe dari sebuah objek yang akan dibuat. Objek dapat berupa benda hidup atau benda mati. Contoh kita akan membuat objek uang. Maka kita perlu membuat blueprint / mesin cetak uangnya terlebih dahulu. Begitu juga dengan objek-objek lainnya yang akan dibuat.
Dalam Bahasa pemrograman python, syntax yang digunakan untuk membuat sebuah template / blueprint dengan konsep kelas secara sederhana adalah sebagai berikut:

```
class NamaKelas:
    pass
```

Misal kita akan membuat objek uang, maka kita perlu membuat blueprint nya terlebih dahulu. Berikut potongan kode untuk membuat blueprint nya:

```
1 class MesinCetakUang:
2     pass
```

Perhatikan bahwa nama kelas tidak boleh dipisahkan oleh Spasi. Kita bisa menggunakan metode huruf capital pada setiap awal kata dari nama kelas tersebut.

1.2 Membuat sebuah Objek dari Template

Class digunakan untuk membuat sebuah blueprint, untuk membuat sebuah objek dari template yang sudah dibuat yaitu menggunakan syntax sebagai berikut:

```
namaObjek = namaKelas()
```

Berdasarkan syntax tersebut, maka kita bisa membuat sebuah objek sesuai dengan banyaknya objek yang kita butuhkan sesaui dengan potongan kode berikut:

```
1  # Membuat template kelas MesinCetakUang
2  class MesinCetakUang:
3      pass
4
5  # Membuat objek pertama dari kelas MesinCetakUang
6  uang1 = MesinCetakUang()
7
8  # Mencetak objek uang1
9  print(uang1)
10
```

Perhatikan bahwa nama objek tidak boleh dipisahkan oleh spasi dan potongan kode tersebut akan menghasilkan output sebagai berikut:

```
<__main__.MesinCetakUang object at 0x032191C0>
```

Artinya kita sudah berhasil membuat objek uang1 yang disimpan di memory 0x032191C0 dari blueprint MesinCetakUang

1.3 Membuat Atribut pada objek

Pada Bahasa pemrograman python kita bisa membuat atau menambahkan atribut dari objek diluar kelas seperti pada syntax pada bari ke 12 berikut:

```
1  # Membuat template kelas MesinCetakUang
2  class MesinCetakUang:
3      pass
4
5  # Membuat objek pertama dari kelas MesinCetakUang
6  uang1 = MesinCetakUang()
7
8  #mencetak objek uang1
9  print(uang1)
10
11 # Menambahkan atribut nilai keobjek uang
12 uang1.harga = 5000
13
14 # Mencetak atribut harga pada objek uang
15 print(uang1.harga)
```

Kita bisa menambahkan dan mengakses isi atau nilai dari atribut yang dimiliki oleh sebuah objek dengan menggunakan operator dot atau titik. Keluaran dari potongan kode baris ke 15 adalah 5000. Karena atribut harga pada objek uang bernilai 5000.

1.4 Membuat Atribut pada objek menggunakan metode konstruktor

Metode / fungsi konstruktor digunakan untuk menambahkan atribut pada objek. Jadi, Selain menggunakan operator dot / titik, kita juga bisa menambahkan atribut dengan fungsi konstruktor.

Kegunaan lain dari Fungsi konstruktor adalah sebagai fungsi khusus yang digunakan Python untuk menginisialisasi pembuatan objek dari kelas tersebut. Jadi ketika objek dibuat, maka fungsi kontraktor adalah fungsi yang pertama kali dijalankan.
Fungsi konstruktor ditulis dengan menggunakan syntax:

def __init__(self, inputArgumen):

```
1  # Membuat template kelas MesinCetakUang
2  class MesinCetakUang:
3      # Fungsi Konstruktor
4      def __init__(self, hargaUang, logoPahlawan):
5          self.harga = hargaUang
6          self.logo = logoPahlawan
7
8  # Membuat objek pertama dari kelas MesinCetakUang
9  uang1 = MesinCetakUang(1000,"Pattimura")
10
11 # mencetak objek uang1
12 print(uang1)
13
14 # mencetak nila dari atribut harga
15 print(uang1.harga)
```

Perhatikan pada baris ke-3. Fungsi ***def __init__(self, hargaUang, logoPahlawan):*** merupakan fungsi konstruktor yang terdiri dari input argument hargaUang dan logoPahlawan.
Argumen *self* merupakan argument default yang disediakan oleh python untuk merepresentasikan objek yang terdapat pada fungsi kontruktor tersebut. Pada fungsi konstruktor tersebut terdapat dua buah atribut, yaitu ***harga*** dan ***logo***. Nilai atribut tersebut diisi melalui input argument ***hargaUang*** dan ***logoPahlawan.***

Perhatikan bahwa jika sebelumnya kita menambahkan atribut dengan cara ***namaObjek.namaAtribut = nilaiAtribut.*** Pada fungsi kontruktor, namaObjek direpresentasikan oleh self.

Output dari baris ke 12 menunjukan objek dari uang1, sedangkan baris ke-15 output nya adalah 1000. Karena kita sudah mengisi nilai dari atribut harga melalui argument hargaUang dengan nilai 1000. Jika kita ingin menampilkan nilai dari atribut logo pada objek uang1, maka kita hanya perlu menambahkan baris berikutnya dengan perintah yang sama seperti baris ke 15 dengan mengganti nama atributnya.

Gambar 1. Gambar uang Rp. 1000

Yey!! Kita sudah berhasil membuat objek uang seharga 1000 dengan logo Pattimura.

1.5 Membuat Fungsi Pada Objek

Pada dasarnya, kita telah membuat fungsi di dalam template ***class MesinCetakUang***. Fungsi tersebut dinamakan sebagai fungsi Konstruktor, yang salah satu kegunaannya digunakan untuk menambahkan atribut kepada objek yang akan dibuat.

Kita akan membuat fungsi kedua, yaitu fungsi belanja. Sebagai mana kita ketahui uang dapat digunakan untuk membeli sesuatu atau membeli objek lainnya. Di sini kita tidak akan membuat objek yang akan dibeli, karena tujuan kita hanya ingin membuat fungsi saja.

```python
class MesinCetakUang:
    def __init__(self, hargaUang, logoPahlawan):
        self.harga = hargaUang
        self.logo = logoPahlawan

    # Membuat Fungsi belanja
    def belanja(self):
        print("Anda telah membeli barang seharga RP.1000")
        self.harga -= 1000

uang1 = MesinCetakUang(1000,"Pattimura")
print(uang1)

# mencetak nila dari atribut harga sebelum belanja
print(uang1.harga)

# memanggil fungsi belanja
uang1.belanja()

# mencetak nila dari atribut harga setelah belanja
print(uang1.harga)
```

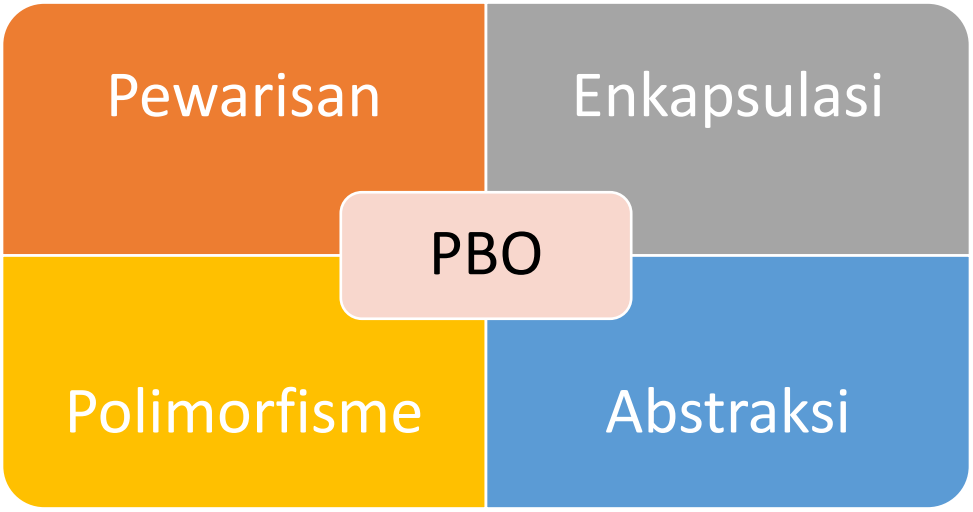
Perhatikan kode baris ke 7 sampai baris ke-9. Kita telah membuat fungsi belanja, dimana fungsi tersebut akan mencetak perintah yang ada di block kode tersebut. Ingat pada mata kuliah konsep pemrograman kalian sudah belajar fungsi pada python. Perbedaan nya pada konsep PBO kalian perlu menambahkan argument self pada fungsi tersebut. Ingat kembali fungsi ada dua jenis, yaitu fungsi void dan fungsi return. Kedua fungsi tersebut dapat digunakan dalam konsep PBO.

Keluaran pada baris ke 15 adalah 1000. Ketika fungsi belanja dipanggil pada baris ke 18, maka akan menjalankan perintan pada baris ke 8 dan 9 dari kode. Sehingga pada baris ke 21 keluarannya menjadi 0. Karena uang 1000 kita sudah digunakan untuk belanja. Yah…! Uang nya habis deh buat belanja.

1.6 Tugas 1. Pendahuluan PBO

1. **Buat objek baru pada potongan kode sebelumnya dengan nilai 5000**
2. **Tambahkan Fungsi return sebagai fungsi baru pada potongan kode sebelumnya. Nama dan isi fungsi bebas.**
3. **Buat sebuah program yang menggunakan konsep PBO tentang Objek Mahasiswa. Atribut dan fungsi Mahasiswa sesuai dengan identitas dan kebutuhan masing-masing. Tambahkan fungsi input pada program kalian untuk mengambil nilai dari keyboard.**

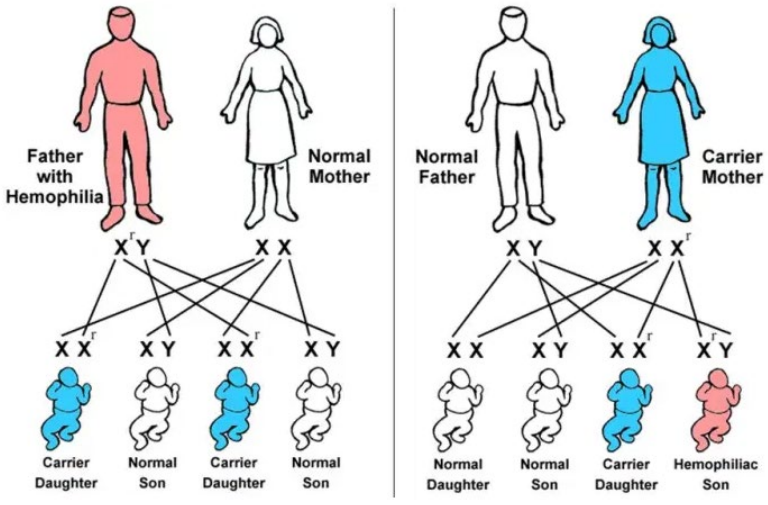
2. Pewarisan



Gambar 2. Empat Dasar sub konsep dari konsep Pemrograman Berorientasi Objek

Pada konsep pemrograman berorientasi objek (PBO) terdapat 4 dasar sub konsep yang harus dikuasi, diantaranya adalah pewarisan, enkapsulasi, polimorfisme, dan abstraksi.

Konsep pewarisan atau dalam Bahasa inggris disebut inheritance adalah sebuah konsep yang mengadopsi dari konsep pewarisan sifat dalam teori reproduksi manusia. Menurut teori reproduksi, seorang anak yang terlahir ke muka bumi, akan mewarisi sifat dari kedua orang tua nya, baik yang berupa sifat fisik maupun psikis, melalui susunan atau kombinasi gen yang ada di dalam tubuh manusia.



Gambar 3. Konsep Pewarisan pada manusia

Dalam konsep pemrograman berorientasi objek, pewarisan mencakup semua lingkup / bidang, jadi tidak hanya pewarisan pada manusia saja, tetapi bisa juga terjadi pada makhluk hidup lainnya atau pun pada makhluk yang tak bernyawa. Hal ini juga selaras dengan pribahasa ***"buah yang jatuh tak jauh dari pohon nya"***.

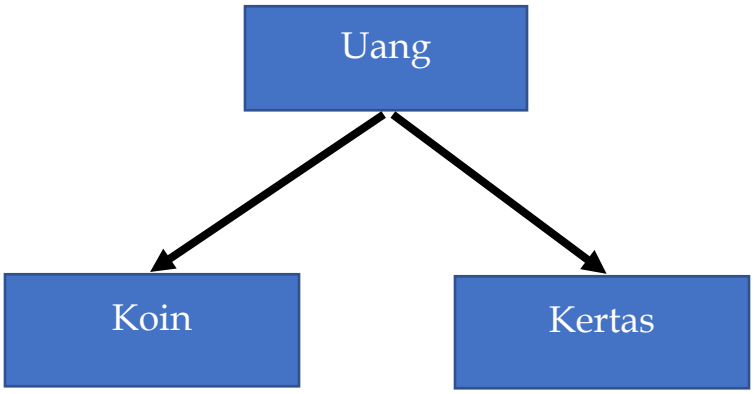
Inheritance / pewarisan merupakan salah satu karakteristik dalam pemrograman berorientasi objek, dimana sebuah class mewarisi / inherit sifat-sifat (dalam hal ini atribut & fungsi) dari class lain yang merupakan parent dari class tadi. Class yang menurunkan sifat-sifatnya disebut superclass, sedangkan class yang mewarisi sifat dari superclass disebut subclass.

2.1 Superclass dan subclass

Supperclass merupakan parent class sedangkan sub class merupakan child class. Syntaks pada python dapat dibuat sebagai berikut:

```
1  class Superclass:
2      pass
3
4  class SubClass(Superclass):
5      pass
```

Misalkan kita akan menggunakan konsep pewarisan pada objek uang, maka salah satu contoh kasus pewarisan seperti pada gambar 4.



Gambar 4. Contoh kasus superclass dan subclass

Pada gambar4, class uang merupakan superclass / parent class sedangkan class koin dan class kertas merupakan sub class / child class. Maka, implementasi dari konsep pewarisan tersebut adalah sebagai berikut:

```python
#konsep pewarisan
class Uang:
    def __init__ (self, nominal):
        self.nominal = nominal

    def belanja (self, totalBelanja):
        print("Terima kasih. Sisa uang anda adalah: ")
        print(self.nominal - totalBelanja)

class Koin(Uang):
    def __init__ (self, bahan):
        self.bahan = bahan

    def beli (self, totalBeli):
        print("Terima kasih. Sisa uang anda adalah: ")
        print(self.nominal - totalBeli)

uang1 = Uang(10000)
uang1.belanja(5000)

koin1 = Koin("logam")
koin1.nominal = 1000
koin1.beli(500)
koin1.belanja(500)
```

Perhatikan sintaks penulisan superclass dan subclass pada baris ke 2 dan baris ke 10. Pada superclass Uang terdapat ***atribut nominal*** dan ***fungsi belanja.*** Sedangkan pada sub class Koin memiliki ***atribut bahan*** dan ***fungsi beli***.

Perhatikan lagi baris kode ke 22 dan 24. Objek koin1 mengakses ***atribut nominal*** dan ***fungsi belanja.*** Padahal subclass koin tidak memiliki atribut dan fungsi tersebut. Namun, karena koin merupakan subclass dari superclass Uang, maka **semua atribut dan fungsi yang terdapat pada class Uang akan diwariskan ke class Koin**. Inilah yang dinamakan konsep pewarisan / inheritance. ***Mudah bukan?***

Jika kode tersebut dijalankan, maka tidak akan terjadi error, dan output yang dihasilkan adalah:

```
Terima kasih. Sisa uang anda adalah:
5000
Terima kasih. Sisa uang anda adalah:
500
Terima kasih. Sisa uang anda adalah:
500
```

Keterangan output tersebut adalah:

Uang 10.000 telah dibelanjakan sehingga sisa uang nya 5.000 (baris kode 18 - 19)

Koin 1.000 telah dipakai untuk membeli sehingga sisa uang nya 500 (baris kode 23)

Koin 1.000 telah dibelanjakan sehingga sisa uang nya 500 (baris kode 24)

Problem Solving – Computational Thinking

1. Perhatikan, sepertinya ada ***semantic error***. Masih ingatkan, semantic error itu apa? Seharusnya koin 1.000 sisa uang nya adalah 0, karena kita sudah gunakan untuk belanja Rp. 500 dan beli Rp. 500, tapi kenapa output baris kode ke 24 masih Rp. 500? (Perbaiki kode diatas agar tidak terjadi semantic error)
2. Perhatiakan kembali gambar 4. Seharusnya terdapat 3 buah class, yaitu Uang, Koin, dan Kertas. Implementasikan class Kertas, agar kode sesuai dengan gambar 4.

2.2 Fungsi Super

Dalam Bahasa pemrograman python, terdapat fungsi super() yang dapat di implementasikan pada konsep pewarisan. Fungsi super () digunakan untuk memberikan akses ke atribut dan metode kelas induk atau saudara. Fungsi super () mengembalikan / mereturn objek yang mewakili kelas induk.

```
14      def beli (self, totalBeli):
15          print("Terima kasih. Sisa uang anda adalah: ")
16          print(self.nominal - totalBeli)
17          super().belanja(totalBeli)
18  uang1 = Uang(10000)
19  uang1.belanja(5000)
20
21  koin1 = Koin("logam")
22  koin1.nominal = 1000
23  koin1.beli(500)
```

Silahkan tambahkan fusngsi super pada baris ke 17, lalu hapus kode baris ke 24, sehingga kode menjadi seperti diatas. Ketika dijalankan, outpur dari kode tersebut akan sama dengan sebelumnya. Fungsi super merepresentasikan objek yang mewakili class Uang yang memberikan akses ke metode belanja yang ada pada class Uang itu sendiri, namun dipanggil dari subclass nya. **Jadi intinya dengan fungsi super kita bisa mengakses atribut dan fungsi yang ada di dalam superclass. Termasuk juga fungsi kontruktor ya….**

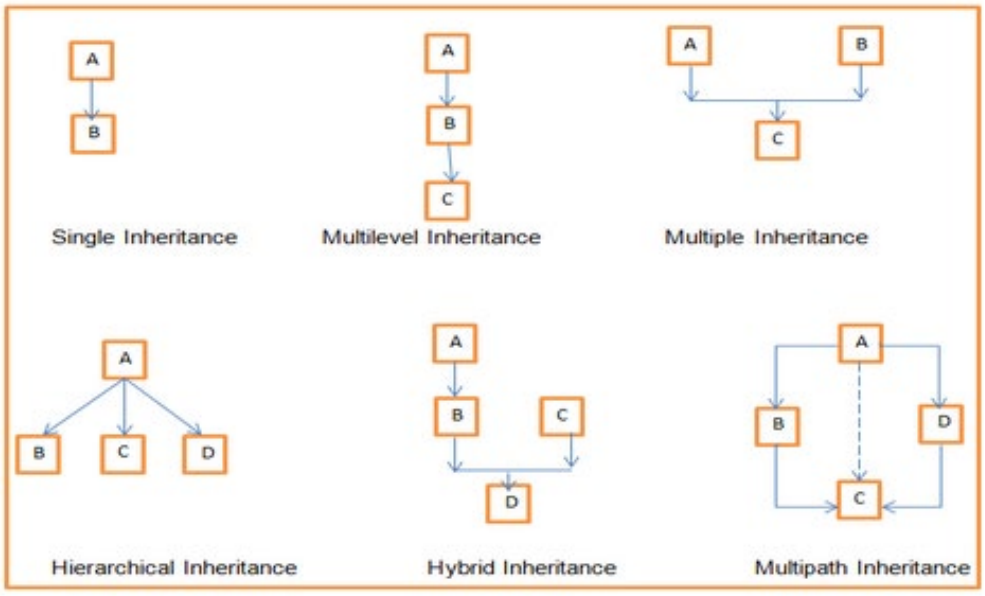
Problem Solving – Computational Thinking

1. Implementasikan fungsi super() untuk mengembalikan atribut dari superclass.
2. Implementasikan fungsi super() untuk mengembalikan fungsi konstruktor dari superclass.

2.3 Jenis-Jenis Inheritance

Dalam konsep pewarisan atau inheritance, terdapat beberapa jenis inheritance yang perlu kita ketahui, diantaranya adalah single inheritance, multilevel inheritance, hierarchical inheritance, multiple inheritance, hybrid inheritance, dan multipath inheritance. Visualisasi jenis-jenis inheritance dapat dilihat pada gambar 5.

single inheritance, multilevel inheritance dan hierarchical inheritance hanya memiliki 1 superclass dari masing-masing sub class nya. Maka implementasi nya akan sama seperti class uang dan koin (single hierarchical). Setelah ditambahkan class kertas, maka tipe inheritance menjadi hierarchical inheritance. Multilevel inheritance dapat diimplentasikan dengan kasus class orangtua, anak, dan cucu.



Gambar 5 Visualisasi Tipe-tipe inheritance

Multiple inheritance

Studi kasus dari multiple inheritance adalah class Ayah, Ibu, dan Anak. Perhatikan kode dibawah ini.

```
1  #Multiple inheritance
2  class Ayah:
3      def __init__(self,nama):
4          self.nama = nama
5
6      def tampilkanNama(self):
7          print("Nama saya adalah" , self.nama)
8
9  class Ibu:
10     def __init__(self,usia):
11         self.usia = usia
12
13     def tampilkanUsia(self):
14         print("Usia saya adalah" , self.usia)
15
16 class Anak(Ayah,Ibu):
17     def __init__(self):
18         print("Hallo Dunia")
19
20 muhamadSoleh = Anak()
21 muhamadSoleh.nama = "Muhamad Soleh"
22 muhamadSoleh.tampilkanNama()
23 muhamadSoleh.usia = 17
24 muhamadSoleh.tampilkanUsia()
```

Perhatikan baris kode ke 16

Ketika program tersebut dijalankan, maka output yang muncul adalah sebagai berikut:

```
Hallo Dunia
Nama saya adalah Muhamad Soleh
Usia saya adalah 17
```

Objek muhamadSoleh dibangkitkan dari kelas Anak. Dimana class Anak memiliki 2 parent, yaitu class Ayah dan class Ibu. Sehingga class Anak mewarisi atribut maupun fungsi dari class Ayah dan class Ibu.

Hybrid inheritance dan multipath inheritance

Hybrid inheritance merupakan gabungan dari berbagai tipe inheritance. Pada gambar 5, hybrid inheritance merupakan gabungan dari single dan multiple inheritance. Sedangkan multipath inheritance dapat direpresentasikan seperti kalimat berikut:
"Nenek ku yang merupakan Ibu dari Ibu ku ternyata adalah Ibu ku juga"

Problem Solving – Computational Thinking

1. Implementasikan kasus hybrid inheritance
2. Implementasikan kasus multipath inheritance

2.4 Tugas 2. Pewarisan

Buat sebuah progam yang menggunakan konsep pewarisan sesuai dengan tema dan nomor terakhir NRP sebagai berikut:

NRP 0 – 3 : Rumah Sakit

NRP 4 – 6 : Artis

NRP 7 – 9 : Youtube

Ketentuan:

- Gunakan fungsi input untuk mengisi nilai atribut pada objek
- Implementasikan semua konsep pada masing-masing sub bab pewarisan
- Gunakan fungsi return dan fungsi void pada program

3. Enkapsulasi

Konsep enkapsulasi berfungsi untuk menyembunyikan detail yang tidak perlu sehingga memudahkan dalam mengatur struktuir program. Disamping itu, konsep enkapsulasi juga bertujuan untuk melindungi data yang penting agar tidak bisa diakses sembarangan. Encapsulasi berasal dari konsep kapsul. Dimana kapsul melindungi obat yang terdapat di dalam cangkang yang mudah larut. Implementasi konsep enkapsulasi terdiri dari 2 ciri utama, yaitu menggunakan ***akses modifier private*** dan menggunakan ***setter dan getter***.

Gambar 6. Kapsul

3.1 Akses Modifier

Access Modifier adalah sebuah "hak akses" yang diberikan kepada sebuah variabel/method dengan tujuan untuk menjaga integritas dari data tersebut ketika ingin diakses object lain. Hak akses sendiri diberikan oleh pembuat program, dengan adanya Access Modifier, kita dapat membatasi resource-resource mana saja yang dapat diakses oleh object tertentu, turunannya, ataupun oleh method tertentu.

Dalam Bahasa pemrograman python, terdapat 3 akses modifier, diantaranya adalah: public, private, dan protected. Perbedaan dari ketiga hak akses tersebut dapat dilihat pada table 1.

Table 1. Perbedaan Hak Akses pada Python

Akses Modifier	Sama Class	Sub Class	Sama Package	Beda Package
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Private	Yes	No	No	No

Default akses modifier dari python adalah public. Pada kasus class Uang dan Koin pada pewarisan, semua atribut dan fungsi memiliki akses modifier public. Artinya,

atribut maupun fungsi tersebut dapat diakses dari kelas yang sama, dari kelas turunanya / subclass, dari package yang sama, dan juga dari package yang berbeda. Akses modifier public intinya dapat diakses dimanapun.

Akses modifeier protected

Akses modifier protected dalam python ditulis menggunakan single underscore. Perhatikan kode berikut ini:

```
1  #konsep enkapsulasi
2  class Uang:
3      def __init__(self,nominal):
4          self._nominal = nominal #protected attribute
5
6      def _belanja (self,totalBelanja): #protected method
7          print("Terima kasih. Sisa uang anda adalah: ")
8          print(self._nominal - totalBelanja)
9
10 class Koin(Uang):
11     def __init__ (self, bahan):
12         self.bahan = bahan #public attribute
13
14     def beli(self, totalBeli):
15         print("Terima kasih. Sisa uang anda adalah: ")
16         print(self._nominal - totalBeli)
17
18 uang1 = Uang(10000)
19 uang1._belanja(5000)
20 koin1 = Koin("logam")
21 koin1._nominal = 1000
22 koin1.beli(500)
23
```

Perhatikan pada baris kode ke 4 (protected attribute) dan 8 (protected method) penulisan atribut dan fungsi menggunakan single underscore. Artinya atribut dan fungsi tersebut memiliki akses modifier protected. Pada dasarnya akses modifier

protected dan public perbedaannya hanya pada package. Ketika diakses dari package yang berbeda atribut protected tidak bisa diakses. Jadi output pada kode diatas sama seperti output ketika kita menggunakan akses modifier public. Yaitu sebagai berikut:

```
Terima kasih. Sisa uang anda adalah:
5000
Terima kasih. Sisa uang anda adalah:
500
```

Problem Solving – Computational Thinking

Bagaimana mengimplementasikan akses modifier protected pada kasus beda package sehingga terlihat perbedaannya dengan akses modifier public.

Pada dasarnya programmer menggunakan akses modifier protected hanya untuk memberikan tanda bahwa atribut / fungsi tersebut tidak akan dia ubah. Sedangkan akses modifier private dibuat agar orang lain / hacker tidak bisa mengetahui atribut / fungsi yang ada di dalam class yang dibuat. Namun ternyata python memiliki sebuah metode untuk mengakses atribut dan fungsi walaupun keduanya memiliki akses modifier private. Menarik bukan?

Akses modifier private

Akses modifier private dalam python ditulis menggunakan double underscore.

Perhatikan kode berikut ini:

```python
class Uang:
    def __init__(self, nominal):
        self._nominal = nominal    # protected variable
        self.__logo = "Pattimura"  # private variable

uang1 = Uang(1000)

print(uang1._Uang__logo)

```

Perhatikan paba baris kode ke 4. Atribut logo merupakan atribut dengan akses modifier private, seharisnya atribut tersebut tidak bisa diakses di luar class Uang, karena kita sudah melindungi atribut tersebut dengan ***__private*** namun ketika baris 8 kode dieksekusi oleh python, maka output yang akan tampil adalah Pattimura. Dalam python kasus seperti ini disebut sebagai **Private name mangling.** Untuk membuktikan bahwa atribut private tidak bisa diakses diluar class, perhatikan kode berikut:

```
1  #konsep enkapsulasi
2  class Uang:
3      def __init__(self,nominal):
4          self.__nominal = nominal #private attribute
5
6      def __belanja (self,totalBelanja): #private method
7          print("Terima kasih. Sisa uang anda adalah: ")
8          print(self.__nominal - totalBelanja)
9
10 class Koin(Uang):
11     def __init__ (self, bahan):
12         self.bahan = bahan #public attribute
13
14     def beli(self, totalBeli):
15         print("Terima kasih. Sisa uang anda adalah: ")
16         print(self.__nominal - totalBeli)
17
18 uang1 = Uang(10000)
19 uang1.__belanja(5000)
```

Perhatikan pada baris kode ke 4 (private attribute) dan 8 (private method) penulisan atribut dan fungsi menggunakan double underscore. Artinya atribut dan fungsi tersebut memiliki akses modifier private. Artinya atribut dan fungsi tersebut hanya bisa diakses pada class Uang saja. Ketika kita mencoba akses dari luar kelas akan error. Perhatikan baris kode 19 (memanggil fungsi private belanja di luar class Uang) maka output dari kode berikut adalah:

```
Traceback (most recent call last):
  File "c:/0. DATA SOLEH/0. KULIAH/KP/PY/oop2.py", line 19, in <module>
    uang1.__belanja(5000)
AttributeError: 'Uang' object has no attribute '__belanja'
```

3.2 Setter dan Getter

Setter dan getter merupakan sebuah method dengan akses modifier public yang bertujuan untuk melakukan interaksi pada variable dan fungsi yang memiliki hak akses private. Selain menggunakan **Private name mangling,** kita juga bisa berinteraksi dengan variable dan method yang meimiliki hak akses private dengan menggunakan setter dan getter. Perhatikan kode berikut untuk penggunaan setter dan getter.

```
1  class Uang:
2      def __init__(self, nominal):
3          self.__nominal = nominal    # private variable
4
5      def setNominal(self,nominalBaru): #setter
6          self.__nominal = nominalBaru
7
8      def getNominal(self): #getter
9          return self.__nominal
10
11 uang1 = Uang(1000)
12 #print(uang1.__nominal) (error)
13 print(uang1.getNominal()) #output = 1000
14
15 uang1.__nominal = 5000
16 print(uang1.getNominal()) #output = 1000
17
18 uang1.setNominal(5000)
19 print(uang1.getNominal()) #output = 5000
20
```

Perhatikan bahwa walaupun atribut __nominal bersifat private, tapi kita masih bisa berinteraksi dengan atribut tersebut melalui setter dan getter. Fungsi dari setter adalah untuk mengganti atau mengisi nilai dari variable, sedangkan fungsi dari getter adalah untuk mengambil nilai yang dimiliki oleh variable. Dengan kita menggunakan akses modifier private serta mengimplementasikan setter dan getter, maka kita sudah mengimplementasikan konsep enkapsulasi dalam pemrograman kita.

3.3 Staticmethod dan Classmethod

Pada Bahasa pemrograman python, atribut maupun fungsi dapat dimiliki oleh objek ataupun kelas. Perhatikan kode berikut ini:

```
1  #staticmethod dan classmethod
2  class Uang:
3
4      jumlah_objek = 0 #atribut class
5
6      def __init__(self, nominal):
7          self.nominal = nominal    #atribut objek
8          Uang.jumlah_objek += 1
9          print(Uang.jumlah_objek)
10
11 uang1 = Uang(1000)
12 uang2 = Uang(2000)
```

Pada baris 4 atribut jumlah_objek adalah milik class Uang, sedangkan pada baris 7, atribut nominal milik objek uang1, maka dari itu cara mengakses atribut yang dimiliki oleh classs dipanggil dari nama class nya bukan nama objek nya seperti baris 8. Output dari kode diatas adalah:

```
DATA SOLEH/0. KULIAH/KP/PY/oop3.py"
1
2
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Setiap kali objek dibuat, maka console akan menampilkan jumlah objek yang telah dibuat. Menarik bukan?

Jika atribut boleh dimiliki oleh class, maka apakah fungsi juga boleh dimiliki oleh class?, bagaimana implementasi nya dalam Bahasa pemrograman python ?.

Selama ini kita sudah mempelajari atribut dan fungsi milik objek. serta atribut milik class. Sekarang saatnya kita pelajari fungsi milik class atau yang biasa disebut sebagai classmethod.

Class Method adalah jenis method yang diawali dengan **decorator @classmethod** dan wajib menyertakan **cls** pada parameternya, dimana parameter cls ini merujuk pada kelas itu sendiri. Class method sangat cocok digunakan jika terdapat case yang tidak memerlukan komunikasi langsung dengan instance suatu class, tetapi masih membutuhkan attribut-atrribut lain dalam classnya. Perhatikan kode berikut ini:

```
1  #staticmethod dan classmethod
2  class Uang:
3
4      jumlah_objek = 0 #atribut class
5
6      def __init__(self, nominal):
7          self.nominal = nominal    #atribut objek
8          Uang.jumlah_objek +=1
9
10     @classmethod #ini adalah decorator
11     def tampilkanJumlah_objek(cls):
12         print("Jumlah objek :" , cls.jumlah_objek)
13
14 uang1 = Uang(1000)
15 uang2 = Uang(2000)
16 Uang.tampilkanJumlah_objek()
```

Perhatikan dari mulai baris kode 10. Fungsi tampilkanJumlah_objek merupakan fungsi yang dimiliki oleh class bukan objek, maka cara memanggilnya juga dari nama class bukan nama objek nya. Seperti pada kode baris ke 16.

Lalu apa itu Staticmethod?

StaticMethod adalah jenis method yang diawali dengan **decorator @staticmethod**, yang bersifat berdikari (berdiri di kaki sendiri) artinya fungsi ini tidak dimiliki oleh siapapun. Sehingga bisa diakses baik dari objek maupun dari class. Fungsi ini juga tidak butuh instance maupun attribut-attribut lain dalam class-nya dan dia juga tidak diwajibkan menyertakan parameter khusus. Seperti yang kita tahu bahwa objectmethod harus

menggunakan parameter self, sedangkan classmethod harus mengunakan parameter cls. Jenis fungsi ini sangat cocok sebagai method yang berfungsi sebagai helper yang biasanya tidak butuh atribut-atibut apapun dari suatu class. Perhatikan kode berikut ini:

```
1  #staticmethod dan classmethod
2  class Uang:
3
4      jumlah_objek = 0 #atribut class
5
6      def __init__(self, nominal):
7          self.nominal = nominal    #atribut objek
8          Uang.jumlah_objek +=1
9
10     @classmethod #ini adalah decorator
11     def tampilkanJumlah_objek(cls):
12         print("Jumlah objek :" , cls.jumlah_objek)
13
14     @staticmethod
15     def tampilkanJumlahObjek():
16         print("Jumlah objek :" , Uang.jumlah_objek)
17
18 uang1 = Uang(1000)
19 uang2 = Uang(2000)
20 Uang.tampilkanJumlah_objek() #memanggil classmethod
21
22 Uang.tampilkanJumlahObjek() #memanggil staticmethod
23 uang2.tampilkanJumlahObjek() #memanggil staticmethod
```

Perhatikan baris kode ke 14. Fungsi tampilkanJumlahObjek() merupakan staticmethod, dimana fungsi tersebut dapat diakses baik dari class (baris kode 22) ataupun diakses dari objek (baris kode 23), karena sifat dari fungsi ini adalah berdikari (berdiri di kaki sendiri) artinya fungsi ini tidak dimiliki oleh siapapun.

Problem Solving – Computational Thinking

Apakah fungsi dari classMethod dapat diakses dari objek? begitu juga sebaliknya apakah fungsi dari objekMethod / instanceMethod dapat diakses dari class?

3.4 @properti dan format()

Fungsi Format() memungkinkan kita untuk menampilkan informasi dari atribut yang terdapat didalam objek dengan lebih mudah. Perhatikan kode berikut ini:

```
1  #format
2  class Uang:
3      def __init__(self, nominal, jenis, logo):
4          self.nominal = nominal
5          self.jenis = jenis
6          self.logo = logo
7          self.info = print("Uang dengan nominal :{}, jenis : {}, logo: {} \
8              \nBerhasil dibuat".format(self.nominal,self.jenis,self.logo))
9
10 uang1 = Uang(1000,"kertas","Pattimura")
11 uang1.info
```

Perhatikan baris kode 7 dan 8. Dengan menggunakan fungsi format, memungkinkan kita membuat placeholder dan mengisi nilai dari atribut objek untuk ditampilkan kepada user ketika objek dibuat. Output dari kode berikut adalah:

```
Uang dengan nominal :1000, jenis : kertas, logo: Pattimura
Berhasil dibuat
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Problem Solving – Computational Thinking

Tambahkan pada baris ke 12 : uang1.nominal = 2000

Tambahkan pada baris ke 13 : uang1.info

Apakah nominal akan berubah menjadi 2000 ? atau tetap 1000 ?

Agar format mengikut perubahan nilai dari atributnya, maka kita perlu menggunakan konsep **decorator @property.** Perhatikan kode berikut ini:

```python
#format
class Uang:
    def __init__(self, nominal, jenis, logo):
        self.nominal = nominal
        self.jenis = jenis
        self.logo = logo

    @property
    def info(self):
        print("Uang dengan nominal :{}, jenis : {}, logo: {} \
            \nBerhasil dibuat".format(self.nominal,self.jenis,self.logo))

uang1 = Uang(1000,"kertas","Pattimura")
uang1.info
uang1.nominal = 2000
uang1.info

```

Perhatikan baris kode 8 sampai akhir. Ketika nominal dierubah dari 1000 menjadi 2000, maka output program akan mengikuti perubahan nilai dari atribut objeknya. Berikut adalah output dari program tersebut:

```
Uang dengan nominal :1000, jenis : kertas, logo: Pattimura
Berhasil dibuat
Uang dengan nominal :2000, jenis : kertas, logo: Pattimura
Berhasil dibuat
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

3.5 Tugas Enkapsulasi

Buat sebuah progam yang menggunakan konsep enkapsulasi sesuai dengan tema dan nomor terakhir NRP sebagai berikut:

NRP 0 – 3 : Negara

NRP 4 – 6 : Rumah

NRP 7 – 9 : Bank

Ketentuan:

- Gunakan fungsi input untuk mengisi nilai atribut pada objek
- Implementasikan semua konsep pada masing-masing sub bab enkapsulasi
- Gunakan fungsi return dan fungsi void pada program

4. Polimorfisme

Dari segi bahasa, ***Polimorfisme*** (bahasa inggris: *Polymorphism*) berasal dari dua kata bahasa Latin yakni ***poly*** dan ***morph***. *Poly* berarti banyak, dan *morph* berarti bentuk. ***Polimorfisme*** **berarti banyak bentuk** (wikipedia). Sehingga dalam Bahasa pemrograman berorientasi objek, polimorphisme merupakan kode program yang memiliki satu entitas tapi berbagai bentuk. Jika bingung, kKita bisa analogikan dengan ilusi optic. Pada Gambar 7, terdapat hanya 1 gambar saja, tapi ternyata ada dua bentuk yaitu seorang ***Bapak Brewok menggunakan topi*** dan ***Anjing memegang tulang*** (jika gambar dibalik 180 drajat)

Gambar 7. Ilusi Optik (Satu Gambar Dua Bentuk: Anjing memegang tulang dan Bapak Brewok menggunakan topi)

Polimorfisme sangat erat kaitannya dengan konsep inheritance / pewarisan. Polimorfisme hanya akan terjadi jika dalam program terdapat konsep pewarisan. Polimorfisme terdiri dari 2 konsep, yaitu Overriding dan Overloading. Overriding biasa juga disebut sebagai True polmorfisme sedangkan Overlaoding disebut Trivia Polimorfisme.

4.1 Polimorfisme Overriding

Polimorfisme Overriding adalah Polimorfisme yang terjadi jika terdapat atribut atau fungsi dengan nama yang sama dalam superclass dan subclassnya. Perhatikan kode berikut ini:

```
1  #Polimorfisme Overriding
2  class Uang:
3      def __init__(self, nominal,logo):
4          self.nominal = nominal
5          self.logo = logo
6          print("Uang : {}, logo: {}.".format(self.nominal,self.logo))
7
8      def info(self):
9          print("Uang dengan nominal: " , self.nominal)
10
11 class Kertas(Uang):
12     def __init__(self,bahan):
13         #memanggil konstruktor ini dengan super()
14         super().__init__(1000,"Pattimura")
15         self.bahan = bahan
16
17     def info(self):
18         print("Uang dengan nominal: " , self.nominal)
19         print("Uang dengan bahan: " , self.bahan)
20
21 uang1 = Kertas("Kapas")
22 uang1.info()
```

Perhatikan pada baris 14. Kita kembali menggunakan konsep super() untuk mengambil fungsi konstruktor pada superclass. Superclass Uang dan subclass Kertas memiliki fungsi ***info().*** Ketika **dua class** yang menunjukan **konsep pewarisan** memiliki **atribut atau fungsi yang sama**, maka konsep itu disebut sebagai **polimorfisme overriding**. Perintah yang akan dijalankan oleh python untuk kasus diatas adalah fungsi ***info(self)*** yang terdapat dalam subclass nya.

Tujuan dari overriding diantaranya adalah Subclass yang berusaha memodifikasi tingkah laku yang diwarisi dari superclass. Sehingga subclass memiliki tingkah laku yang lebih spesifik. Method pada parent class disebut overriden method, sedangkan Method pada subclass disebut overriding method.

Aturan Overriding diantaranya adalah:

a. Mode akses overriding method harus sama atau lebih luas dari pada overriden method,
b. Subclass hanya boleh meng-override method superclass satu kali saja, tidak boleh ada lebih dari satu method pada kelas yang sama yang sama persis.

c. Overriding method tidak boleh throw checked exceptions yang tidak dideklarasikan oleh overridden method (ateri Exception Handling).

4.2 Polimorfisme Overloading

Polimorfisme Overloading terjadi ketika kita memiliki method dengan nama yang sama, namun jumlah argumen nya berbeda pada dua kelas class yang menerapkan konsep inheritance. Tujuan dari polimorfisme overloading adalah untuk memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang mirip. Perhatikan kode berikut ini:

```
#Polimorfisme Overriding
class Uang:
    def __init__(self, nominal,logo):
        self.nominal = nominal
        self.logo = logo
        print("Uang : {}, logo: {}.".format(self.nominal,self.logo))

    def info(self):
        print("Uang dengan nominal: " , self.nominal)

class Kertas(Uang):
    def __init__(self,bahan):
        #memanggil konstruktor ini dengan super()
        super().__init__(1000,"Pattimura")
        self.bahan = bahan

    def info(self,noSeri):
        print("Uang dengan nominal: " , self.nominal)
        print("Uang dengan bahan: " , self.bahan)
        print("Uang dengan no Seri: " , noSeri)

uang1 = Kertas("Kapas")
uang1.info(12345)
```

Superclass Uang dan subclass Kertas sama-sama memiliki fungsi ***info()*** namun argument nya berbeda**.** noSeri menjadi argumen pada class Kertas, sedangkan pada class Uang tidak memilki argumen noSeri. Ketika **dua class** yang menunjukan **konsep pewarisan** memiliki **atribut atau fungsi yang sama**, namun **argumennya berbeda**, maka konsep itu disebut sebagai **polimorfisme overloading**. Perintah yang akan

dijalankan oleh python untuk kasus diatas adalah fungsi ***info(self,noSeri)*** yang terdapat dalam subclass nya.

Ingat! Dalam Bahasa pemorgraman python, kita tidak diperbolehkan menulis nama fungsi yang sama pada satu class.

4.3 Method Resolution Order

Method Resolution Order adalah sebuah property yang disediakan oleh python untuk menampilkan urutan pemanggilan fungsi pada kasus multiple inheritance. Perhatikan kode berikut ini:

```
1  #Method Resolution Order
2  class Ayah:
3      def __init__ (self, nama):
4          self.nama = nama
5
6      def tampilkanNama(self):
7          print("Nama Saya: ",self.nama)
8
9  class Ibu:
10     def __init__ (self, nama):
11         self.nama = nama
12
13     def tampilkanNama(self):
14         print("Nama Saya: ",self.nama)
15
16 class Anak(Ayah,Ibu):
17     def __init__ (self, nama):
18         self.nama = nama
19
20     def tampilkanNama(self):
21         print("Nama Saya: ",self.nama)
22
23 anak = Anak("Muhamad Soleh")
24 anak.tampilkanNama()
25 #help(anak)
```

Pada kode diatas, terdapat kasus multiple inheritance dan polimofisme overriding pada fungsi ***tampilkanNama(self).*** Ketika dijalankan, maka output dari program akan menampikan hasil sebagai berikut:

```
DATA SOLEH/0. KULIAH/KP/PY/oop5.py"
Nama Saya:  Muhamad Soleh
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Fungsi ***tampilkanNama(self)*** yang dijalankan adalah fungsi yang terdapat pada kelas anak. Hal ini selaras dengan method resolution order yang disediakan oleh python. Untuk mengetahui method resolution order, silahkan nonaktifkan komentar pada baris kode ke 25, maka output nya akan tampil seperti berikut:

```
Help on Anak in module __main__ object:

class Anak(Ayah, Ibu)
 |  Anak(nama)
 |
 |  Method resolution order:
 |      Anak
 |      Ayah
 |      Ibu
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, nama)
```

Method resolution order dimulai dari Anak, lalu Ayah, dan terakhir adalah Ibu. Lalu, bagaimana jika kita ingin berbakti terlebih dahulu kepada Ibu kita lalu baru kemudian Ayah, maka kita hanya perlu menukar posisi Ayah dan Ibu pada penulisan class Anak menjadi seperti berikut ***class Anak (Ibu,Ayah).*** Perhatikan kode berikut:

```python
#Method Resolution Order
class Ayah:
    def __init__ (self, nama):
        self.namaAyah = nama

    def tampilkanNamaOrtu(self):
        print("Nama Ibu Saya: ",self.namaAyah)

class Ibu:
    def __init__ (self, nama):
        self.namaIbu = nama

    def tampilkanNamaOrtu(self):
        print("Nama Ibu Saya: ",self.namaIbu)

class Anak(Ibu,Ayah):
    def __init__ (self, nama):
        self.nama = nama

    def tampilkanNama(self):
        print("Nama Saya: ",self.nama)

anak = Anak("Muhammad")
anak.namaIbu = "Aminah"
anak.tampilkanNamaOrtu()
#help(anak)
```

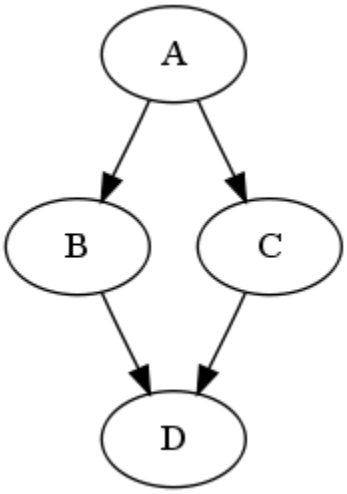
Output dari porgram diatas adalah sebagai berikut:

```
DATA SOLEH/0. KULIAH/KP/PY/oop5.py"
Nama Ibu Saya:  Aminah
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Fungsi tampilkanNamaOrtu() akan menjalankan fungsi yang terletak pada class Ibu, bukan pada class Ayah sesuai dengan Method resolution order. Untuk membuktikan hal tersebut silahkan kembali di nonaktifkan komentar pada baris kode ke 26.

4.4 Diamond Problem

Diamond Problem adalah sebuah kondisi dimana terdapat kasus polimorfisme pada hybrid inheritance yang berbentuk diamond. Gambar 8 menunjukan kondisi hybrid inheritance, dimana pada gambar tersebut terdapat 2 gabungan konsep pewarisan, yaitu hierarchical inheritance dan multiple inheritance.



Gambar 8. Diamond Problem pada kasus polimorfisme hybrid inheritance

Perhatikan kode berikut:

```python
#diamond problem
class A:
    def show(self):
        print("ini class A")

class B(A):
    def show(self):
        print("ini class B")

class C(A):
    def show(self):
        print("ini class C")

class D(B,C):
    pass

d = D()
d.show()
```

Secara intuisi berdasarkan pemahaman method resolution order yang sudah kita pelajari, kita akan mengetahui bahwa output dari program tersebut adalah:

```
DATA SOLEH/0. KULIAH/KP/PY/oop6.py"
ini class B
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Problem Solving – Computational Thinking

Bagaimana jika fungsi def show() pada class B kita ganti menjadi pass, maka apa output dari program tersebut ?

Bagaimana Method resolution order dari objek d tanpa menggunakan fungsi help(d) ?

4.5 Magic Method

Pada bagian ini kita akan mempelajari magic method yang terdapat pada python. Magic method adalah method yang diawali dan diakhiri dengan **double underscore**. Pada dasarnya kita telah menggunakan konsep magic method pada saat pembuatan fungsi konstruktor, dimana pembuatan fungsi konstruktor dibuat menggunakan sintaks **def __init__(self):**

Kita akan mempelajari magic method lainnya yang terdapat dalam Bahasa pemorgman python, diantaranya adalah;

1. def __repr__(self):
2. def __str__(self):
3. def __dict__(self):
4. def __add__ (self):

__repr__

Magic method ini akan merubah tampilan objek sesuai dengan format yang kita inginkan. Masih ingat dengan output dari kode ini ?

```
1  # Membuat template kelas MesinCetakUang
2  class MesinCetakUang:
3      pass
4
5  # Membuat objek pertama dari kelas MesinCetakUang
6  uang1 = MesinCetakUang()
7
8  # Mencetak objek uang1
9  print(uang1)
10
```

Yaps… potongan kode tersebut akan menghasilkan output sebagai berikut:

```
<__main__.MesinCetakUang object at 0x032191C0>
```

Kita bisa merubah tampilan tersebut menjadi tampilan yang lebih bermakna.

Perhatikan kode berikut:

```
1  #magic method
2  class MesinCetakUang():
3      def __repr__(self):
4          return "objek ini dihasilkan dari class MesinCetakUang"
5
6  uang = MesinCetakUang()
7
8  print(uang)
```

Sudah bisa menebak kan output nya akan seperti apa, ketika kita menampilkan objek uang, yaps outputnya adalah:

```
DATA SOLEH/0. KULIAH/KP/PY/oop6.py"
objek ini dihasilkan dari class MesinCetakUang
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Kita bisa juga menampilkan informasi dari atribut yang dimiliki oleh class tersebut. Sehingga ketika objek ditampilkan, maka output nya sesuai dengan keinginan pembuat program.

__str__

Magic method ini pada dasarnya sama seperti magic method sebelumnya, yaitu __repr__, hanya saja penggunaan magic method __repr__ digunakan untuk debugging, sedangkan **__str__ digunakan pada saat programnya sudah siap diproduksi**.

__dict__

Magic method ini pada dasarnya digunakan untuk menampilkan dictionary yang berisi namespace dari kelas.

```python
#magic method
class Uang():
    def __init__(self,nominal, logo):
        self.nominal = nominal
        self.logo = logo

    def __str__(self):
        return "Objek ini dihasilkan dari class Uang"

uang1 = Uang(1000,"Pattimura")
uang2 = Uang(100000,"Soekarno-Hatta")

print(uang1)
print(uang1.__dict__)
```

Output dari potongan kode diatas adalah:

```
Objek ini dihasilkan dari class Uang
{'nominal': 1000, 'logo': 'Pattimura'}
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

Perhatikan penggunaan magic method __str__ pada baris kode ke 7. Cara menggunakannya sama seperti magic method __repr__. Magic method __dict menampilkan dictionary yang berisi namespace dari kelas Uang yang ditandai dengan karakter kurung kurawal { }.

__add__

Magic method ini sangat berguna untuk operasi aritmatika, tidak hanya penjumlahan, tetapi ada juga pengurangan, perkalian, dan lain sebagainya. Perhatikan kode berikut:

```python
#magic method
class Uang():
    def __init__(self,nominal, logo):
        self.nominal = nominal
        self.logo = logo

    def __add__(self,objek):
        return self.nominal + objek.nominal

uang1 = Uang(1000,"Pattimura")
uang2 = Uang(100000,"Soekarno-Hatta")

print(uang1 + uang2)
```

Output dari program berikut adalah 101000.

Beberpa magic method yang lainnya adalah:

- __doc__ – mengakses string dokumentasi (docstring) dari kelas
- __name__ – nama kelas
- __module__ – nama modul tempat kelas didefinisikan. Nilai attribut ini di mode interaktif adalah "__main__".
- __bases__ – dasar dari kelas, bila kelas tidak merupakan turunan dari kelas lain, maka induknya dalah kelas object.
- dan masih banyak lagi yang lainnya

Problem Solving – Computational Thinking

Bagaimana penulisan magic method untuk operasi aritmatika yang lainnya seperti perkalian, pengurangan, dan pembagian?

4.6 Tugas Polimorfisme

Buat sebuah progam yang menggunakan konsep enkapsulasi sesuai dengan tema dan nomor terakhir NRP sebagai berikut:

NRP 0 – 3 : Bauh
NRP 4 – 6 : Hewan
NRP 7 – 9 : Sayur

Ketentuan:
- Gunakan fungsi input untuk mengisi nilai atribut pada objek
- Implementasikan semua konsep pada masing-masing sub bab polimorfisme
- Gunakan fungsi return dan fungsi void pada program

5. Abstraksi

Sub konsep terakhir dari pemrograman berorientasi objek adalah abstraksi / class abstract. Pemrograman berorientasi objek tidak hanya terjadi pada objek real, tetapi kita juga bisa membuat class abstrak. Abstraksi biasa disebut juga sebagai Generalisasi.

Gambar 9. Lukisan Abstrak

Class abtrak adalah prototype/blueprint dari class objek. Dalam Bahasa pemrograman python, Class abstrak menggunakan konsep pewarisan dalam implementasinya. Class abstrak tidak bisa menghasilkan objek, seperti halnya class objek yang selama ini kita gunakan. Class abstrak akan memaksa class objek yang menjadi subclass dari nya harus mengimplementasikan method yang terdapat di dalam class abstrak. Dengan adanya abstrak class, maka programmer bisa mengontrol jika lupa mengimplementasikan fungsi yang seharusnya ada pada class tertentu. Controh programmer akan membuat sebuah graphical user interface (GUI), dimana didalam gui terdapat berbagai jenis button, seperti buttonSubmit, buttonEdit, buttonDelete, dan sebagainya, masing-masing dari button tersebut harus memiliki fungsi click. Jadi ketika programmer lupa memberikan fungsi click pada salah satu sub class button, maka program akan error. Maka dari itu, dibuatlan konsep abstrak class. Abstrak class tidak hanya terdapat pada python, melainkan juga pada Bahasa pemrograman lain seperti java dan cpp. Kegunaan lain dari abstrak class adalah ketika programmer tidak membutuhkan objek dari class tersebut, tetapi objek dari subclass membutuhkan method yang terdapat di dalam

super class nya, maka implementasi abstrak class sangat cocok untuk digunakan.Berikut adalah implementasi dari class abstrak:

```
1  #membuat class abstrak
2  #abc = abstract base class
3  from abc import ABC , abstractclassmethod
4
5  class Lukisan(ABC):
6
7      @abstractclassmethod
8      def warna (self):
9          print("Sub class ini harus menggunakan fungsi warna")
10
11 class LukisanAbstrak (Lukisan):
12     def warna(self):
13         print("Dominasi warna lukisan adalah merah dan hijau")
14
15 lukisan = LukisanAbstrak()
16 lukisan.warna()
```

Perhatikan kode pada baris ke 3, abc merupakan abstrack base class. class Lukisan(ABC) merupakan abstrak class, dimana fungsi warna adalah abstrack method / abstractclassmethod. Pada kelas tersebut menerapkan prinsip pewarisan. Dimana class Lukisan merupakan subclass dari ABC (Abstrack Base Class), maka dari itu, class Lukisan merupakan abstrak class, sedangkan fungsi warna ditulis menggunakan decorator @abstractclassmethod yang menandakan bahwa fungsi tersebut adalah fungsi abstrak yang harus di implementasikan oleh subclassnya. Perhatikan pada class LukisanAbstrak yang merupakan subclass dari class Lukisan, maka class tersebut harus mengimplementasikan fungsi warna. Perhatikan juga bahwa objek dari class Lukisan tidak dibuat, karena memang tidak bisa atau tidak boleh.

Problem Solving – Computational Thinking

1. Untuk pembuktian, silahkan buat objek pada kode tersebut, lalu jalankan programnya, apakah yang akan terjadi?
2. Selain itu, silahkan hapus atau ganti fungsi warna pada class LukisanAbstrak, lalu jalankan programnya, apakah yang akan terjadi?

5.1. Prototype Class

Pada dasarnya abstract class dibuat sebagai prototype dari class yang menjadi subclassnya. Pada implementasinya, method yang terdapat di dalam class abstract tidak memiliki implementasi apapun di dalam nya, implementasinya akan dijelaskan dimasing-masing subclassnya. Contoh kasusnya kita akan membuat sebuah class Hewan sebagai abstrack class, dan berbagai hewan lain sebagai class instance nya. Perhatikan kode berikut:

```
1  from abc import ABC, abstractmethod
2  class Hewan(ABC):
3      @abstractmethod
4      def prilaku(self):
5          pass
6
7  class Ular(Hewan):
8      def prilaku(self):
9          print("Saya berbisa dan melata")
10
11 class Anjing(Hewan):
12     def prilaku(self):
13         print("Penciuman saya tajam")
14
15 ular = Ular()
16 ular.prilaku()
17 anjing = Anjing()
18 anjing.prilaku()
```

Perhatikan pada baris ke 3. Penulisan decorator untuk abstrack method terdapat 2 cara, yang pertama adalah @abstrackmethod / abstractclassmethod. Pada fungsi Prilaku yang terdapat pada baris kode ke 4 dalam class abstrak hanya berisi pass, karena class abstrak memang dibuat untuk prototype saja bagi sub classnya, jadi implementasinya terdapat di dalam class Ular dan class Anjing.

5.2. Tugas Pemrograman

Buat sebuah kasus yang mengimplementasikan class abstract.

6. Hubungan antar objek

Pada dasarnya semua konsep dari pemrograman berorientasi objek telah kita bahas dari mulai bab 1 hingga bab 5. Pada bab kali ini, kita akan membahas hubungan antar objek, karena bagaimanapun, objek-objek yang ada dalam Bahasa pemrograman tidak berdiri sendiri-sendiri melainkan saling berinteraksi antara satu dengan yang lainnya, sebagaimana objek-objek dalam kehidupan nyata.

Gambar 10 Hubungan interaksi antar manusia

Pada bahasan kali ini, Beberapa hubungan antar objek yang akan kita pelajari adalah; Asosiasi, Agregasi, dan Komposisi.

6.1 Asosiasi

Asosiasi adalah hubungan antar dua objek yang saling **"berinteraksi"** antara satu dengan yang lainnya. Contoh hubungan asosiasi adalah hubungan antara class murid dan class guru. Guru mengajar murid, sedangkan murid diajar oleh guru. Contoh lain dari hubungan asosiasi adalah pedagang dan pembeli. Pedagang menjajakan jualannya kepada pembeli, sedangkan pembeli membeli barang dagangan yang dijual oleh penjual. Pada kasus ini, kita akan membahas hubungan asosiasi antara penjual dan pembeli. Pengembangan dari kasus ini masih bisa diperluas dan diperkaya dengan

berbagai pengetahuan lainnya terkait dengan konsep pemrograman berorientasi objek. khususnya 4 sub pokok bahasan dalam pemrogaman berorientasi objek yaitu Pewarisan, Enkapsulasi, Polimorfisme, dan abstraksi.

```
1  class Penjual():
2      def __init__ (self, nama, barang, harga):
3          self.nama = nama
4          self.barang = barang
5          self.harga = harga
6
7      def jual(self, pembeli):
8          print(self.nama, "menjual", self.barang, "kepada" , pembeli,\
9              "seharga", self.harga)
10
11 class Pembeli():
12     def __init__ (self,nama):
13         self.nama = nama
14
15 def main():
16     print("Hubungan asosiasi antara penjual dan pembeli")
17     arif = Penjual("Arif","Semangka",10000)
18     budi = Pembeli("Budi")
19     arif.jual(budi.nama)
20
21 main()
```

Perhatikan pada baris kode 8, jika kita ingin menulis code menjadi beberapa baris, agar kode terlihat rapih dan mudah dibaca, kita bisa menggunakan backslace. Pada kode diatas, terdapat dua buah class, yaitu class Penjual dan class Pembeli. Pada fungsi jual yang terdapat di class Penjual, terdpat argument pembeli, dimana argument tersebut akan melibatkan class Pembeli.

Selain kedua class tersebut, terdapat fungsi main yang berfungsi untuk menjalankan hubungan antar kedua objek tersebut. Obejk arif merupakan instance dari class Penjual, sedangkan objek budi adalah instance dari class Pembeli. Arif menjual barang daganagnnya yaitu Semangka kepada Pembeli (Budi) seharga Rp. 10.000. Output dari program diatas adalah sebagai berikut:

```
Hubungan asosiasi antara penjual dan pembeli
Arif menjual Semangka kepada Budi seharga 10000
PS C:\0. DATA SOLEH\0. KULIAH\KP\PY>
```

6.2 Agregasi

Agregasi adalah hubungan antar dua objek yang saling **"memiliki"** antara satu dengan yang lainnya. Contoh dari kasus agregasi adalah Karyawan memiliki Honor. Owner atau pemilik objek dalam kasus ini adalah Karyawan, sedangkan objek yang dimiliki dalam kasus ini adalah Honor. Karyawan / Owner disebut juga sebagai ***aggregating object,*** sedangkan Honor disebut sebagai ***aggregated object***.

```
1  class Karyawan():
2      def __init__(self, nama, nip):
3          self.nama = nama
4          self.nip = nip
5
6      def gajian(self, honor):
7          totalGaji = honor.gajiPokok + honor.lembur + honor.tunjangan
8          totalGaji = totalGaji - honor.pajak
9          print(self.nama, "mendapatkan Honor sebanyak", totalGaji)
10
11 class Honor():
12     def __init__(self, gajiPokok, lembur, tunjangan, pajak):
13         self.gajiPokok = gajiPokok
14         self.lembur = lembur
15         self.tunjangan = tunjangan
16         self.pajak = pajak
```

```
17
18 def main():
19     arif = Karyawan("Arif", "001")
20     honor = Honor(4000000,100000,200000,50000)
21     arif.gajian(honor)
22
23 main()
```

Pada kode diatas, diketahui bahwa arif merupakan objek yang dibangkitkan dari class Karyawan. Dimana Setiap Karyawan memiliki Honor yang akan dibayarkan setiap akhir bulan atau pada tanggal tertentu sesuai keputusan perusahaan. Objek honor dibangkitkan oleh class Honor dan menjadi aggregated object dari class Karyawan yang dipanggil dari fungsi gajian. Output dari program tersebut adalah:

```
Arif mendapatkan Honor sebanyak 4250000
```

6.3 Komposisi

Komposisi adalah hubungan yang menyatakan bahwa sebuah aggregated objek hanya dimiliki oleh owner / aggregating object tertentu. Jadi komposisi merupakan bentuk khusus dari agregasi. Contoh dari kasus Komposisi adalah WNI dan KTP. Dimana setiap WNI Pasti memiliki KTP. Dan KTP hanya dimiliki oleh WNI. Tidak dimiliki oleh objek lainnya.

```
1  class WNI():
2      def __init__ (self,nama):
3          self.nama = nama
4          self.ktp = KTP(34010212890003,"RI210102")
5
6  class KTP():
7      def __init__ (self,noKTP,idKTP):
8          self.noKTP = noKTP
9          self.idKTP = idKTP
10
11     def cetakKTP(self):
12         print("Nomor KTP ini:", self.noKTP, "dengan id:" ,self.idKTP)
13
14 def main():
15     budi = WNI("Budi")
16     budi.ktp.cetakKTP()
17
18 main()
```

Perhatikan pada kode baris ke 4. Objek WNI memiliki atribut yang dibangkitkan dari objek KTP. Jadi selain objek diparsing melalui argument, objek juga dapat menjadi atribut dari objek yang lainnya. Cara mengakses atribut dan fungsi dari objek class yang dijadikan atribut oleh class lain mengunakan dot (titik). Perhatikan baris kode 16. Objek budi mengakses atribut ktp, dimana atribut ktp merupakan sebuah objek yang dibangkitkan dari class KTP, dimana pada class KTP terdapat fuingsi cetakKTP(), maka cara mengaksesnya menggunakan dot sererti pada baris kode 16.

Output dari program diatas adalah:

```
Nomor KTP ini: 34010212890003 dengan id: RI210102
```

Agar program menjadi lebih dinamis, maka kita akan menggunakan fungsi input yang akan mengisi nilai noKTP dan idKTP. Perhatikan kode berikut ini:

```python
class WNI():
    def __init__ (self,nama):
        self.nama = nama
        self.ktp = KTP(None,None)

class KTP():
    def __init__ (self,noKTP,idKTP):
        self.noKTP = noKTP
        self.idKTP = idKTP

    def cetakKTP(self):
        print("Nomor KTP ini:", self.noKTP, "dengan id:" ,self.idKTP)

def main():
    budi = WNI("Budi")
    budi.ktp.noKTP = input("Silahkan masukan no KTP: ")
    budi.ktp.idKTP = input("Silahkan masukan id KTP: ")
    budi.ktp.cetakKTP()

main()
```

Output dari kode tersebut sama seperti kode sebelumnya, hanya saja noKTP dan idKTP merupakan input dari keyboard yang diketik oleh user. Sehingga user bisa mengisi noKTP dan idKTP sesuai kebutuhan.

6.4 Tugas Pemrograman

Buat sebuah kasus yang mengimplementasikan hubungan antar objek. Agar program menjadi dinamis, gunakan fungsi input untuk pembuatan objek nya.

7. Class Diagram
8. Pemrograman Berorientasi Objek & Struktur Data
9. Graphical User Interface (GUI) dengan TkInter
10. Pemrograman Berorientasi Objek dengan GUI TkInter