UNIVERSITAS MIKROSKIL

Computer Laboratory Handbook

Perancangan dan Pemrograman Berbasis Objek

IF2105

By:

Apriyanto Halim, S.Kom., M.Kom.

Course Version: 2021

COURSE OVERVIEW

COURSE OVERVIEW

Mata kuliah ini membekali mahasiswa dengan pengetahuan dan keterampilan mengenai konsep dari pemrograman berbasis objek dan lanjutannya pada konsep Solid dan Design Pattern. Dengan mempelajari mata kuliah ini, mahasiswa diharapkan dapat mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari.

COURSE GOALS

Adapun tujuan yang ingin dicapai pada matakuliah ini, yaitu :

- Memimpin dan bekerja dalam tim, mandiri, dan bertanggung jawab terhadap pekerjaannya.
- Berpikir kreatif dan inovatif.
- Mampu bertanggung jawab atas pencapaian hasil kerja kelompok dan melakukan supervisi serta evaluasi terhadap penyelesaian pekerjaan yang ditugaskan kepada pekerja yang berada di bawah tanggung jawabnya.
- Menguasai konsep teoritis bidang pengetahuan Teknik Informatika secara umum dan konsep teoritis bagian khusus dalam bidang pengetahuan Teknik Informatika secara mendalam, serta mampu memformulasikan penyelesaian masalah prosedural.
- Mempunyai pengetahuan dalam penyusunan algoritma pemrograman yang efektif dan efisien serta dapat merancang, membangun, dan mengelola solusi perangkat lunak yang orisinal dan holistik terhadap permasalahan bisnis ataupun ilmiah.

COURSE OBJECTIVES

Pencapaian yang diharapkan dapat dicapai oleh mahasiswa, adalah :

- Mahasiswa mampu menerapkan konsep kelas dan objek.
- Mahasiswa mampu menerapkan konsep inheritance dan polymorphism.
- Mahasiswa dapat menangani masalah dengan tepat menggunakan Exception.
- Mahasiswa mampu memanfaatkan penggunakan Iterator Pattern.
- Mahasiswa mampu menerapkan Design Pattern.
- Mahasiswa mengetahui langkah-langkah yang digunakan dalam melakukan testing pada OOP.
- Mahasiswa mampu menerapkan prinsip SOLID pada kode yang dibuat.

REQUIREMENT

Sebelum kita masuk ke dalam modul ini, ada beberapa kebutuhan yang harus kalian lakukan, yaitu :

- Pastikan kalian sudah melakukan instalasi Python versi 3 keatas.
- Untuk UML yang digunakan menggunakan Class Diagram.
- Untuk software pembuatan UML berupa website Draw.IO.
- Untuk software pembuatan kode Python berupa Visual Studio Code.

DAFTAR ISI

UNIT 2
INHERITANCE DAN POLYMORPHISM (2)

UNIT OVERVIEW

Pada chapter kali ini kita akan membahas bagaimana cara membuat sebuah fungsi yang dapat di daur ulang menjadi lebih spesifik dan lebih rinci sehingga bisa lebih jelas fungsi untuk digunakan. Pada chapter ini juga kita akan mempelajari kelas abstrak yang berfungsi memberikan karakteristik yang harus dipenuhi kelas keturanan tersebut, sehingga tujuan dari pembuatan kelas dapat tercapai.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Membuat method overridding dan overloading.
- Membuat kelas abtraks.

UNIT CONTENTS

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa yang bisa kita sebut sebagai inheritance?
2. Menurut kalian, apakah ayah kepada paman merupakan inheritance?
3. Dari daftar nama-nama berikut, kelas induk apa yang bisa kita buat untuk menampung objek-objek berikut :
   - Masker
   - Hand sanitizer
   - Desinfektan
   - Tisu basah
   - Alkohol

CONTENT LESSON

CASE STUDY / PROJECT

Kita kembali pada permasalahan yang dihadapi oleh perpustakaan “Home Learning is Best”. Karena, sebelumnya sudah membuka cabang pada beberapa daerah dan setiap daerah memiliki ciri khas yang berbeda, perpustakaan tersebut mengalami kendala dalam pencetakan brosur dan ketikdak konsistenan dalam data serta tindakan yang dilakukan. Oleh karena itu, untuk setiap cabangnya, perpustakaan tersebut mengharuskan :

1. Harus mempunyai 1 buah data berupa data buku, dalam bentuk nama buku.
2. Harus dapat mencetak buku tersebut supaya bisa melakukan pendataan yang disesuaikan dengan daerah masing-masing, seperti :
   a. Daerah Jawa, harus bisa mencetak jenis gamelan yang ada.
   b. Daerah Batak, harus bisa mencetak jenis ulos yang dibuat.

Tentunya hal ini bertujuan supaya proses yang dilakukan dapat berjalan dengan lancar serta bisa diproses seperti perpustakaan pusat dan mudah untuk dikelola.

Nah kali ini agak sedikit berbeda karena ada syarat yang harus dipenuhi untuk setiap cabangnya. Gimana ya cara membuat syarat tersebut di dalam kelas?

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus tersebut, ada beberapa hal yang harus diperhatikan, yaitu ada syarat yang harus dipenuhi serta ada proses yang berbeda namun memiliki tujuan yang sama. Nah, pertanyaan muncul bagaimana cara kita membuat syarat? Terus, gimana cara buat dua tindakan yang sama tapi memiliki proses yang berbeda?

Dari pada pusing, yuk simak penjelasan berikut !

LESSON 3: POLYMORPHISM

LESSON 3: OVERVIEW

Seperti yang telah dijelaskan diawal, dimana polymorphism merupakan sebuah proses yang dapat kita gunakan untuk membuat 2 buah tindakan dengan nama yang sama namun memiliki proses dan tujuan yang mungkin berbeda. Kenapa bisa begitu? Dan gimana caranya? Sebelum kita lanjut, kita lihat baris kode yang ada dibawah ini :

```
class kakek:
    def panggil(self):
        print("Panggil saya kakek ya!")

class ayah(kakek):
    def panggil(self):
        print("Panggil saya ayah donk!")

class paman(kakek):
    def panggil(self, namaIstri):
        print(f"Tante saya bernama {namaIstri}")
```

Figure 1 Contoh Dasar Polymorphism

Pada contoh tersebut terdapat 1 kelas induk, yaitu kakek dan 2 kelas keturunan, yaitu ayah dan paman. Dimana kelas ayah dan kelas paman merupakan keturunan dari kelas kakek. Hanay saja, ada yang membedakan dari kelas tersebut, yaitu method panggil yang digunakan. Dimana setiap kelas keturunan memiliki nama method yang sama. Apakah ini bisa berjalan?

Yuk coba kita jalankan terlebih dahulu, dan kita lihat gimana hasilnya! Silahkan perhatikan data berikut :

```
>>> k = kakek()
>>> a = ayah()
>>> p = paman()
>>> k.panggil()
Panggil saya kakek ya!
>>> a.panggil()
Panggil saya ayah donk!
>>> p.panggil()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: panggil() missing 1 required positional argument: 'namaIstri'
>>> p.panggil("Donita")
Tante saya bernama Donita
>>>
```

Figure 2 Hasil Kode Polymorphism

Dari hasil kode yang kita jalankan, ada sedikit keanehan pada kelas ayah dan kelas paman. Dimana hasil diberikan tidak sama. Mengapa demikian? Inilah merupakan konsep dari kelas Morphism yang bertujuan untuk membuat nama method yang sama namun memiliki tingkahlaku/tindakan yang berbeda. Hal ini bertujuan supaya kita dapat membuat sebuah fungsi yang lebih spesifik dan lebih mudah untuk dimengerti oleh programmer maupun pengguna yang menggunakan.

Hal ini tentunya dapat membantu kita dalam menangani proses yang mengharuskan kita untuk menggunakan nama method yang ada, namun memiliki tingkah laku yang berbeda. Sebagai contoh proses perhitungan luas pada **persegi** dengan **persegi panjang**. Dimana, kedua bidang tersebut merupakan bentuk dari segi empat yang memiliki tindakan berupa **luas** dan **keliling**. Hanya saja dalam proses pencarian yang dilakukan untuk kedua tersebut menggunakan cara yang berbeda. Silahkan perhatikan potongan kode berikut :

```
class segiEmpat:
    def __init__(self, sisi1, sisi2, sisi3, sisi4):
        self.sisi1 = sisi1
        self.sisi2 = sisi2
        self.sisi3 = sisi3
        self.sisi4 = sisi4
    def luas(self):
        return 0
    def keliling(self):
        return 1

class persegi(segiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi1
    def keliling(self):
        return 4 * self.sisi1

class persegiPanjang(segiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi2
    def keliling(self):
        return 2 * (self.sisi1 + self.sisi2)
```

Figure 3 Contoh Polymorphism (2)

Pada kelas tersebut terlihat jelas, bahwa ada terdapat 1 kelas induk berupa kelas segiEmpat dan 2 kelas keturunan berupa kelas persegi dan kelas persegiPanjang. Hanya saja pada kedua kelas keturunan memiliki method dengan yang sama, hanya saja proses yang ada kelas keturunan tersebut yang berbeda. Apakah hasil keluarannya akan sama atau berbeda?

Untuk tidak menjadi tanda tanya, mari kita jalankan kode tersebut :

```
>>> a = segiEmpat(10, 20, 30, 40)
>>> b = persegi(10, 20, 30, 40)
>>> c = persegiPanjang(10, 20, 30, 40)
>>> a.luas()
0
>>> a.keliling()
1
>>> b.luas()
100
>>> b.keliling()
40
>>> c.luas()
200
>>> c.keliling()
60
>>>
```

Figure 4 Hasil Polymorphism (2)

Pada keluaran kode tersebut terlihat meskipun data yang kita input sama namun, tetap memiliki keluaran yang berbeda bergantung dari tindakan yang kita lakukan pada masing-masing kelas tersebut.

Namun, bagaimana cara kita membuat syarat untuk kelas-kelas tertentu, sehingga kelas tersebut dibuat dengan sesuai yang kita harapkan? Untuk menjawab hal ini, mari kita pelajari lesson berikutnya!

LESSON 4: MEMBUAT ABSTRAK CLASS

LESSON 4: OVERVIEW

Sebagai cara untuk membuat kriteria yang harus dibuat supaya kelas tersebut sesuai dengan yang kita harapkan, maka kita membutuhkan ABC (Abstract Base Class). ABC merupakan sebuah kelas yang dapat kita gunakan untuk membuat persyaratan terhadap kelas yang ingin kita buat. Pada kelas ABC ini, nanti aka nada sekumpulkan method dan properties yang harus diterapkan pada kelas yang menggunakan kelas ini. Sebagai contoh perhatikan kode berikut :

```python
import abc

class syaratSegiEmpat(metaclass=abc.ABCMeta):
    # Untuk syaratnya saya buat dalam bentuk komen berikut
    # Untuk syarat yang pertama kelas tersebut harus ada 2 propeties (atibut) sisi1 dan sisi2
    @abc.abstractproperty
    def sisi1(self):
        pass
    @abc.abstractproperty
    def sisi2(self):
        pass

    # Untuk syarat berikut harus memiliki fungsi berupa luas dan keliling
    @abc.abstractmethod
    def luas(self):
        pass
    @abc.abstractmethod
    def keliling(self):
        pass
```

Figure 5 Kelas Abstract (ABC)

Untuk menggunakan abstract class, terlebih dahulu kita harus memanggil module abc yang nantinya akan kita gunakan untuk membuat kriteria dari sebuah kelas. Pada kelas syaratSegiEmpat, memiliki kelas induk berupa **metaclass=abc.ABCMeta**, bagian ini tidak akan dijelaskan lebih lanjut. Karena, bagian ini tidak begitu sering digunakan untuk pelajaran-pelajaran berikutnya. Hanya bagi kalian yang ingin berlajar lebih lanjut, bisa mencari pada Google atau website pencarian lainnya.

Pada kelas tersebut, kita berfokus kepada bagian **@abc.abstractproperty** dan **@abc.abstractmethod** yang biasanya kita sebut sebagai *constructs*. Bagian ini merupakan bagian yang sangat penting untuk membuat method dan properties menjadi abstrak atau menjadi sebuah syarat dalam pembuatan sebuah kelas. Terus, gimana cara kita bisa menggunakannya? Perhatikan kode berikut :

```
from abstract import syaratSegiEmpat

class segiEmpat:
    def __init__(self, sisi1, sisi2):
        self.sisi1 = sisi1
        self.sisi2 = sisi2

class persegi(segiEmpat, syaratSegiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi2

class persegiPanjang(segiEmpat, syaratSegiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi2
    def keliling(self):
        return 2 * (self.sisi1 + self.sisi2)
```

Figure 6 Mengakses Kelas Abstract

NB : Pada konsep penerapan ini, kita sebenarnya bisa membuat 1 kelas turunan yang memiliki 2 kelas induk seperti pada contoh diatas. Kita hanya perlu menambahkan tanda koma dan dilanjutkan dengan nama kelas induk berikutnya. Sebagai contoh : **class kelasTurunan(kelasInduk1, kelasInduk2)**. Hal ini merupakan merupakan penerapan dari multi inheritance.

Dari kelas tersebut terlihat kita melakukan import modul yang ada pada file abstract yang telah saya buat sebelumnya, dan ada terdapat 1 kelas induk berupa segiEmpat, 1 kelas abstract berupa syaratSegiEmpat dan 2 kelas keturunan, yaitu persegi dan persegiPanjang. Kedua kelas turunan tersebut merupakan kelas turunan dari kelas segiEmpat dan kelas syaratSegiEmpat. Terus, gimana hasilnya? Apakah akan sama atau terjadi error (kesalahan dalam program)? Mari kita lihat hasilnya sebagai berikut :

```
>>> a = persegi(10, 20)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class persegi with abstract methods keliling, sisi1, sisi2
>>> b = persegiPanjang(10, 20)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class persegiPanjang with abstract methods sisi1, sisi2
>>>
```

Figure 7 Hasil Keluaran Kelas Abstract (1)

Ternyata terjadi kesalahan dalam program ketika kita ingin melakukan inisialisasi ke dalam kelas tersebut. Mengapa demikian? Coba perhatikan kembali bagian kelas syaratSegiEmpat! Apa saja syarat yang kita buat sebelumnya? Ternyata harus ada 2 properties berupa sisi1 dan sisi2 serta harus ada 2 buah method berupa luas dan keliling. Nah, mari kita coba sesuaikan kembali, dan lihat hasilnya gimana. Untuk hasil perbaikan kelas tersebut, seperti tampilan berikut :

```
from abstract import syaratSegiEmpat

class segiEmpat:
    def __init__(self, sisi1, sisi2):
        self.sisi1 = sisi1
        self.sisi2 = sisi2

class persegi(segiEmpat, syaratSegiEmpat):
    sisi1 = 0
    sisi2 = 0
    def luas(self):
        return self.sisi1 * self.sisi2
    def keliling(self):
        return 4 * self.sisi1

class persegiPanjang(segiEmpat, syaratSegiEmpat):
    sisi1 = 0
    sisi2 = 0
    def luas(self):
        return self.sisi1 * self.sisi2
    def keliling(self):
        return 2 * (self.sisi1 + self.sisi2)
```

Figure 8 Mengakses Kelas Abstract (Perbaikan)

Dari kelas turunan telah kita buat sesuai dengan syaratnya. Nah, gimana kalau kita jalankan kembali? Apakah masih ada error? Silahkan perhatikan keluaran berikut :

```
>>> a = persegi(10, 10)
>>> b = persegiPanjang(10, 20)
>>> a.luas()
100
>>> a.keliling()
40
>>> b.luas()
200
>>> b.keliling()
60
>>>
```

Figure 9 Hasil Keluaran Kelas Abstract (2)

Ternyata setelah kita jalankan kembali, sudah tidak terjadi error. Oleh karena itu, penting untuk kita pahami kembali bagaimana cara membuat dan tentunya hal ini akan sangat membantu bagi programmer dalam membuat kelas sesuai dengan yang kita harapkan.

Nah bagaimana sekarang, apakah sudah siap untuk menjawab studi kasus kita?

SOLUTION

Setelah kita membaca berbagai macam topik diatas, yuk kita kembali ke studi kasus kita diawal.

Setelah kita baca kembali studi kasus tersebut, ada hal atau syarat yang harus kita lakukan supaya bisa menjawab permasalahan tersebut. Hal ini tentunya dapat terselesaikan dengan menerapkan penerapan dari kelas abstract yang telah kita pelajarin sebelumnya. Setelah itu, ada 1 buah method yang memiliki nama yang sama, hanya saja proses yang dilakukan berbeda-beda. Tentunya untuk mengatasi masalah tersebut kita bisa menerapkan prinsip dari polymorphism.

Nah setelah kita pelajari semuanya, mari kita jawab studi kasus tersebut!

INSTRUCTION

1. Terlebih dahulu mari kita buat dulu kelas abstract sesuai dengan syarat yang ada pada studi kasus, dimana syaratnya, yaitu :

- Harus mempunyai 1 buah data berupa data buku, dalam bentuk nama buku.
- Harus dapat mencetak buku tersebut supaya bisa melakukan pendataan.

```
import abc

class syaratPerpus(metaclass=abc.ABCMeta):
    @abc.abstractproperty
    def namaBuku(self):
        pass

    @abc.abstractmethod
    def cetakData(self):
        pass
```

2. Setelah kita membuat syaratnya, selanjutnya kita buat terlebih dahulu kelas induk perpus sebagai dasar dalam pembuat kelas untuk kelas keturunannya.

```
class perpus:
    def __init__(self, jlhBuku):
        self.jlhBuku = jlhBuku
```

3. Setelah kita membuat kelas induknya, selanjutnya kita membuat kelas untuk kelas keturunannya, yaitu perpusJawa dan perpusBatak. Namun, pada kelasnya jangan lupa juga kita tambahkan sesuai dengan studi kasus, berupa :

- Daerah Jawa, harus bisa mencetak jenis gamelan yang ada.
- Daerah Batak, harus bisa mencetak jenis ulos yang dibuat.

```
class perpusJawa(perpus, syaratPerpus):
    namaBuku = ""
    def __init__(self, namaBuku, jlhBuku):
        super().__init__(jlhBuku)
        self.namaBuku = namaBuku
    def cetakData(self, jenisGamelan):
```

```
        print("Pada Perpus Jawa")
        print(f"Terdapat buku berjudul {self.namaBuku} dengan jumlah {self.jlhBuku} buku")
        print(f"Dan terdapat jenis gamelan berupa {jenisGamelan}")

class perpusBatak(perpus, syaratPerpus):
    namaBuku = ""
    def __init__(self, namaBuku, jlhBuku):
        super().__init__(jlhBuku)
        self.namaBuku = namaBuku
    def cetakData(self, jenisUlos):
        print("Pada Perpus Batak")
        print(f"Terdapat buku berjudul {self.namaBuku} dengan jumlah {self.jlhBuku} buku")
        print(f"Dan terdapa ulos yang berasal dari {jenisUlos}")
```

4. Setelah semua selesai, yuk kita jalankan dan lihat gimana hasilnya :

```
>>> a = perpusJawa("Habis Gelap Terbi Terang", 10)
>>> b = perpusBatak("Pedoman Upacara Batak", 5)
>>> a.cetakData("Bonang")
Pada Perpus Jawa
Terdapat buku berjudul Habis Gelap Terbi Terang dengan jumlah 10 buku
Dan terdapat jenis gamelan berupa Bonang
>>> b.cetakData("Batak Toba")
Pada Perpus Batak
Terdapat buku berjudul Pedoman Upacara Batak dengan jumlah 5 buku
Dan terdapa ulos yang berasal dari Batak Toba
>>>
```

Figure 10 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada kelas yang ada di Mikroskil, ada dosen dan mahasiswa yang memiliki identitas yang berbeda-beda. Hal ini bertujuan supaya proses yang dilakukan jelas dan tidak bertumpang tindih. Namun, pada data mahasiswa dan dosen harus ada berupa nama dan nomor identitas. Dan untuk melakukan absensinya kita harus membedakan antara dosen dan mahasiswa. Dimana ketika dosen melakukan absensi maka muncul pesan berupa “Silahkan mulai pelajaran”. Sedangkan ketika mahasiswa melakukan absensi maka akan muncul pesan berupa “Terima kasih sudah hadir”.

Untuk menyelesaikan ini, harus ada 1 buah abstract class, 1 buah kelas induk dan 2 buat kelas turunan. Untuk penamaan kelas induk dan abstract class dibebaskan. Jadi jangan sampai dijumpain nama abastract class dan kelas induk yang sama.

TASK 2:

Pada sebuah private les, terdapat berbagai murid yang memiliki berbagai tingkatan yang berbeda diantaranya : SD, SMP dan SMA. Hal ini tentunya disesuiakan juga dengan biaya les yang diberikan, dimana biaya disesuaikan dengan data berikut :

Tingkatan	Biaya Les
SD	Rp. 500.000,00
SMP	Rp. 800.000,00
SMA	Rp. 1.200.000,00

Setelah itu, untuk data dari murid les, harus melengkapi data berupa Nama, dan NO HP orangtua yang digunakan. Hal ini bertujuan supaya ketika murid les sudah selesai dapat menghubungi orangtua masing-masing supaya bisa dijemput untuk pulang.

Untuk data tambahan yang dibutuhkan oleh private les tersebut disesuai juga berdasarkan tingkatannya. Dimana :

- Tingkat SD diperlukan data tambahan berupa Jenis Kelamin.
- Tingkat SMP diperlukan data tambahan berupa Umur.
- Tingkat SMA diperlukan data tambahan berupa jenis kelas (IPA atau IPS).

Untuk menyelesaikan ini, harus ada 1 buah abstract class, 1 buah kelas induk dan 3 buat kelas turunan. Untuk penamaan kelas induk dan abstract class dibebaskan. Jadi jangan sampai dijumpain nama abastract class dan kelas induk yang sama.

UNIVERSITAS
MIKROSKIL