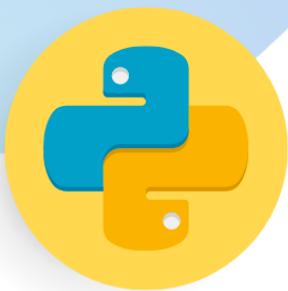




PTI
KREATIF



MODUL PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

Disusun Oleh :
Arif Setiawan, S.Kom., M.Eng



PENDIDIKAN TEKNIK INFORMATIKA
FAKULTAS KEGURUAN DAN ILMU PENDIDIKAN
UNIVERSITAS MUHAMMADIYAH SURAKARTA

Modul Praktikum
Pemrograman Berorientasi Objek
2023

Disusun Oleh :
Arif Setiawan, S.Kom., M.Eng

Diterbitkan oleh:
Program Studi Pendidikan Teknik Informatika
Fakultas Keguruan dan Ilmu Pendidikan
Universitas Muhammadiyah Surakarta
Jln. A Yani Pabelan Kartasura Surakarta 57102

Kata Pengantar

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas segala limpahan rahmat dan karunia-Nya sehingga modul "Praktikum Pemrograman Berorientasi Objek" ini dapat diselesaikan dengan baik. Modul ini disusun untuk memberikan panduan praktis kepada mahasiswa dalam memahami dan mengimplementasikan konsep dasar pemrograman berorientasi objek (OOP) yang sangat penting dalam pengembangan perangkat lunak modern. Kami berharap modul ini dapat menjadi sumber belajar yang efektif dalam membantu mahasiswa menguasai teknik-teknik dasar OOP melalui berbagai kegiatan praktikum yang terstruktur.

Modul ini terdiri dari beberapa bab yang mencakup berbagai topik seperti pengenalan class dan object, inheritance, polymorphism, encapsulation, serta penerapan prinsip-prinsip SOLID dalam desain perangkat lunak. Setiap bab dilengkapi dengan tujuan pembelajaran, materi teori, contoh kode, serta kegiatan praktikum yang dirancang untuk mengembangkan kemampuan analitis dan keterampilan pemrograman mahasiswa. Kami berharap modul ini dapat menjadi acuan yang bermanfaat bagi mahasiswa dalam proses belajar mengajar dan memberikan kontribusi positif dalam pengembangan kompetensi pemrograman mahasiswa.

Surakarta, 14 Februari 2023

Tim Penyusun

Daftar Isi

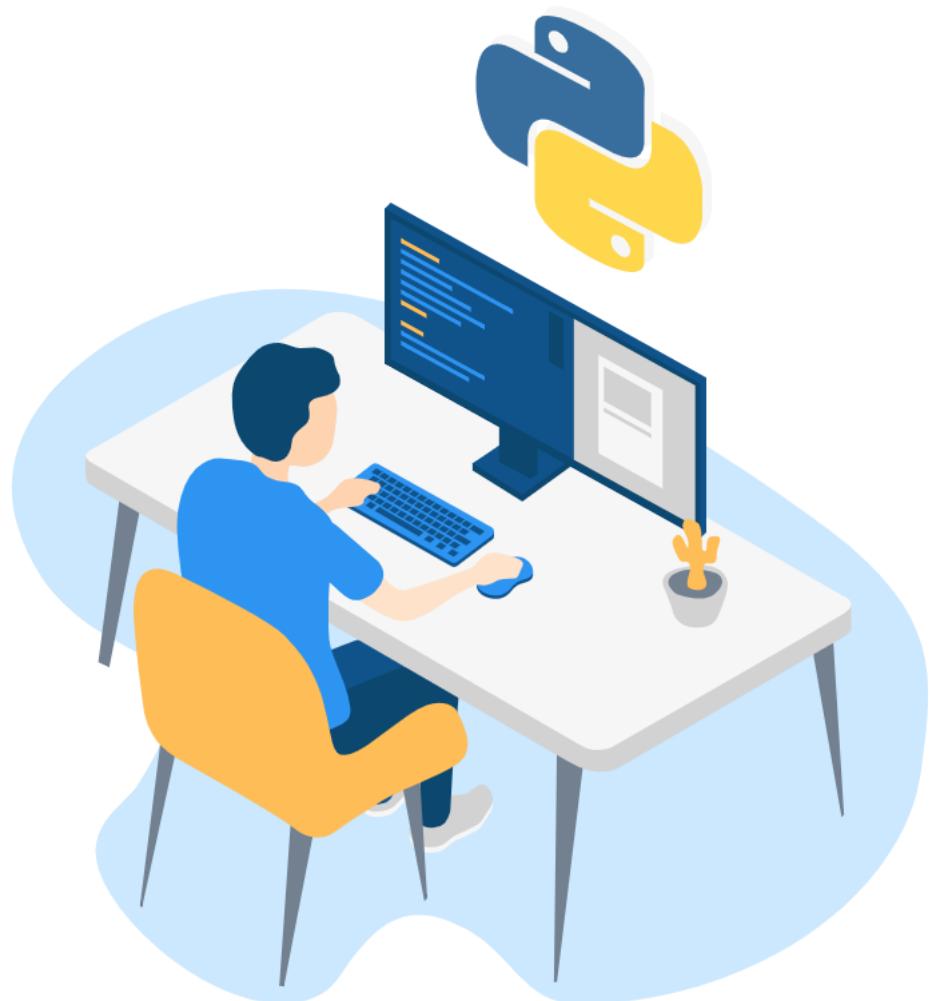
BAB 1 Class dan Object.....	1
1.1 Tujuan	1
1.2 Pengantar	2
2.1 Kegiatan Praktikum.....	6
2.1.1 Kegiatan 1 : Class dengan satu instance.....	6
2.3.2 Kegiatan 2 : Class dengan dua instance.....	7
2.3.3 Kegiatan 3 : Mendalami Class.....	8
2.4 Tugas.....	9
BAB 2 Exception Handling.....	10
2.1 Tujuan	10
2.2 Pengantar	11
2.2.1 Error dan Exception.....	11
2.2.2 Syntax Error.....	11
2.2.3 Exception.....	12
2.2.4 Exception Handling.....	13
2.2.5 Tipe Exception Pada Python	14
2.3 Kegiatan Praktikum.....	16
2.3.1 Kegiatan 1 : Exception Handler pada Python.....	16
2.3.2 Kegiatan 2 : Exception Handler dengan konsep OOP.....	16
2.3.3 Kegiatan 3 : Raise Exception Handler.....	17
2.4 Tugas.....	18
BAB 3 Encapsulation	19
3.1 Tujuan	19
3.2 Pengantar	20
3.2.1 Encapsulation	20
3.2.2 Private Variable	21
3.2.3 Getter dan Setter	22
3.2.4 Abstraction	24
3.3 Kegiatan Praktikum.....	25
3.3.1 Kegiatan 1 : Implementasi Private Variabel.....	25
3.3.2 Kegiatan 2 : Implementasi Getter dan Setter.....	26
3.3.3 Kegiatan 3 : Abstraction pada OOP	27
3.4 Tugas.....	28
BAB 4 Polymorphism.....	29
4.1 Tujuan	29
4.2 Pengantar	30
4.2.1 Polymorphism.....	30
4.2.2 Polymorphism pada Python.....	31
4.2.3 Overloading Polymorphism.....	32
4.2.4 Overriding Polymorphism	33
4.3 Kegiatan Praktikum.....	34
4.3.1 Kegiatan 1 : Konsep Polymorphism.....	34
4.3.2 Kegiatan 2 : Polymorphism pada OOP	34
4.3.3 Kegiatan 3 : Polymorphism pada OOP	35
4.3.4 Kegiatan 4 : Polymorphism pada OOP	36
4.4 Tugas.....	37
BAB 5 Inheritance	38
5.1 Tujuan	38
5.2 Pengantar	39

5.2.1	Konsep Inheritance	39
5.2.2	Superclass dan Subclass	40
5.2.3	Inheritance Single Level.....	40
5.2.4	Inheritance Multi Level	41
5.2.5	Inheritance Hierarki.....	41
5.3	Kegiatan Praktikum.....	42
5.3.1	Kegiatan 1 : Inheritance Single Level	42
5.3.2	Kegiatan 2 : Inheritance Multi Level.....	43
5.3.3	Kegiatan 3 : Inheritance Hierarki	44
5.4	Tugas.....	45
BAB 6 Class Diagram.....		46
6.1	Tujuan	46
6.2	Pengantar	47
6.2.1	Pengertian Class Diagram	47
6.2.2	Komponen Class Diagram.....	48
6.2.3	Jenis-Jenis Hubungan dalam Class Diagram.....	49
6.2.5	Langkah Membuat Class Diagram	51
6.2.6	Penerapan Class Diagram dalam Pemodelan Sistem	52
6.3	Kegiatan Praktikum.....	53
6.3.1	Kegiatan 1 : Membuat Class Diagram.....	53
6.3.2	Studi Kasus 1 : Sistem Perpustakaan.....	54
6.3.3	Studi Kasus 2 : Memahami Hubungan antar Class	55
6.4	Tugas.....	56
BAB 7 SOLID PRINCIPLE		58
7.1	Tujuan	58
7.2	Pengantar	59
7.2.1	Pengantar SOLID	59
7.2.2	5 Prinsip SOLID	59
7.2.2.1	Single Responsibility Principle (SRP).....	59
7.2.2.2	Open/Closed Principle (OCP).....	60
7.2.2.3	Liskov Substitution Principle (LSP)	60
7.2.2.4	Interface Segregation Principle (ISP)	61
7.2.2.5	Dependency Inversion Principle (DIP).....	61
7.3	Kegiatan Praktikum.....	62
7.3.1	Kegiatan 1 : Single Responsibility Principle (SRP)	62
7.3.2	Kegiatan 2 : Open/Closed Principle (OCP)	64
7.3.3	Kegiatan 3 : Open/Closed Principle (OCP).....	65
7.4	Tugas.....	67
BAB 8 PyQt.....		70
8.1	Tujuan	70
8.2	Pengantar	71
8.2.1	Pendahuluan PyQt	71
8.2.2	Instalasi dan Setup PyQt untuk Windows	71
8.2.3	Konsep Dasar PyQT	72
8.2.4	Struktur Aplikasi PyQT	72
8.2.5	Widget dan Layout	75
8.2.5.1	Qlabel.....	75
8.2.5.2	QPushButton.....	75
8.2.5.3	QLineEdit	76
8.2.5.4	QTextEdit	76
8.2.5.5	QCheckBox.....	77
8.2.5.6	QRadioButton.....	78
8.2.5.7	QcomboBox.....	78
8.2.5.8	Qslider.....	79

8.2.5.9	QVBoxLayout.....	80
8.2.5.10	QHBoxLayout.....	81
8.2.5.11	QGridLayout.....	82
8.2.6	Signal dan Slot.....	82
8.3	Kegiatan Praktikum.....	84
8.3.1	Kegiatan 1 : Layout	84
8.3.2	Kegiatan 2 : Widget.....	86
8.3.3	Kegiatan 3 : Signal dan Slot	87
8.4	Tugas.....	89
BAB 9 Form dan Database		90
9.1	Tujuan	90
9.2	Pengantar	91
9.2.1	Form	91
9.2.2	Database	94
9.2.3	Widget dalam operasi CRUD.....	98
9.2.3.1	QTableWidget.....	98
9.3	Kegiatan Praktikum.....	102
9.3.1	Kegiatan 1 : Data dan Form.....	102
9.3.2	Kegiatan 2 : Studi Kasus Aplikasi Kontak	104
9.4	Tugas.....	109
BAB 10 PyQt Designer		110
10.1	Tujuan	110
10.2	Pengantar	111
10.2.1	Pengenalan Qt Designer	111
10.2.2	Instalasi Qt Designer.....	111
10.2.3	Membuat Form dengan Qt Designer	113
10.2.4	Mengubah UI menjadi Kode Python	114
10.3	Kegiatan Praktikum.....	115
10.3.1	Kegiatan 1 : Membuat Aplikasi Pertama	115
10.3.2	Kegiatan 2 : Signal dan Slot pada PyQt Designer	117
10.3.3	Kegiatan 3 : Memproses Input.....	122
10.4	Tugas.....	123
DAFTAR PUSTAKA		125
LAMPIRAN		126

BAB 1

Class dan Object



1.1 Tujuan

1. Dapat menjelaskan class dan object pada bahasa python.

2. Dapat mengimplementasikan class dan object pada pemrograman bahasa python

1.2 Pengantar

Pemrograman Berorientasi Object (PBO) merupakan salah satu paradigma yang diterapkan hampir di seluruh bahasa pemrograman. Konsep dasar dari PBO adalah mengumpulkan data dan fungsi yang memiliki hubungan kedalam suatu pulau informasi. Pulau ini disebut dengan object.

Jika dibandingkan dengan bahasa pemrograman prosedural, PBO akan melihat suatu masalah secara keseluruhan. Dalam bahasa pemrograman prosedural suatu masalah akan dipecahkan dengan cara memanggil prosedur yang biasa disebut dengan function. Dalam PBO, alih-alih berurusan dengan data secara langsung PBO akan memahami data mana yang akan digunakan dengan cara melakukan modeling. Untuk melakukan modeling ini ada beberapa istilah yang perlu dipahami yaitu class dan object

1.2.1 Pengenalan Class dan Object

Class merupakan sebuah blueprint dari object yang akan kita buat. Class berarti cetakannya sedangkan object (instance) adalah hasil dari cetakan tersebut. Untuk memahami secara lebih jelas perhatikan tabel berikut ini.

Tabel 8.1 Contoh Class dan Object

Class	Object/Instance
Car	Honda Jazz, Toyota Avanza, Suzuki Jimny
Cat	Persia, Siam, Kucing Bengal
Coffe	Americano, Capucino, Late
Dog	Labrador, Husky, Bulldog, Doberman

Untuk membuat sebuah class kita dapat menggunakan kode berikut ini. Nama sebuah kelas harus diawali dengan huruf kapital.

```
1. class NamaKelas:  
2.     # atribut atau metode yang digunakan di class ini
```

Contoh dalam pembuatan class sebuah Dog

```
1. class Dog:  
2.     def __init__(self, nama, umur):  
3.         self.nama = nama  
4.         self.umur = umur  
5.  
6.     def duduk(self):  
7.         print(f"{self.nama} sekarang duduk")  
8.  
9.     def berdiri(self):  
10.        print(f"{self.nama} sekarang berdiri")  
11.
```

Kode diatas akan dijelaskan pada sub bab dibawah ini

1.2.2 Method __init__

Fungsi yang berada dalam sebuah class dinamakan dengan method. Semua aturan fungsi yang sudah kita pelajari di bab sebelumnya berlaku juga pada pembuatan method. Jika kita lihat pada baris ke-2 terdapat method `__init__`, method ini merupakan spesial method yang secara otomatis akan berjalan setiap kali object-instance dari class Dog dibuat. Penulisan method ini diawali dengan dua kali underscore (`_`) dan diakhiri juga dengan dua kali underscore (`_`). Tanpa penulisan dua kali underscore di awal dan di akhir ini maka python tidak akan menjalankan method ini secara otomatis.

1.2.3 Parameter self

Pada method `__init__` terdapat tiga parameter yaitu `self`, `nama` dan `umur`. Parameter `self` merupakan sebuah parameter yang harus ada didalam pembuatan method, dan harus ditulis di awal sebelum parameter lainnya. Jika kita lihat pada method `duduk` dan `berdiri`, parameter `self` harus ditulis walaupun tidak ada parameter lain di dalam method tersebut.

Parameter self ini berfungsi untuk mendapatkan akses secara internal terhadap atribut atau method didalam sebuah class saat kita membuat objek/instance.

1.2.4 Membuat Objek/Instance dari class Dog

Jika sebuah class adalah blueprint maka object adalah hasil cetakannya. Mari kita buat object dari class Dog

```
1. class Dog:  
2.     def __init__(self, nama, umur):  
3.         self.nama = nama  
4.         self.umur = umur  
5.  
6.     def duduk(self):  
7.         print(f"{self.nama} sekarang duduk")  
8.  
9.     def berdiri(self):  
10.        print(f"{self.nama} sekarang berdiri")  
11.  
12.my_dog = Dog("Labrador",6)  
13.  
14.print(f"anjingku bernama {my_dog.nama}")  
15.print(f"anjingku berumur {my_dog.umur} tahun ")
```

Baris ke 12 merupakan cara pembuatan object my_dog berdasarkan class Dog. Pembuatan object harus menggunakan aturan lowercase. Pada pembuatan object my_dog kita mengirim dua variabel yaitu Labrador dan 6. Parameter ini disesuaikan dengan kebutuhan method __init__ pada class Dog. Ingat parameter self akan diproses secara otomatis sehingga kita hanya perlu mengirim variabel untuk parameter nama dan umur. Jika baris ke 12 ini dijalankan maka python akan membuat sebuah object bernama my_dog dengan nama = Labrador dan umur = 6

1.2.5 Mengakses Attribute

Untuk mengakses attribute pada sebuah objek/instance kita menggunakan bantuan titik (.) perhatikan baris ke 14 dan 15 dari kode diatas. Saat akan mengakses attribute nama dari object my_dog penulisannya menjadi

```
1. {my_dog.nama}
```

Dengan bantuan titik(.) disini maka python akan mencari objek dengan nama my_dog kemudian mencari atribut nama di object my_dog tersebut. Jika baris 14 dan 15 dijalankan maka akan menghasilkan output seperti berikut

anjingku bernama Wili

anjingku berumur 6 tahun

1.2.6 Memanggil Method

Jika kita sudah membuat object dari suatu class maka kita dapat menggunakan method yang ada pada class tersebut. Perhatikan kode dibawah ini

```
1. class Dog:  
2.     def __init__(self, nama, umur):  
3.         self.nama = nama  
4.         self.umur = umur  
5.  
6.     def duduk(self):  
7.         print(f"{self.nama} sekarang duduk")  
8.  
9.     def berdiri(self):  
10.        print(f"{self.nama} sekarang berdiri")  
11.  
12.my_dog = Dog("Wili",6)  
13.  
14.print(f"anjingku bernama {my_dog.nama}")  
15.print(f"anjingku berumur {my_dog.umur} tahun")  
16.  
17.my_dog.duduk()  
18.my_dog.berdiri()
```

Perhatikan baris ke 17 dan 18, hampir mirip seperti pengaksesan atribut, pemanggilan method juga menggunakan bantuan titik(.). Kita tulis nama object terlebih dahulu, beri tanda titik, diakhiri dengan nama method yang akan dipanggil. Jika kode tersebut dijalankan maka akan menghasilkan output seperti berikut :

anjingku bernama Wili

anjingku berumur 6 tahun

Wili sekarang duduk

Wili sekarang berdiri

1.2.7 Memanggil Method dengan parameter

Perhatikan kembali contoh kode pada class Dog berikut ini

```
my_dog = Dog("Wili", 6)
```

merupakan cara pembuatan objek my_dog dari Class Dog dengan dua paramater yang terdiri dari string dan integer, yaitu Wili dan 6. Aturan pembuatan objek dengan paramater seperti ini memiliki aturan yang sama dengan pemanggilan function pada pemrograman prosedural

2.1 Kegiatan Praktikum

2.1.1 Kegiatan 1 : Class dengan satu instance

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

```
1. class LightSwitch():
2.     def __init__(self):
3.         self.switchIsOn = False
4.
5.     def turnOn(self):
6.         # turn the switch on
7.         self.switchIsOn = True
8.
9.     def turnOff(self):
10.        # turn the switch off
11.        self.switchIsOn = False
12.
13.
14.    def show(self): # added for testing #
15.        print(self.switchIsOn)
16.
17.# Main code
18.oLightSwitch = LightSwitch()
19.
20.# Calls to methods
21.oLightSwitch.show()
22.oLightSwitch.turnOn()
23.oLightSwitch.show()
```

```
24.oLightSwitch.turnOff()
25.oLightSwitch.show()
26.oLightSwitch.turnOn()
27.oLightSwitch.show()
```

- 3 Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.3.2 Kegiatan 2 : Class dengan dua instance

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

```
1. class LightSwitch():
2.     def __init__(self):
3.         self.switchIsOn = False
4.
5.     def turnOn(self):
6.         # turn the switch on
7.         self.switchIsOn = True
8.
9.     def turnOff(self):
10.        # turn the switch off
11.        self.switchIsOn = False
12.
13.    def show(self): # added for testing #
14.        print(self.switchIsOn)
15.
16.
17.# Main code
18.oLightSwitch1 = LightSwitch()
19.oLightSwitch2 = LightSwitch()
20.
21.# Test code
22.oLightSwitch1.show()
23.oLightSwitch2.show()
24.oLightSwitch1.turnOn() # Turn switch 1 on
25.# Switch 2 should be off at start, but this makes it cleare
   r
26.oLightSwitch2.turnOff()
27.oLightSwitch1.show()
28.oLightSwitch2.show()
```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.3.3 Kegiatan 3 : Mendalami Class

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

```
1. class RemoteTv():
2.
3.     def __init__(self):
4.         self.switchIsOn = False
5.         self.brightness = 0
6.
7.     def turnOn(self):
8.         self.switchIsOn = True
9.
10.    def turnOff(self):
11.        self.switchIsOn = False
12.
13.    def raiseLevel(self):
14.        if self.brightness < 10:
15.            self.brightness = self.brightness + 1
16.
17.    def lowerLevel(self):
18.        if self.brightness > 0:
19.            self.brightness = self.brightness - 1
20.
21.    # Extra method for debugging
22.    def show(self):
23.        print('Switch is on ?', self.switchIsOn)
24.        print('Brightness is:', self.brightness)
25.
26. # Main code
27. remoteSatu = RemoteTv()
28.
29. # Turn switch on, and raise the Level 5 times
30. remoteSatu.turnOn()
31. remoteSatu.raiseLevel()
32. remoteSatu.raiseLevel()
33. remoteSatu.raiseLevel()
34. remoteSatu.raiseLevel()
35. remoteSatu.raiseLevel()
36. remoteSatu.show()
37.
```

```
38.# Lower the Level 2 times, and turn switch off  
39.remoteSatu.lowerLevel()  
40.remoteSatu.lowerLevel()  
41.remoteSatu.turnOff()  
42.remoteSatu.show()
```

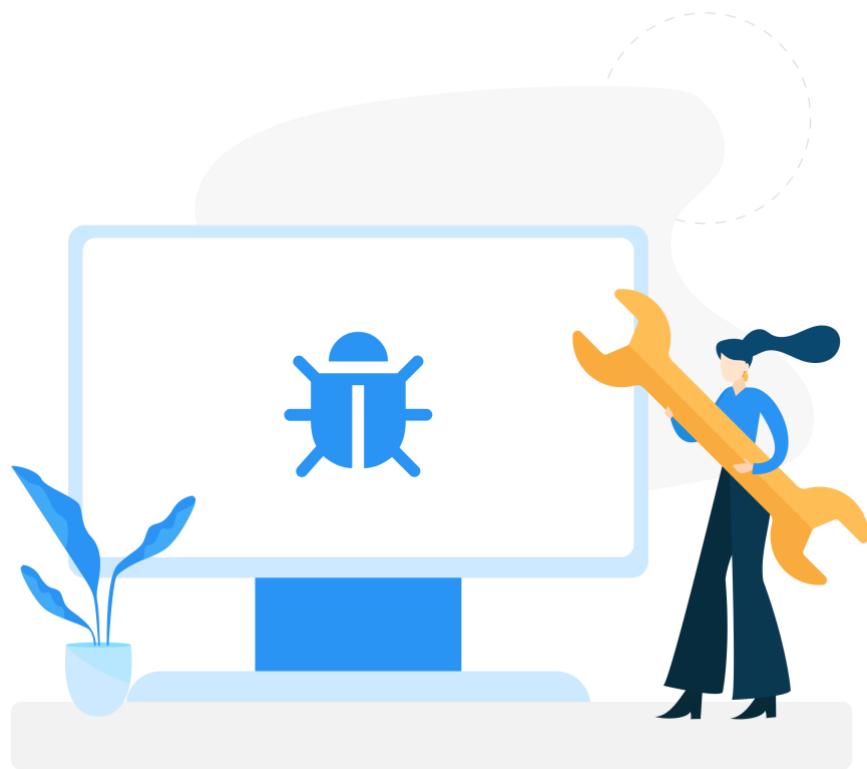
2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.4 Tugas

1. Dengan menggunakan source code dari Kegiatan 3, buatlah method **volumeUp** yang berfungsi menaikkan level volume dan **volumeDown** yang berfungsi menurunkan level volume

BAB 2

Exception Handling



2.1 Tujuan

1. Mahasiswa dapat memahami tipe error pada kode python
2. Mahasiswa dapat menangani permasalahan error pada kode python

2.2 Pengantar

2.2.1 Error dan Exception

Pada saat kita bekerja dalam pembuatan kode, ada kemungkinan-kemungkinan yang menyebabkan kode kita tersebut mengalami kesalahan sehingga program tidak dapat berjalan. Secara umum kesalahan pada penulisan kode Python terbagi menjadi dua:

1. **Syntax Error**, kesalahan pada penulisan kode. Disebut juga dengan Parsing Errors
2. **Exception**, pengecualian atau disebut dengan Runtime Errors

Penanganan error ini diperlukan pada program karena dapat membantu meningkatkan keandalan dan keamanan program, serta memberikan pengalaman yang lebih baik bagi pengguna. Tanpa adanya penanganan error, sebuah program dapat mengalami crash atau terhenti secara tiba-tiba ketika terjadi kesalahan atau kegagalan dalam proses program. Hal ini dapat mengakibatkan kerugian data atau informasi yang telah diolah oleh program, serta mengakibatkan ketidaknyamanan bagi pengguna.

Dalam pemrograman, penanganan error juga diperlukan untuk menguji kode program dan memastikan bahwa program yang dibuat dapat berjalan dengan baik dalam berbagai kondisi yang mungkin terjadi. Dengan melakukan penanganan error secara baik dan efektif, maka kualitas program yang dibuat dapat ditingkatkan dan meminimalkan risiko kesalahan atau kegagalan dalam program.

2.2.2 Syntax Error

Kesalahan sintaks atau kesalahan pada penulisan kode merupakan hal yang sangat wajar Ketika belajar suatu Bahasa pemrograman. Kesalahan ini bisa terjadi karena kesalahan indentasi, kesalahan penulisan atau kekurangan karakter pada suatu fungsi

```
1. if 1 > 2 :  
2. print("1 lebih besar dari 2")
```

kode diatas akan memunculkan pesan error berikut ini

```
File "main.py", line 2
    print("1 lebih besar dari 2")
    ^
IndentationError: expected an indented block
```

Dikarenakan menggunakan blok if maka baris ke dua harus memiliki indentasi, sehingga penulisan kode yang benar adalah sebagai berikut

```
1. if 1 > 2 :
2.     print("1 lebih besar dari 2")
```

2.2.3 Exception

Error yang kemungkinan bisa terjadi walaupun kita sudah memastikan bahwa tidak ada penulisan kode yang salah adalah kesalahan pada saat program dijalankan. Kesalahan ini disebut **runtime errors** atau **exception**. Tipe kesalahan ini harus kita perhatikan dan tangani agar program tidak macet di tengah jalan.

```
1. total = 0
2. def sum(nominal):
3.     global total
4.     total += float(nominal)
5.
6. price = input("Masukkan Harga: ")
7. sum(price)
8. print("Total Harga : {0}".format(total))
```

Kode diatas akan normal Ketika dijalankan, namun Ketika kita menginputkan string ke dalam program tersebut akan muncul error seperti berikut

```
Masukkan Harga: seribu
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    sum(price)
  File "main.py", line 4, in sum
    total += float(nominal)
ValueError: could not convert string to float: 'seribu'
```

Exception yang terjadi adalah **ValueError** karena program tersebut hanya akan memproses inputan angka bukan string. Sehingga python akan memunculkan exception dan program akan berhenti di tengah jalan. Secara default python akan memberitahukan kepada kita nama tipe exception pada pesan error sehingga kita dapat melakukan penanganan error sesuai exception yang terjadi

2.2.4 Exception Handling

Exception handling atau penangan error pada python dapat menggunakan sintaks try dan except sesuai dengan format berikut ini

```
1. try:
2.     # baris kode yang ingin dijalankan
3. except:
4.     # baris kode yang ingin dijalankan
5.     # jika TERJADI Exception
6. else:
7.     # baris kode yang akan dijalankan
8.     # jika TIDAK TERJADI Exception
9. finally:
10.    # baris kode yang akan selalu dijalankan
11.    # di akhir baik ketika TERJADI atau
12.    # TIDAK TERJADI Exception
13.    # baris ini akan selalu di eksekusi
```

Sesuai dengan format diatas maka kita dapat menambahkan try dan except kedalam kode sehingga kode kita menjadi seperti berikut

```
1. total = 0
2. def sum(nominal):
3.     global total
4.     try:
5.         total += float(nominal)
6.     except:
7.         print("Harga harus berupa angka")
8.
9. price = input("Masukkan Harga: ")
10.sum(price)
11.print("Total Harga : {0}".format(total))
```

Perhatikan baris ke 4 dan 6, kita menambahkan sintaks try dan except untuk membungkus baris ke 5. Baris ke 5 inilah program utama kita yaitu untuk menghitung inputan user. Sehingga Ketika terjadi error pada baris ke 5 ini maka program akan memunculkan pesan “**Harga harus berupa angka**”.

Syntax except ini akan menghandle semua exception yang terjadi. Kekurangannya adalah semua error akan dianggap sama dan pesan error yang muncul juga akan seragam. Akan lebih baik jika pesan error yang muncul menyesuaikan tipe exception yang terjadi.

Berikut contoh kode untuk menghitung pembagian sederhana dengan tipe exception **ZeroDivisionError** dimana pesan error akan muncul jika user menginputkan pembilang dengan angka 0.

```
1. def division(a, b):
2.     try:
3.         print("{0} / {1} = {2}".format(a, b, a / b))
4.     except ZeroDivisionError:
5.         print("Tidak bisa membagi angka dengan 0")
6.
7. division(10, "2")
```

2.2.5 Tipe Exception Pada Python

Berikut adalah beberapa jenis exception handling pada Python:

1. **Exception** : Exception paling umum yang digunakan untuk menangani kesalahan yang tidak spesifik. Jika kita tidak yakin dengan jenis exception yang akan terjadi, maka kita bisa menggunakan Exception sebagai penanganan kesalahan yang umum.
2. **ValueError** : Exception yang digunakan ketika terjadi kesalahan pada tipe data yang dimasukkan pada program. Misalnya, ketika kita mencoba mengkonversi string menjadi bilangan bulat (integer), tetapi string tersebut tidak valid sebagai bilangan bulat.
3. **TypeError** : Exception yang digunakan ketika kita mencoba menggunakan tipe data yang tidak sesuai untuk melakukan operasi tertentu. Misalnya, ketika kita mencoba menggabungkan (concatenate) string dan integer.
4. **IndexError** : Exception yang digunakan ketika kita mencoba mengakses indeks yang tidak ada pada suatu list atau array.
5. **KeyError** : Exception yang digunakan ketika kita mencoba mengakses kunci yang tidak ada pada suatu dictionary.
6. **FileNotFoundException** : Exception yang digunakan ketika kita mencoba membuka atau mengakses file yang tidak ditemukan atau tidak ada.
7. **ZeroDivisionError** : Exception yang digunakan ketika kita mencoba melakukan pembagian dengan angka nol.
8. **AssertionError** : exception yang digunakan ketika sebuah pernyataan asumsi (assertion) tidak terpenuhi. Misalnya, ketika kita membuat sebuah asumsi bahwa sebuah variabel akan selalu memiliki nilai positif, tetapi ternyata nilainya negatif.
9. **KeyboardInterrupt** : exception yang digunakan ketika kita ingin menghentikan program secara paksa (biasanya dengan menekan tombol Ctrl + C pada terminal).
10. **ImportError** : exception yang digunakan ketika ada kesalahan saat melakukan import module atau package.

2.3 Kegiatan Praktikum

2.3.1 Kegiatan 1 : Exception Handler pada Python

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

```
1. try:  
2.     # blok kode yang mungkin menimbulkan exception  
3.     num1 = int(input("Masukkan angka pertama: "))  
4.     num2 = int(input("Masukkan angka kedua: "))  
5.     hasil = num1 / num2  
6.     print(f"Hasil pembagian: {hasil}")  
7. except ValueError:  
8.     # exception handler untuk nilai yang tidak valid  
9.     print("Input harus berupa angka!")  
10. except ZeroDivisionError:  
11.     # exception handler untuk pembagian dengan nol  
12.     print("Tidak bisa melakukan pembagian dengan nol!")  
13. except Exception as e:  
14.     # exception handler umum  
15.     print(f"Terjadi kesalahan: {e}")  
16. finally:  
17.     # blok kode yang akan selalu dijalankan  
18.     print("Program selesai.")
```

2. Amati Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.3.2 Kegiatan 2 : Exception Handler dengan konsep OOP

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

```
1. class BankAccount:  
2.     def __init__(self, balance):  
3.         self.balance = balance  
4.  
5.     def deposit(self, amount):  
6.         try:  
7.             if amount <= 0:  
8.                 raise ValueError("Jumlah deposit harus lebih dari n  
ol!")  
9.             self.balance += amount  
10.            print(f"Deposit sebesar {amount} berhasil dilakukan. Sa  
aldo sekarang: {self.balance}")  
11.        except ValueError as e:  
12.            print(e)  
13.
```

```

14.     def withdraw(self, amount):
15.         try:
16.             if amount > self.balance:
17.                 raise ValueError("Saldo tidak mencukupi untuk melak
ukan penarikan!")
18.             self.balance -= amount
19.             print(f"Penarikan sebesar {amount} berhasil dilakukan.
Saldo sekarang: {self.balance}")
20.         except ValueError as e:
21.             print(e)
22.
23.# membuat objek dari class BankAccount
24.my_account = BankAccount(1000)
25.
26.# melakukan deposit sebesar -500 (akan menghasilkan ValueError)
27.my_account.deposit(-500)
28.
29.# melakukan deposit sebesar 500 (berhasil)
30.my_account.deposit(500)
31.
32.# melakukan penarikan sebesar 1500 (akan menghasilkan ValueError)
33.my_account.withdraw(1500)
34.
35.# melakukan penarikan sebesar 500 (berhasil)
36.my_account.withdraw(500)

```

2. Amati Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.3.3 Kegiatan 3 : Raise Exception Handler

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

```

1. def divide_numbers(a, b):
2.     if b == 0:
3.         raise ValueError("Cannot divide by zero!")
4.     else:
5.         return a/b
6.
7. try:
8.     result = divide_numbers(10, 0)
9. except ValueError as e:
10.    print(e)
11.else:
12.    print("The result is:", result)

```

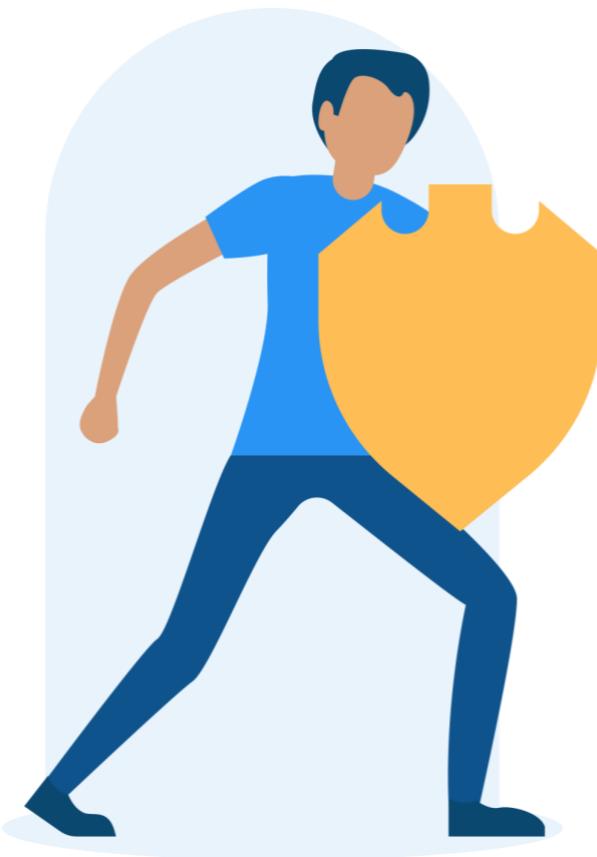
2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.4 Tugas

1. Buatlah sebuah program yang meminta input dari user berupa bilangan bulat (integer), lalu program akan mencetak hasil pangkat dua (kuadrat) dari bilangan tersebut. Namun, jika user memasukkan input yang tidak valid (bukan bilangan bulat), maka program akan menampilkan pesan error "**Input yang dimasukkan tidak valid!**" dan meminta user untuk memasukkan input yang valid.

BAB 3

Encapsulation



3.1 Tujuan

3. Dapat memahami konsep encapsulation dan abstraction pada Python
4. Dapat mengimplementasikan encapsulation dan abstraction dengan menggunakan bahasa pemrograman Python

3.2 Pengantar

3.2.1 Encapsulation

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

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

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

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

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

3.2.2 Private Variable

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

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

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

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

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

3.2.3 Getter dan Setter

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

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

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

```

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

```

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

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

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

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

3.2.4 Abstraction

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

Perbedaan antara encapsulation dan abstraction adalah sebagai berikut

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

Konsep abstraction dapat kita lihat pada kode implementasi stack pada python

```
1. #inisialisasi variabel mystack
2. myStack = []
3.
4. #insert data menggunakan fungsi append
5. myStack.append('a')
6. myStack.append('b')
7. myStack.append('c')
8.
9. #print isi variabel mystack
10.print(myStack)
11.
12.#hapus data menggunakan fungsi pop
13.myStack.pop()
14.print(myStack)
15.
16.#hapus data menggunakan fungsi pop
17.myStack.pop()
18.print(myStack)
19.
20.#hapus data menggunakan fungsi pop
21.myStack.pop()
22.print(myStack)
```

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

3.3 Kegiatan Praktikum

3.3.1 Kegiatan 1 : Implementasi Private Variabel

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

```
1. class Car:
2.     def __init__(self, make, model, year):
3.         self.__make = make
4.         self.__model = model
5.         self.__year = year
```

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

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

3.3.2 Kegiatan 2 : Implementasi Getter dan Setter

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

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

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

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

3.3.3 Kegiatan 3 : Abstraction pada OOP

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

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

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

4. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

3.4 Tugas

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

BAB 4

Polymorphism



4.1 Tujuan

5. Dapat memahami konsep polymorphism pada Pemrograman Berorientasi Objek
6. Dapat mengimplementasikan polymorphism menggunakan Bahasa pemrograman python

4.2 Pengantar

4.2.1 Polymorphism

Polymorphism adalah salah satu konsep dalam OOP (Object-Oriented Programming) di mana suatu objek dapat memiliki banyak bentuk atau tipe data. Dengan Polymorphism kita dapat menggunakan suatu objek dengan cara yang berbeda-beda, tergantung dari konteks dan kebutuhan saat penggunaan.

Konsep polymorphism dapat dijumpai dalam berbagai aspek kehidupan sehari-hari, terutama pada hal-hal yang berkaitan dengan interaksi manusia dengan lingkungan sekitarnya. Berikut adalah beberapa contoh polymorphism :

- Kendaraan

Kendaraan merupakan contoh nyata dari polymorphism. Meskipun ada banyak jenis kendaraan, seperti mobil, sepeda, motor, kereta, dan lain sebagainya, namun mereka memiliki beberapa karakteristik yang sama, seperti roda, mesin, dan kendali. Dengan begitu maka manusia dapat naik dan mengendarai berbagai jenis kendaraan, meskipun setiap jenis kendaraan memiliki cara pengoperasian yang berbeda.

- Peralatan rumah tangga

Peralatan rumah tangga seperti blender, mixer, oven, dan mesin cuci, juga dapat menjadi contoh polymorphism. Meskipun setiap peralatan rumah tangga memiliki fungsi yang berbeda-beda, namun mereka memiliki fitur yang sama, seperti tombol on/off, pengaturan kecepatan, dan lain sebagainya.

- Alat musik

Alat musik juga merupakan contoh nyata dari polymorphism. Meskipun ada banyak jenis alat musik, seperti gitar, drum, biola, piano, dan lain sebagainya, namun alat-alat tersebut memiliki beberapa karakteristik yang sama, seperti nada, ritme, dan tempo. Sehingga kita dapat bermain dan memainkan

berbagai jenis alat musik, meskipun setiap jenis alat musik memiliki teknik dan cara bermain yang berbeda.

4.2.2 Polymorphism pada Python

Konsep polymorphism pada Python dapat ditemukan di dalam library bawaan Python. Sebagai contoh, kita dapat menggunakan fungsi `len()` untuk mendapatkan panjang dari objek yang berbeda-beda, seperti string, list, tuple, set, dan dictionary. Meskipun objek-objek tersebut memiliki tipe data yang berbeda, namun kita dapat memanggil fungsi `len()` pada setiap objek tersebut dengan cara yang sama, sehingga fungsi tersebut bersifat polymorphic.

Berikut ini adalah contoh penggunaan fungsi `len()` pada beberapa tipe data yang berbeda:

```
1. my_string = "Hello, World!"  
2. my_list = [1, 2, 3, 4, 5]  
3. my_tuple = (6, 7, 8, 9, 10)  
4. my_set = {11, 12, 13, 14, 15}  
5. my_dict = {"name": "John", "age": 30, "city": "New York"}  
6.  
7. print(len(my_string))    # output: 13  
8. print(len(my_list))     # output: 5  
9. print(len(my_tuple))    # output: 5  
10.print(len(my_set))     # output: 5  
11.print(len(my_dict))    # output: 3
```

Polymorphism dalam OOP dapat diimplementasikan melalui dua cara, yaitu:

- Overloading: mengimplementasikan fungsi atau method dengan nama yang sama, tetapi dengan parameter yang berbeda. Dengan demikian, saat pemanggilan fungsi atau method, program akan memilih implementasi yang sesuai dengan parameter yang diberikan.

- Overriding: mengimplementasikan method yang sama di dalam class yang berbeda. Dengan demikian, saat pemanggilan method pada objek, program akan memilih implementasi method pada setiap classnya.

4.2.3 Overloading Polymorphism

Pada contoh kode berikut, class Calculator memiliki dua method yang bernama sama yaitu add. Namun, keduanya memiliki jumlah parameter yang berbeda. Saat pemanggilan method, program akan memilih method yang sesuai dengan jumlah parameter yang diberikan. Jika diberikan dua parameter, program akan memilih method add dengan dua parameter, dan jika diberikan tiga parameter, program akan memilih method add dengan tiga parameter. Dengan demikian, kita dapat menggunakan class Calculator dengan cara yang berbeda-beda, tergantung pada kebutuhan penggunaannya.

```

1. class Calculator:
2.     def add(self, a, b):
3.         return a + b
4.
5.     def add(self, a, b, c):
6.         return a + b + c
7.
8. # membuat objek calculator
9. my_calculator = Calculator()
10.
11.# memanggil method add dengan dua parameter
12.result1 = my_calculator.add(2, 3)
13.print(result1)    # output: 5
14.
15.# memanggil method add dengan tiga parameter
16.result2 = my_calculator.add(2, 3, 4)
17.print(result2)    # output: 9

```

4.2.4 Overriding Polymorphism

Pada contoh kode berikut, class Cow memiliki method sound yang menghasilkan output "Moo". Namun, class Dog dan Cat yang memiliki method sound yang sama, namun dengan implementasi yang berbeda di masing-masing class. Ketika method sound dipanggil pada objek Dog, program akan memilih implementasi method pada class Dog, yaitu "Bark Bark". Demikian juga ketika method sound dipanggil pada objek Cat, program akan memilih implementasi method pada class Cat, yaitu "Meoooow". Dengan demikian, kita dapat menggunakan class Cow, Dog, dan Cat dengan cara yang berbeda-beda, tergantung pada kebutuhan penggunaannya..

```
1. class Cow:
2.     def __init__(self, name):
3.         self.name = name
4.
5.     def sound(self):
6.         print("Moooo")
7.
8. class Dog:
9.     def __init__(self, name):
10.        self.name = name
11.
12.    def sound(self):
13.        print("Bark Bark")
14.
15.class Cat:
16.    def __init__(self, name):
17.        self.name = name
18.
19.    def sound(self):
20.        print("Meoooow")
21.
22.my_cow = Cow("Sapi")
23.my_cow.sound()
24.my_dog = Dog("Anjing")
25.my_dog.sound()
26.my_cat = Cat("Kucing")
27.my_cat.sound()
```

4.3 Kegiatan Praktikum

4.3.1 Kegiatan 1 : Konsep Polymorphism

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

```
1. # contoh penggunaan duck typing
2. def print_all(items):
3.     for item in items:
4.         print(item)
5.
6. # list
7. my_list = [1, 2, 3]
8.
9. # tuple
10.my_tuple = (4, 5, 6)
11.
12.# string
13.my_string = "Hello, world!"
14.
15.# memanggil fungsi print_all dengan parameter my_list, my_tuple, dan my_string
16.print_all(my_list)    # output: 1 2 3
17.print_all(my_tuple)  # output: 4 5 6
18.print_all(my_string) # output: H e l l o ,   w o r l d !
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.3.2 Kegiatan 2 : Polymorphism pada OOP

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

```
1. class India():
2.     def capital(self):
3.         print("New Delhi is the capital of India.")
4.
5.     def language(self):
6.         print("Hindi is the most widely spoken language of India.")
7.
8.     def type(self):
9.         print("India is a developing country.")
10.
```

```
11. class USA():
12.     def capital(self):
13.         print("Washington, D.C. is the capital of USA.")
14.
15.     def language(self):
16.         print("English is the primary language of USA.")
17.
18.     def type(self):
19.         print("USA is a developed country.")
20.
21. obj_ind = India()
22. obj_usa = USA()
23. for country in (obj_ind, obj_usa):
24.     country.capital()
25.     country.language()
26.     country
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.3.3 Kegiatan 3 : Polymorphism pada OOP

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

```
1. class Shape:
2.     def area(self):
3.         pass
4.
5. class Rectangle(Shape):
6.     def __init__(self, length, width):
7.         self.length = length
8.         self.width = width
9.
10.    def area(self):
11.        return self.length * self.width
12.
13. class Circle(Shape):
14.     def __init__(self, radius):
15.         self.radius = radius
16.
17.     def area(self):
18.         return 3.14 * self.radius * self.radius
19.
20. # membuat objek rectangle
```

```

21.my_rectangle = Rectangle(5, 6)
22.
23.# memanggil method area pada objek rectangle
24.result1 = my_rectangle.area()
25.print(result1) # output: 30
26.
27.# membuat objek circle
28.my_circle = Circle(7)
29.
30.# memanggil method area pada objek circle
31.result2 = my_circle.area()
32.print(result2) # output: 153.86

```

6. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.3.4 Kegiatan 4 : Polymorphism pada OOP

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

```

1. class Dog():
2.     def __init__(self, name):
3.         self.name = name
4.
5.     def speak(self):
6.         print(self.name, 'says bark, bark, bark!')
7.
8. class Cat():
9.     def __init__(self, name):
10.        self.name = name
11.
12.    def speak(self):
13.        print(self.name, 'says meeeoooow')
14.
15.class Bird():
16.    def __init__(self, name):
17.        self.name = name
18.
19.    def speak(self):
20.        print(self.name, 'says tweet')
21.
22.oDog1 = Dog('Rover')
23.oDog2 = Dog('Fido')
24.oCat1 = Cat('Fluffy')
25.oCat2 = Cat('Spike')

```

```
26.oBird = Bird('Big Bird')
27.
28.petsList = [oDog1, oDog2, oCat1, oCat2, oBird]
29.
30.for oPet in petsList:
31.    oPet.speak()
```

2. Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

4.4 Tugas

3. Buatlah 3 buah class yaitu Mobil, Pesawat dan Kapal, setiap class memiliki method yang sama bernama move(). Di dalam Method move() terdapat fungsi print yang memiliki output berbeda tergantung dengan class dimana method tersebut berada.

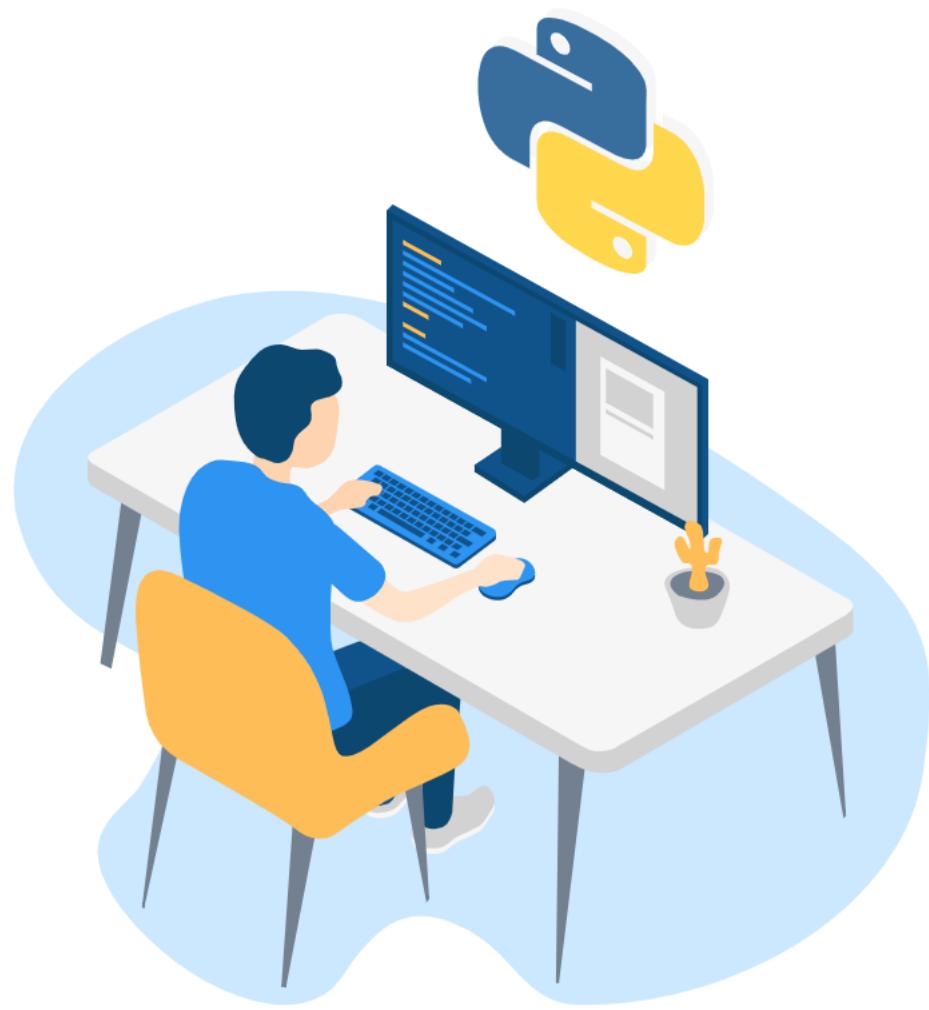
Output move() pada class Mobil = Berjalan di jalan raya

Output move() pada class Pesawat = Terbang di udara

Output move() pada class Kapal = Berlayar di laut

BAB 5

Inheritance



5.1 Tujuan

7. Dapat memahami konsep inheritance.
8. Dapat mengimplementasikan konsep inheritance menggunakan Bahasa pemrograman python.

5.2 Pengantar

5.2.1 Konsep Inheritance

Konsep inheritance pada OOP (Object-Oriented Programming) adalah sebuah konsep di mana sebuah kelas baru (subclass) dibuat dengan mewarisi sifat atau perilaku dari kelas yang sudah ada (superclass). Kelas yang mewarisi sifat dari superclass disebut sebagai subclass atau turunan, sedangkan kelas yang memberikan sifat atau perilaku disebut sebagai superclass atau induk.

Dengan menggunakan konsep inheritance, kita dapat membuat sebuah kelas baru dengan mewarisi sifat dan perilaku dari kelas yang sudah ada, sehingga mempermudah pembuatan program dan meningkatkan fleksibilitas dan modularitas pada program. Selain itu, kita juga dapat menghindari duplikasi kode dan membuat perubahan pada superclass secara otomatis diterapkan pada semua subclass yang mewarisinya.

Contoh inheritance dalam kehidupan sehari-hari adalah warisan atau turunan yang diterima oleh anak dari orang tua atau keluarga lainnya. Konsep inheritance dalam OOP mirip dengan konsep warisan dalam kehidupan nyata, dimana sifat atau ciri-ciri tertentu dapat diwariskan dari generasi sebelumnya ke generasi selanjutnya.

Sebagai contoh, jika seorang ayah memiliki bisnis restoran, maka anaknya dapat mewarisi bisnis tersebut dan melanjutkan usaha yang telah dimulai oleh ayahnya. Anak tersebut dapat mengembangkan bisnis restoran dengan menambahkan menu baru atau melakukan perubahan yang diperlukan, tetapi tetap mempertahankan sifat atau ciri khas yang dimiliki oleh bisnis restoran yang diwarisi dari ayahnya.

Konsep inheritance juga dapat diterapkan pada ilmu biologi, dimana sifat-sifat atau ciri-ciri biologis dapat diwariskan dari generasi sebelumnya ke generasi selanjutnya. Misalnya, sifat tertentu seperti warna mata atau tinggi badan dapat diwariskan dari orang tua ke anaknya. Seorang anak juga dapat mewarisi kecenderungan terhadap penyakit tertentu dari orang tua atau kakek neneknya.

5.2.2 Superclass dan Subclass

Perbedaan antara subclass dan superclass pada konsep inheritance di OOP adalah:

- Superclass adalah kelas yang memberikan sifat atau perilaku yang dapat diwarisi oleh subclass, sedangkan subclass adalah kelas yang mewarisi sifat atau perilaku dari superclass.
- Superclass adalah kelas yang lebih umum atau generik, sedangkan subclass adalah kelas yang lebih khusus atau spesifik.
- Superclass bisa memiliki beberapa subclass yang berbeda, sedangkan subclass hanya memiliki satu superclass.
- Subclass dapat menambahkan atribut atau method yang unik untuk dirinya sendiri, sementara superclass tidak dapat mempengaruhi subclass dengan cara apapun.

5.2.3 Inheritance Single Level

Single level inheritance adalah ketika sebuah kelas turunan hanya memiliki satu kelas induk atau superclass. Dalam single level inheritance, subclass mewarisi semua metode dan properti dari superclassnya.

```
1. class Manusia:  
2.     def __init__(self, nama, umur):  
3.         self.nama = nama  
4.         self.umur = umur  
5.  
6.     def info(self):  
7.         print(f"Nama: {self.nama}")  
8.         print(f"Umur: {self.umur}")  
9.  
10. class Pelajar(Manusia):  
11.     def __init__(self, nama, umur, kelas):  
12.         super().__init__(nama, umur)  
13.         self.kelas = kelas  
14.  
15.     def info(self):  
16.         super().info()  
17.         print(f"Kelas: {self.kelas}")
```

5.2.4 Inheritance Multi Level

Multi level inheritance adalah ketika sebuah kelas turunan memiliki lebih dari satu kelas induk atau superclass. Dalam multi level inheritance, sebuah subclass mewarisi semua metode dan properti dari superclassnya, termasuk dari kelas induk.

```
1. class Kendaraan:
2.     def __init__(self, jenis):
3.         self.jenis = jenis
4.
5. class Sepeda(Kendaraan):
6.     def __init__(self, jenis, warna):
7.         super().__init__(jenis)
8.         self.warna = warna
9.
10. class SepedaMotor(Sepeda):
11.     def __init__(self, jenis, warna, merk):
12.         super().__init__(jenis, warna)
13.         self.merk = merk
```

5.2.5 Inheritance Hierarki

Hierarki inheritance adalah ketika beberapa kelas turunan berasal dari satu kelas induk atau superclass yang sama. Dalam hierarki inheritance, kelas-kelas turunan memiliki metode dan properti yang sama dari superclassnya.

```
1. class Hewan:
2.     def __init__(self, jenis):
3.         self.jenis = jenis
4.
5. class Kucing(Hewan):
6.     def __init__(self, jenis, warna):
7.         super().__init__(jenis)
8.         self.warna = warna
9.
10. class Anjing(Hewan):
11.     def __init__(self, jenis, ras):
12.         super().__init__(jenis)
13.         self.ras = ras
14.
15. class Singa(Hewan):
16.     def __init__(self, jenis, habitat):
17.         super().__init__(jenis)
```

```
18.         self.habitat = habitat
```

5.3 Kegiatan Praktikum

5.3.1 Kegiatan 1 : Inheritance Single Level

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

```
1. class Person:  
2.     def __init__(self, name, age):  
3.         self.name = name  
4.         self.age = age  
5.  
6.     def introduce(self):  
7.         print(f"My name is {self.name} and I am {self.age} years ol  
d.")  
8.  
9. class Student(Person):  
10.    def __init__(self, name, age, student_id):  
11.        super().__init__(name, age)  
12.        self.student_id = student_id  
13.  
14.    def introduce(self):  
15.        super().introduce()  
16.        print(f"My student ID is {self.student_id}.")  
17.  
18.person1 = Person("John", 30)  
19.person1.introduce()  
20.  
21.student1 = Student("Mary", 20, "12345")  
22.student1.introduce()
```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

5.3.2 Kegiatan 2 : Inheritance Multi Level

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

```
1. class Kendaraan:  
2.     def __init__(self, jenis):  
3.         self.jenis = jenis  
4.  
5. class Sepeda(Kendaraan):  
6.     def __init__(self, jenis, warna):  
7.         super().__init__(jenis)  
8.         self.warna = warna  
9.  
10. class SepedaMotor(Sepeda):  
11.     def __init__(self, jenis, warna, merk):  
12.         super().__init__(jenis, warna)  
13.         self.merk = merk  
14.  
15. class Mobil(SepedaMotor):  
16.     def __init__(self, jenis, warna, merk, tahun):  
17.         super().__init__(jenis, warna, merk)  
18.         self.tahun = tahun  
19.  
20.     def info(self):  
21.         print(f"Jenis: {self.jenis}")  
22.         print(f"Warna: {self.warna}")  
23.         print(f"Merl: {self.merk}")  
24.         print(f"Tahun: {self.tahun}")  
25.  
26. mobil1 = Mobil("Sedan", "Merah", "Toyota", 2022)  
27. mobil1.info()
```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

5.3.3 Kegiatan 3 : Inheritance Hierarki

- Buat sebuah file program baru kemudian tulis kode program berikut ini

```
1. class Hewan:  
2.     def __init__(self, nama, jenis):  
3.         self.nama = nama  
4.         self.jenis = jenis  
5.  
6. class Mamalia(Hewan):  
7.     def __init__(self, nama, jenis, mamalia_bertelur):  
8.         super().__init__(nama, jenis)  
9.         self.mamalia_bertelur = mamalia_bertelur  
10.  
11.class Unggas(Hewan):  
12.    def __init__(self, nama, jenis, unggas_terbang):  
13.        super().__init__(nama, jenis)  
14.        self.unggas_terbang = unggas_terbang  
15.  
16.class Gajah(Mamalia):  
17.    def __init__(self, nama, jenis, mamalia_bertelur, berat_badan):  
18.        super().__init__(nama, jenis, mamalia_bertelur)  
19.        self.berat_badan = berat_badan  
20.  
21.class Ayam(Unggas):  
22.    def __init__(self, nama, jenis, unggas_terbang, jumlah_telur):  
23.        super().__init__(nama, jenis, unggas_terbang)  
24.        self.jumlah_telur = jumlah_telur  
25.  
26.    def info(self):  
27.        print(f"Nama: {self.nama}")  
28.        print(f"Jenis: {self.jenis}")  
29.        print(f"Unggas Terbang: {self.unggas_terbang}")  
30.        print(f"Jumlah Telur: {self.jumlah_telur}")  
31.  
32.ayam1 = Ayam("Ayam Serama", "Ayam", True, 20)  
33.ayam1.info()  
34.  
35.gajah1 = Gajah("Gajah Sumatera", "Gajah", False, 4000)  
36.print(f"Nama: {gajah1.nama}")  
37.print(f"Jenis: {gajah1.jenis}")  
38.print(f"Mamalia Bertelur: {gajah1.mamalia_bertelur}")  
39.print(f"Berat Badan: {gajah1.berat_badan}")
```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

5.4 Tugas

1. Buatlah sebuah program untuk menghitung harga tiket bioskop yang menerapkan konsep inheritance. Buatlah superclass 'Tiket' dan subclass 'TiketBiasa', 'TiketVIP', dan 'TiketGold'. Setiap subclass memiliki atribut dan method berisi harga tiket yang berbeda-beda. Output yang diharapkan dari program ini adalah seperti berikut

Masukkan jenis tiket (biasa/vip/gold) : *(input dari user)*

Masukkan jumlah tiket : *(input dari user)*

Total Harga Tiket : Rp *(Output Hasil perhitungan sistem)*

BAB 6

Class Diagram



6.1 Tujuan

1. Dapat memahami konsep class diagram
2. Dapat mengimplementasikan pembuatan class diagram dalam pengembangan pemrograman berbasis objek

6.2 Pengantar

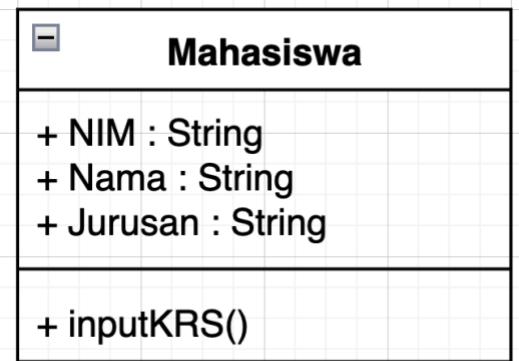
6.2.1 Pengertian Class Diagram

Class Diagram adalah salah satu jenis diagram dalam Unified Modeling Language (UML), sebuah bahasa standar untuk merancang dan mendokumentasikan sistem perangkat lunak. Class Diagram digunakan untuk memodelkan struktur sistem dengan menunjukkan class-class dalam sistem, atribut dan metode dalam class tersebut, serta hubungan antara class-class tersebut.

Class Diagram memberikan gambaran statis tentang sistem. Dalam kata lain, diagram ini menunjukkan apa yang ada di sistem dan bagaimana mereka saling terkait, tetapi tidak menunjukkan bagaimana sistem beroperasi dan bagaimana objek-objek dalam sistem berinteraksi satu sama lain. Untuk itu, kita menggunakan jenis diagram UML lainnya seperti sequence diagram atau activity diagram.

Class Diagram sangat penting dalam proses pengembangan perangkat lunak berorientasi objek. Diagram ini akan membantu perancang perangkat lunak untuk memvisualisasikan struktur kode yang mereka rancang sebelum benar-benar menulis kode tersebut. Hal ini akan memudahkan dalam membuat perubahan dan optimasi pada struktur tersebut sebelum melakukan pekerjaan yang lebih detail dan memakan waktu. Selain itu, Class Diagram juga berguna sebagai alat komunikasi antara berbagai stakeholder dalam proyek, seperti programmer, manajer proyek, dan klien.

Dalam Class Diagram, class biasanya digambarkan sebagai kotak yang dibagi menjadi tiga bagian: bagian atas berisi nama class, bagian tengah berisi atribut class, dan bagian bawah berisi metode class. Hubungan antara class digambarkan dengan berbagai jenis garis dan panah. Berikut ini contoh class diagram dengan nama class Mahasiswa, Atribut berisi NIM, nama dan Jurusan serta method dengan nama daftarKursus().



Gambar 6. 1 Class Diagram

6.2.2 Komponen Class Diagram

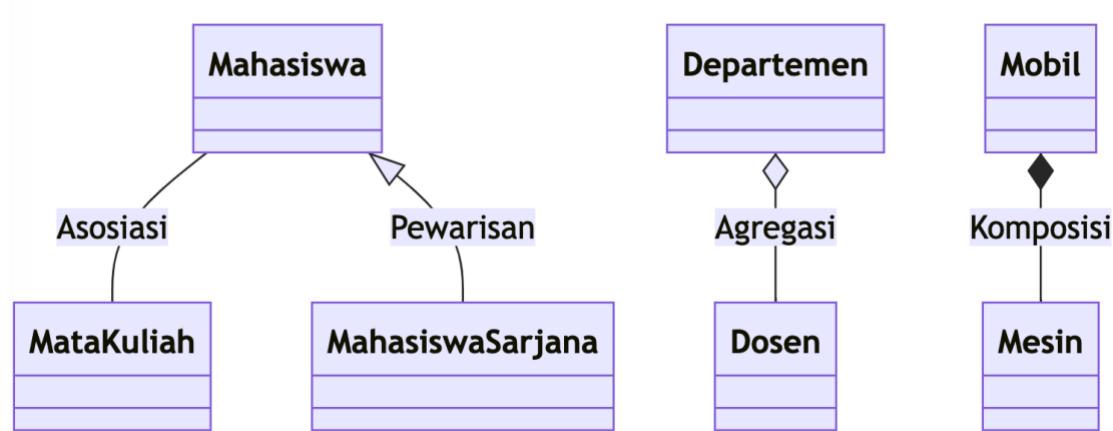
Class Diagram terdiri dari beberapa komponen utama yang membentuk struktur dasar diagram. Berikut adalah penjelasan tentang komponen-komponen tersebut:

- **Class :** Class adalah entitas utama dalam class diagram. Class digambarkan sebagai kotak dengan tiga bagian. Bagian atas berisi nama class, bagian tengah berisi atribut class, dan bagian bawah berisi metode class. Nama class biasanya ditulis dengan huruf kapital di awal, sementara atribut dan metode ditulis dengan huruf kecil di awal.
- **Atribut:** Atribut adalah variabel yang menyimpan nilai atau status objek. Atribut didefinisikan dalam bagian tengah kotak class. Setiap atribut biasanya memiliki tipe data, seperti int, string, atau boolean. Atribut biasanya ditulis dengan format: `+ namaAtribut : tipeData`.
- **Metode:** Metode adalah fungsi atau operasi yang dapat dilakukan oleh objek. Metode didefinisikan dalam bagian bawah kotak class. Setiap metode biasanya memiliki daftar parameter dan tipe pengembalian. Metode biasanya ditulis dengan format: `+ namaMetode(parameter : tipeData) : tipePengembalian`.

- Hubungan: Hubungan adalah tautan antara dua class yang menunjukkan bagaimana class tersebut berinteraksi satu sama lain. Hubungan dalam class diagram antara lain: Asosiasi, Agregasi, Komposisi dan Pewarisan

6.2.3 Jenis-Jenis Hubungan dalam Class Diagram

Dalam class diagram, hubungan antara class digambarkan menggunakan berbagai jenis garis dan simbol.



Gambar 6. 2 Jenis Hubungan dalam Class Diagram

Asosiasi: Asosiasi adalah hubungan dasar antara dua class. Asosiasi menunjukkan bahwa objek dalam satu class terkait dengan objek dalam class lain. Misalnya, class 'Mahasiswa' mungkin memiliki asosiasi dengan class 'MataKuliah', yang menunjukkan bahwa seorang mahasiswa dapat mengambil beberapa mata kuliah. Asosiasi biasanya digambarkan sebagai garis lurus antara dua class.

Agregasi: Agregasi adalah hubungan "memiliki" di mana class satu adalah bagian dari class lain, tetapi masih dapat ada secara independen. Misalnya, class 'Departemen' mungkin memiliki agregasi dengan class 'Dosen', yang menunjukkan bahwa departemen memiliki beberapa dosen, tetapi seorang dosen masih dapat ada tanpa

departemen. Agregasi biasanya digambarkan sebagai garis dengan simbol berlian di satu ujung.

Komposisi: Komposisi adalah hubungan "memiliki" yang lebih kuat di mana class satu tidak dapat ada tanpa class lain. Misalnya, class 'Mobil' mungkin memiliki komposisi dengan class 'Mesin', yang menunjukkan bahwa mobil memiliki mesin dan mesin tidak dapat ada tanpa mobil. Komposisi biasanya digambarkan sebagai garis dengan simbol berlian yang diisi di satu ujung.

Pewarisan (Inheritance): Pewarisan adalah hubungan "adalah jenis dari" di mana satu class adalah subclass dari class lain. Misalnya, class 'MahasiswaSarjana' mungkin mewarisi dari class 'Mahasiswa', yang menunjukkan bahwa MahasiswaSarjana adalah jenis khusus dari Mahasiswa. Pewarisan biasanya digambarkan sebagai garis dengan simbol panah berbentuk segitiga di satu ujung.

6.2.4 Kardinalitas

Kardinalitas dalam konteks pemrograman dan basis data merujuk pada jumlah elemen dalam satu set atau, lebih spesifik, jumlah hubungan antara entitas dalam model data. Kardinalitas digunakan untuk mendefinisikan hubungan antara entitas dalam model entitas-relasi (ER) dalam basis data. Ada beberapa jenis kardinalitas, termasuk:

1. **One-to-One (1:1):** Dalam hubungan ini, satu entitas dari set entitas A hanya dapat terkait dengan satu entitas dari set entitas B, dan sebaliknya. Misalnya, dalam sebuah perusahaan, setiap karyawan memiliki satu nomor identifikasi unik.
2. **One-to-Many (1:N) atau Many-to-One (N:1):** Dalam hubungan ini, satu entitas dari set entitas A dapat terkait dengan banyak entitas dari set entitas B. Namun, satu entitas dari set entitas B hanya dapat terkait dengan satu entitas dari set entitas A.

Misalnya, satu ibu bisa memiliki banyak anak, tetapi setiap anak hanya memiliki satu ibu.

3. **Many-to-Many (M:N):** Dalam hubungan ini, satu entitas dari set entitas A dapat terkait dengan banyak entitas dari set entitas B dan sebaliknya. Misalnya, dalam model basis data untuk sebuah toko buku, satu buku bisa dibeli oleh banyak pelanggan dan satu pelanggan bisa membeli banyak buku.

6.2.5 Langkah Membuat Class Diagram

Membuat class diagram melibatkan beberapa langkah penting. Berikut adalah langkah-langkah umum dalam proses pembuatan class diagram:

1. Identifikasi Class: Langkah pertama dalam membuat class diagram adalah mengidentifikasi class-class yang akan digambarkan dalam diagram. Class biasanya mewakili objek nyata atau konsep dalam sistem, seperti 'Mahasiswa', 'MataKuliah', atau 'Fakultas'.
2. Definisikan Atribut dan Metode: Setelah class diidentifikasi, langkah selanjutnya adalah mendefinisikan atribut dan metode untuk setiap class. Atribut adalah variabel yang menyimpan nilai atau status objek, sementara metode adalah fungsi atau operasi yang dapat dilakukan oleh objek.
3. Identifikasi Hubungan: Langkah selanjutnya adalah mengidentifikasi hubungan antara class. Hubungan bisa berupa asosiasi (hubungan dasar antara dua class), agregasi (hubungan "memiliki" di mana class satu adalah bagian dari class lain), komposisi (hubungan "memiliki" yang lebih kuat di mana class satu tidak dapat ada tanpa class lain), atau pewarisan (hubungan "adalah jenis dari" di mana satu class adalah subclass dari class lain).
4. Gambar Diagram: Setelah semua class, atribut, metode, dan hubungan diidentifikasi, langkah selanjutnya adalah menggambar diagram. Class

digambarkan sebagai kotak dengan tiga bagian, sementara hubungan digambarkan sebagai garis antara kotak.

5. Review dan Refine: Langkah terakhir adalah meninjau dan menyempurnakan diagram. Pastikan semua class, atribut, metode, dan hubungan digambarkan dengan benar dan jelas. Jika perlu, buat perubahan atau penyesuaian untuk meningkatkan kejelasan atau akurasi diagram.

6.2.6 Penerapan Class Diagram dalam Pemodelan Sistem

Class Diagram memiliki peran penting dalam pemodelan sistem, terutama dalam konteks pengembangan perangkat lunak berorientasi objek. Berikut adalah beberapa penerapan class diagram dalam pemodelan sistem:

- Perancangan Sistem: Class Diagram digunakan dalam tahap perancangan sistem untuk memvisualisasikan struktur dasar sistem yang akan dibangun. Dengan class diagram, pengembang dapat memahami class-class yang ada dalam sistem, atribut dan metode dalam class tersebut, serta hubungan antara class-class tersebut.
- Dokumentasi Sistem: Class Diagram juga berfungsi sebagai alat dokumentasi yang efektif. Dengan class diagram, struktur sistem dapat didokumentasikan dengan jelas dan sistematis. Ini memudahkan pengembang lain untuk memahami sistem, terutama saat melakukan pemeliharaan atau peningkatan sistem.
- Komunikasi antar Stakeholder: Class Diagram dapat digunakan sebagai alat komunikasi antara berbagai stakeholder dalam proyek, seperti programmer, manajer proyek, analis sistem, dan klien. Dengan class diagram, semua pihak dapat memiliki pemahaman yang sama tentang struktur sistem.
- Pengujian dan Validasi Sistem: Class Diagram juga dapat digunakan dalam tahap pengujian dan validasi sistem. Dengan membandingkan class diagram

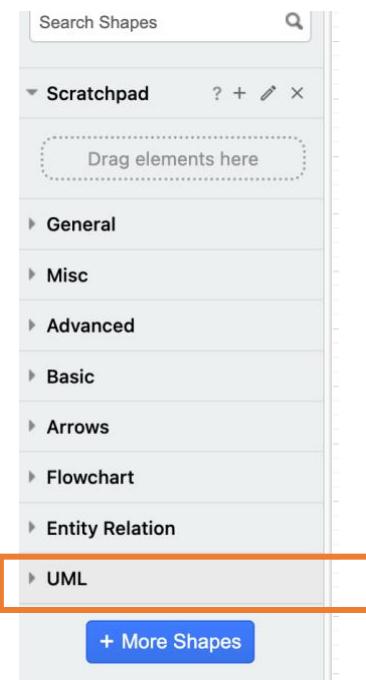
dengan implementasi sistem yang sebenarnya, pengembang dapat memastikan bahwa sistem telah dibangun sesuai dengan desain awal.

- Pendidikan dan Pelatihan: Dalam konteks pendidikan dan pelatihan, class diagram digunakan untuk mengajarkan konsep-konsep pemrograman berorientasi objek, seperti class, atribut, metode, dan pewarisan.

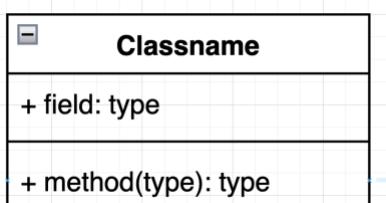
6.3 Kegiatan Praktikum

6.3.1 Kegiatan 1 : Membuat Class Diagram

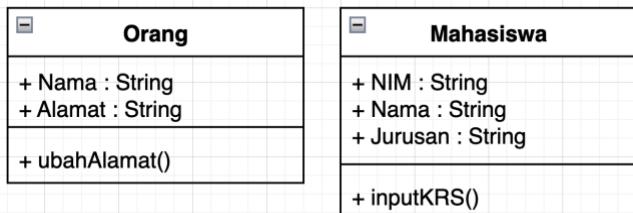
3. Kunjungi website <https://app.diagrams.net/> kemudian Pilih Create New Diagram
4. Pilih UML pada bagian Shape



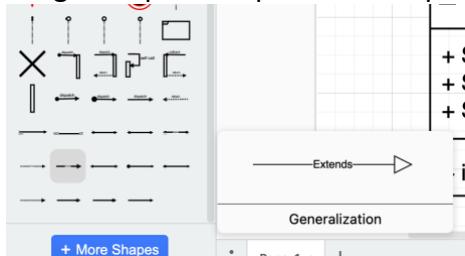
5. Pilih Class Diagram dengan cara drag n drop ke area kerja, Untuk melakukan perubahan klik 2x pada simbol class tersebut.



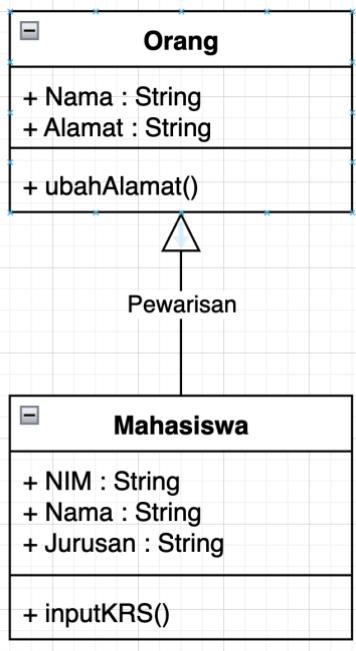
6. Buatlah 2 class seperti berikut



7. Drag n drop notasi pewarisan seperti berikut untuk 2 class tersebut

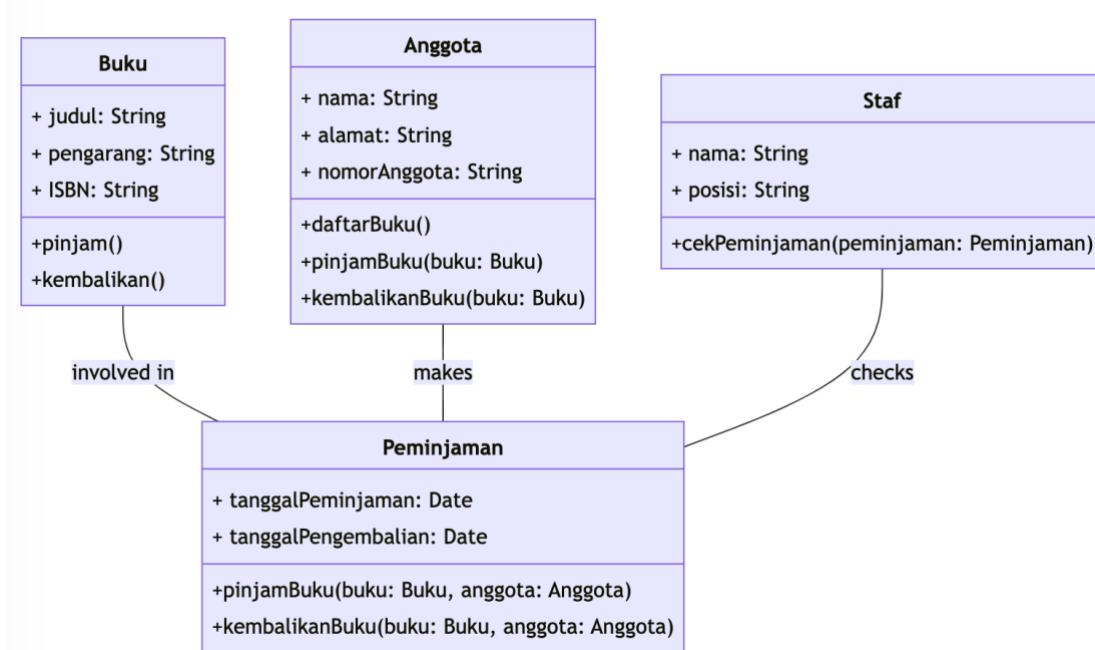


8. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini



6.3.2 Studi Kasus 1 : Sistem Perpustakaan

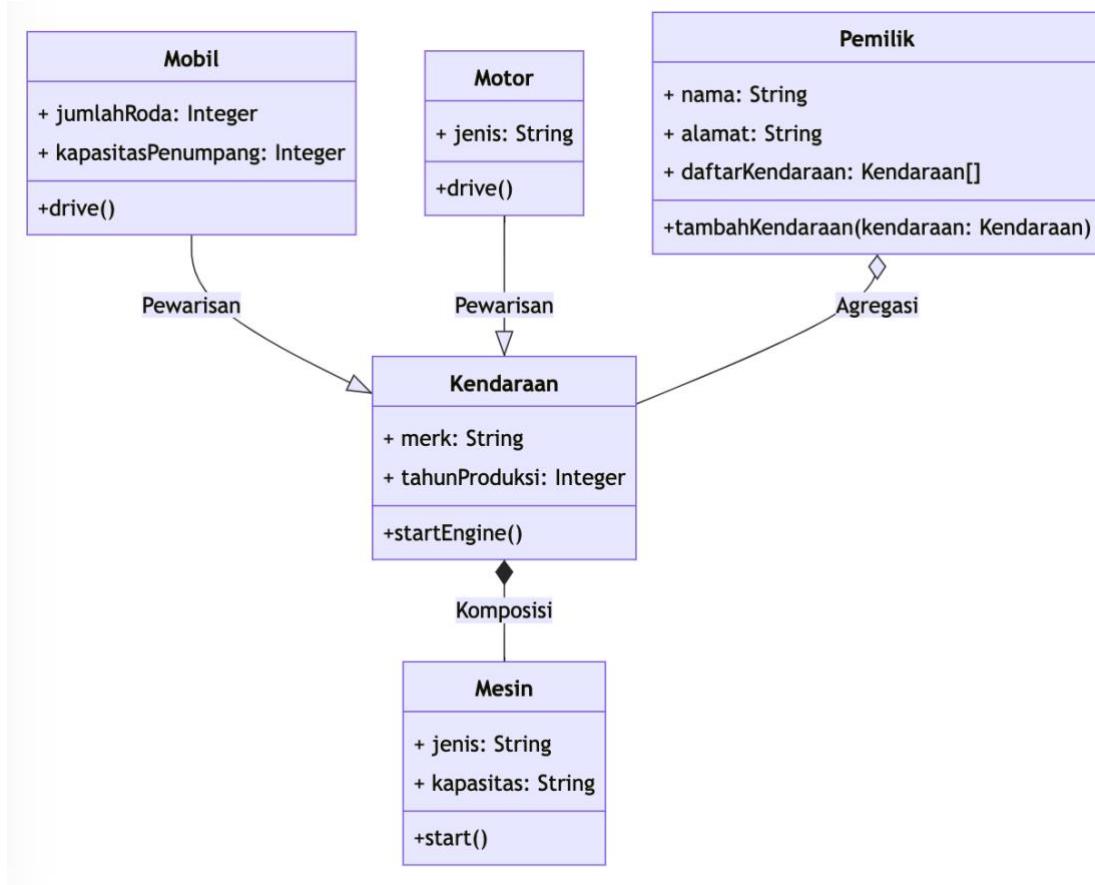
2. Buatlah Class Diagram seperti berikut ini



3. Buatlah analisa terhadap asosiasi yang terjadi pada setiap classnya

6.3.3 Studi Kasus 2 : Memahami Hubungan antar Class

1. Buatlah Class Diagram seperti berikut ini



2. Buatlah analisa terhadap asosiasi yang terjadi pada setiap classnya

6.4 Tugas

Sebuah perusahaan teknologi memiliki beberapa tim yang berbeda, termasuk tim pengembangan, tim pemasaran, dan tim dukungan pelanggan. Setiap tim memiliki sejumlah anggota dan setiap anggota memiliki peran tertentu dalam timnya. Selain itu, setiap anggota memiliki sejumlah keterampilan yang berbeda.

Buatlah class diagram yang mencakup kelas-kelas berikut:

- Perusahaan: Perusahaan memiliki nama dan alamat. Perusahaan juga memiliki beberapa tim.

- Tim: Setiap tim memiliki nama dan deskripsi. Tim juga memiliki beberapa anggota.
- Anggota: Setiap anggota memiliki nama dan email. Anggota juga memiliki peran dalam tim dan beberapa keterampilan.
- Peran: Setiap peran memiliki nama dan deskripsi.
- Keterampilan: Setiap keterampilan memiliki nama dan tingkat (misalnya, pemula, menengah, mahir).

Dalam diagram Anda, pastikan untuk menunjukkan hubungan berikut:

- **Asosiasi** antara Tim dan Anggota (setiap tim memiliki beberapa anggota dan setiap anggota adalah bagian dari satu tim).
- **Agregasi** antara Perusahaan dan Tim (setiap perusahaan memiliki beberapa tim, tetapi tim dapat ada tanpa perusahaan).
- **Komposisi** antara Anggota dan Peran (setiap anggota memiliki satu peran dan peran tidak dapat ada tanpa anggota).
- **Pewarisan** antara kelas baru "AnggotaPengembangan" dan "Anggota" (AnggotaPengembangan adalah jenis khusus dari Anggota).

Jelaskan setiap kelas, atribut, metode, dan hubungan dalam diagram Anda.

BAB 7

SOLID PRINCIPLE



7.1 Tujuan

1. Dapat memahami konsep SOLID pada Pemrograman Berorientasi Objek
2. Dapat mengimplementasikan konsep SOLID menggunakan Python

7.2 Pengantar

7.2.1 Pengantar SOLID

Konsep SOLID adalah sebuah akronim yang mencakup lima prinsip dasar dalam pemrograman berorientasi objek dan desain. SOLID berasal dari Single Responsibility Principle (SRP), Single Responsibility Principle (SRP), Single Responsibility Principle (SRP), Interface Segregation Principle (ISP) dan Interface Segregation Principle (ISP). Prinsip-prinsip ini ditujukan untuk membuat sistem perangkat lunak yang mudah dipahami, mudah dikelola, dan dapat diperluas seiring berjalananya waktu. Secara umum, tujuan dari prinsip SOLID adalah untuk mengurangi ketergantungan dan kopling dalam kode, membuat kode lebih modular, dan meningkatkan fleksibilitas dan pemanfaatan kembali kode.

Prinsip SOLID bukanlah aturan yang harus selalu diikuti, tetapi lebih ke arah panduan yang dirancang untuk membantu pengembang membuat perangkat lunak yang lebih mudah dikelola dan dipahami. Mengikuti prinsip-prinsip ini biasanya menghasilkan kode yang lebih bersih, lebih mudah diperluas, dan lebih mudah diuji.

Namun, penting untuk diingat bahwa tidak ada aturan yang absolut dalam pemrograman, dan ada situasi di mana mungkin lebih masuk akal untuk melanggar prinsip-prinsip ini. Misalnya, dalam kasus di mana pengembangan cepat lebih penting daripada pemeliharaan jangka panjang, atau di mana kode yang ditulis tidak mungkin perlu diperluas atau dimodifikasi di masa depan, mungkin tidak perlu untuk menerapkan prinsip SOLID secara ketat.

7.2.2 5 Prinsip SOLID

7.2.2.1 Single Responsibility Principle (SRP)

Setiap class harus memiliki tanggung jawab tunggal. Dalam kata lain, setiap class harus memiliki satu alasan untuk berubah. Tujuan dari prinsip ini adalah untuk mengurangi kompleksitas sistem dengan memisahkan fungsionalitas sehingga perubahan dalam

satu bagian sistem tidak mempengaruhi bagian lainnya. Selain itu juga untuk meminimalkan dampak perubahan pada kode. Jika sebuah class atau modul hanya memiliki satu tanggung jawab, maka ada lebih sedikit risiko bahwa perubahan pada bagian lain dari sistem akan mempengaruhi class atau modul tersebut.

Bayangkan seseorang memiliki sebuah restoran dengan satu karyawan yang melakukan segalanya - memasak, membersihkan, melayani pelanggan, mengurus keuangan, dan sebagainya. Jika karyawan tersebut sakit, seluruh operasi restoran akan terganggu. Dengan menerapkan SRP, maka akan ada orang yang berbeda untuk setiap tugas (seorang koki, pelayan, pembersih, dll.), sehingga jika satu orang tidak bisa bekerja, itu tidak akan mengganggu seluruh operasi.

7.2.2.2 Open/Closed Principle (OCP)

"Buka untuk ekstensi, tutup untuk modifikasi". Artinya, entitas perangkat lunak (kelas, modul, fungsi, dll.) harus dibuka untuk ekstensi tetapi ditutup untuk modifikasi. Ini berarti bahwa kita harus dapat menambahkan fitur atau perilaku baru ke entitas tersebut tanpa mengubah kode yang ada. Tujuannya adalah untuk membuat kode yang lebih mudah diperluas. Dengan membuat entitas yang "terbuka untuk ekstensi tetapi tertutup untuk modifikasi", kita dapat menambahkan fitur atau perilaku baru tanpa mengubah kode yang ada.

Bayangkan kita memiliki sebuah mobil yang tidak dapat dimodifikasi. Jika kita ingin menambahkan fitur baru seperti sistem navigasi, kita harus merusak bagian interior mobil untuk memasangnya. Mobil ideal adalah mobil yang memiliki slot yang dapat disesuaikan untuk memasukkan sistem navigasi baru tanpa harus merusak bagian lain dari mobil.

7.2.2.3 Liskov Substitution Principle (LSP)

Subtypes harus dapat menggantikan tipe dasarnya. Dalam kata lain, jika sebuah program dirancang untuk menggunakan objek tipe tertentu, maka objek dari subtype tersebut harus dapat bekerja dengan program tersebut tanpa perlu modifikasi apa pun pada program tersebut. Tujuannya adalah untuk memastikan bahwa penggantian

tipe dasar dengan subtype-nya tidak akan mempengaruhi perilaku program. Hal ini membuat kode lebih mudah dipahami dan dikelola, karena kita dapat mempercayai bahwa semua objek yang mengimplementasikan tipe atau antarmuka tertentu akan berperilaku dengan cara yang sama.

Bayangkan memiliki burung yang bisa terbang. Kita merancang kandang dengan tinggi tertentu karena burung tersebut bisa terbang. Sekarang, kita memutuskan untuk menambahkan pinguin (yang juga merupakan burung) ke dalam kandang tersebut. Pinguin tidak bisa terbang sehingga kandang tersebut tidak cocok untuk pinguin. Dengan kata lain, meskipun pinguin adalah burung, ia tidak bisa digunakan sebagai pengganti burung yang bisa terbang.

7.2.2.4 Interface Segregation Principle (ISP)

Klien tidak boleh dipaksa untuk bergantung pada antarmuka yang mereka tidak gunakan. Class tidak harus mengimplementasikan metode yang tidak mereka butuhkan. Sebaliknya, jika suatu kelas membutuhkan hanya sebagian dari metode dalam suatu antarmuka, maka antarmuka tersebut sebaiknya dibagi menjadi antarmuka yang lebih kecil yang lebih spesifik.

Tujuannya adalah untuk mengurangi ketergantungan yang tidak perlu antara modul. Dengan memastikan bahwa klien hanya bergantung pada antarmuka yang mereka butuhkan, kita membuat sistem yang lebih modular dan lebih mudah dikelola.

Bayangkan remote kontrol TV yang juga memiliki tombol untuk mengontrol AC, mesin cuci, dan lemari es. Meskipun ini mungkin terdengar efisien, ini bisa menjadi masalah jika kita hanya ingin mengontrol TV. Dengan memisahkan remote menjadi lebih spesifik (satu untuk TV, satu untuk AC, dll.), Kita membuat interaksi menjadi lebih mudah dan intuitif.

7.2.2.5 Dependency Inversion Principle (DIP)

Class tingkat tinggi tidak harus bergantung pada kelas tingkat rendah. Kedua jenis class tersebut harus bergantung pada abstraksi. Dengan demikian detail implementasi harus bergantung pada kebijakan dan bukan sebaliknya. Tujuannya adalah untuk

mengurangi ketergantungan antara modul tingkat tinggi dan modul tingkat rendah, yang membuat sistem lebih mudah dikelola dan diperluas.

Bayangkan lampu yang bisa dinyalakan dengan menekan saklar. Jika lampu tersebut dihubungkan langsung ke saklar, maka kita tidak akan bisa mengubah cara menghidupkannya tanpa merusak koneksi tersebut. Namun, jika lampu dan saklar sama-sama tergantung pada sistem listrik (abstraksi), kita bisa dengan mudah mengganti saklar dengan pengendali suara atau sensor gerak tanpa merusak lampu atau membutuhkan perubahan signifikan pada sistem.

7.3 Kegiatan Praktikum

7.3.1 Kegiatan 1 : Single Responsibility Principle (SRP)

1. Misalkan kita memiliki kelas `UserManager` yang bertanggung jawab untuk mengelola data pengguna dan juga mencetak laporan pengguna. Ini melanggar SRP karena `UserManager` memiliki lebih dari satu tanggung jawab.

```
1. class UserManager:  
2.     def __init__(self):  
3.         self.users = []  
4.  
5.     def add_user(self, user):  
6.         self.users.append(user)  
7.  
8.     def print_user_report(self):  
9.         for user in self.users:  
10.             print(f'User: {user.name}, Email: {user.email}')
```

2. Untuk mematuhi SRP, kita bisa memindahkan tanggung jawab mencetak laporan ke kelas lain, seperti `UserReport`.

```
1. class User:  
2.     def __init__(self, name, email):  
3.         self.name = name  
4.         self.email = email  
5.  
6.  
7. class UserManager:  
8.     def __init__(self):
```

```
9.         self.users = []
10.
11.     def add_user(self, user):
12.         self.users.append(user)
13.
14.
15. class UserReport:
16.     @staticmethod
17.     def print_user_report(users):
18.         for user in users:
19.             print(f'User: {user.name}, Email: {user.email}')
20.
21.
22. # Membuat objek User
23.user1 = User('Alice', 'alice@example.com')
24.user2 = User('Bob', 'bob@example.com')
25.
26. # Membuat objek UserManager dan menambahkan User
27.user_manager = UserManager()
28.user_manager.add_user(user1)
29.user_manager.add_user(user2)
30.
31. # Membuat objek UserReport dan mencetak Laporan
32.user_report = UserReport()
33.user_report.print_user_report(user_manager.users)
```

7.3.2 Kegiatan 2 : Open/Closed Principle (OCP)

1. Misalkan kita memiliki program yang mencetak pesan selamat datang dalam berbagai bahasa. Dalam contoh ini, jika kita ingin menambahkan bahasa baru, kita harus memodifikasi metode greet pada kelas Greeter. Ini melanggar Prinsip Open/Closed Principle (OCP) karena kita harus memodifikasi kelas yang ada untuk menambahkan fungsi baru.

```
1. class Greeter:  
2.     def __init__(self, language):  
3.         self.language = language  
4.  
5.     def greet(self):  
6.         if self.language == 'english':  
7.             return 'Hello!'  
8.         elif self.language == 'spanish':  
9.             return '¡Hola!'  
10.        elif self.language == 'french':  
11.            return 'Bonjour!'  
12.        else:  
13.            return 'Language not supported.'
```

2. Untuk mematuhi Prinsip Open/Closed Principle (OCP), kita bisa merancang kelas Greeter agar dapat menerima objek "Greeting" yang dapat menyapa dalam bahasa apa pun, seperti ini:

```
1. class Greeter:  
2.     def __init__(self, greeter):  
3.         self.greeter = greeter  
4.  
5.     def greet(self):  
6.         return self.greeter.greet()  
7.  
8.  
9. class EnglishGreeter:  
10.    def greet(self):  
11.        return 'Hello!'  
12.  
13.  
14. class SpanishGreeter:  
15.    def greet(self):  
16.        return '¡Hola!'  
17.  
18.
```

```

19. class FrenchGreeter:
20.     def greet(self):
21.         return 'Bonjour!'
22.
23.
24. # Membuat objek greeter dalam berbagai bahasa
25. english_greeter = EnglishGreeter()
26. spanish_greeter = SpanishGreeter()
27. french_greeter = FrenchGreeter()
28.
29. # Membuat objek Greeter dan menggreet dalam berbagai bahasa
30. greeter = Greeter(english_greeter)
31. print(greeter.greet()) # Output: Hello!
32.
33. greeter = Greeter(spanish_greeter)
34. print(greeter.greet()) # Output: ¡Hola!
35.
36. greeter = Greeter(french_greeter)
37. print(greeter.greet()) # Output: Bonjour!

```

7.3.3 Kegiatan 3 : Open/Closed Principle (OCP)

- Perhatikan contoh kode berikut, Ostrich adalah turunan dari Bird, namun Ostrich tidak dapat terbang seperti Bird lainnya. Ini melanggar Liskov Substitution Principle karena kita tidak bisa menggantikan Bird dengan Ostrich dalam konteks terbang.

```

1. class Bird:
2.     def fly(self):
3.         return "I can fly!"
4.
5. class Duck(Bird):
6.     pass
7.
8. class Ostrich(Bird):
9.     def fly(self):
10.        return "I can't fly!"

```

- Untuk mematuhi Liskov Substitution Principle, kita bisa mendefinisikan ulang hirarki kelas kita. Kita bisa membuat kelas FlyingBird dan NonFlyingBird yang merupakan turunan dari Bird, dan memindahkan metode fly ke FlyingBird.

```

1. class Bird:
2.     def __init__(self, name):
3.         self.name = name
4.
5.     def introduce(self):
6.         return f"Hello, I'm a {self.name}."
7.
8.
9. class FlyingBird(Bird):
10.    def fly(self):
11.        return "I can fly!"
12.
13.
14. class NonFlyingBird(Bird):
15.    def walk(self):
16.        return "I can walk!"
17.
18.
19. class Duck(FlyingBird):
20.    def quack(self):
21.        return "Quack!"
22.
23.
24. class Ostrich(NonFlyingBird):
25.    def run(self):
26.        return "I can run fast!"
27.
28.
29. # Membuat objek Duck dan Ostrich
30. duck = Duck("Duck")
31. ostrich = Ostrich("Ostrich")
32.
33. # Duck bisa terbang dan berbunyi "quack"
34. print(duck.introduce()) # Output: Hello, I'm a Duck.
35. print(duck.fly()) # Output: I can fly!
36. print(duck.quack()) # Output: Quack!
37.
38. # Ostrich tidak bisa terbang, dan bisa berjalan dan berlari cepat
39. print(ostrich.introduce()) # Output: Hello, I'm an Ostrich.
40. print(ostrich.walk()) # Output: I can walk!
41. print(ostrich.run()) # Output: I can run fast!

```

7.4 Tugas

Anda diberikan skenario berikut: Anda adalah seorang pengembang di sebuah perusahaan yang membuat perangkat lunak untuk mengelola data pegawai dan laporan gajinya. Anda diminta untuk merancang dan mengimplementasikan sistem ini dengan menggunakan prinsip SOLID. Berikut adalah spesifikasi tugas yang harus dilakukan:

1. Class Employee yang memiliki atribut name, position, dan salary.
2. Class EmployeeManager yang bertanggung jawab untuk menambahkan, menghapus, dan mencari pegawai.
3. Class EmployeeReport yang bertanggung jawab untuk menghasilkan laporan gaji pegawai.
4. Pastikan bahwa setiap Class memiliki satu tanggung jawab (Single Responsibility Principle).
5. Pastikan bahwa aplikasi dapat menambahkan jenis laporan gaji baru tanpa mengubah kelas EmployeeReport (Open/Closed Principle).
6. Pastikan bahwa semua metode pada Class EmployeeManager dapat diterapkan pada kelas turunannya (Liskov Substitution Principle).
7. Pastikan bahwa EmployeeReport tidak bergantung langsung pada EmployeeManager, tetapi hanya pada abstraksi dari EmployeeManager (Dependency Inversion Principle).

Berikut ini merupakan kode dari Tugas diatas, tuliskan analisis Anda tentang bagaimana prinsip SOLID diterapkan pada kode berikut

```
1. class Employee:  
2.     def __init__(self, name, position, salary):  
3.         self.name = name  
4.         self.position = position  
5.         self.salary = salary  
6.  
7. class EmployeeManager:  
8.     def __init__(self):  
9.         self.employees = []  
10.
```

```

11.     def add_employee(self, employee):
12.         self.employees.append(employee)
13.
14.     def remove_employee(self, employee):
15.         self.employees.remove(employee)
16.
17.     def find_employee(self, name):
18.         for employee in self.employees:
19.             if employee.name == name:
20.                 return employee
21.         return None
22.
23. class EmployeeReport:
24.     def generate_report(self, employee):
25.         return f"Employee: {employee.name}, Position: {employee.p
osition}, Salary: {employee.salary}"
26.
27. class SalaryReport(EmployeeReport):
28.     def generate_report(self, employee):
29.         return f"Salary Report: {employee.name} earns {employee.s
alary}"
30.
31. class EmployeeManagerAbstraction:
32.     def get_employee(self, name):
33.         pass
34.
35. class EmployeeManagerAdapter(EmployeeManagerAbstraction):
36.     def __init__(self, manager):
37.         self.manager = manager
38.
39.     def get_employee(self, name):
40.         return self.manager.find_employee(name)
41.
42. class ReportGenerator:
43.     def __init__(self, manager, report):
44.         self.manager = manager
45.         self.report = report
46.
47.     def generate_report(self, name):
48.         employee = self.manager.get_employee(name)
49.         if employee:
50.             return self.report.generate_report(employee)
51.         return f"Employee {name} not found"
52.
53. # Mencoba implementasi
54. manager = EmployeeManager()

```

```
55.manager.add_employee(Employee("John Doe", "Developer", 5000))
56.manager.add_employee(Employee("Jane Smith", "Manager", 6000))
57.
58.adapter = EmployeeManagerAdapter(manager)
59.
60.generator = ReportGenerator(adapter, SalaryReport())
61.print(generator.generate_report("John Doe")) # Output: Salary Re
    port: John Doe earns 5000
62.print(generator.generate_report("Jane Smith")) # Output: Salary
    Report: Jane Smith earns 6000
```

BAB 8

PyQt



8.1 Tujuan

3. Dapat memahami konsep dan penggunaan PyQt
4. Dapat mengimplementasikan PyQt untuk membuat tampilan GUI sederhana

8.2 Pengantar

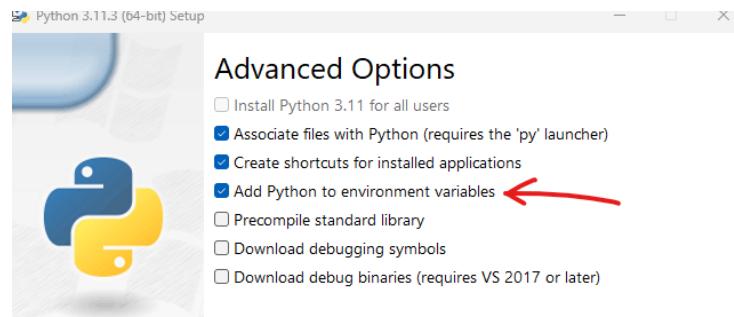
8.2.1 Pendahuluan PyQt

PyQt adalah set binding Python untuk pustaka Qt yang digunakan untuk membuat aplikasi GUI (Graphical User Interface). PyQt dikembangkan oleh Riverbank Computing Limited. Ada dua versi PyQt utama yang tersedia yaitu PyQt4 dan PyQt5. Versi ini merujuk pada versi Qt library yang digunakan. PyQt5 mendukung Qt5, dan PyQt4 mendukung Qt4.

8.2.2 Instalasi dan Setup PyQt untuk Windows

Step 1. Instalasi Python

Instal Python dari situs web resmi Python di <https://www.python.org/downloads/>. Pastikan untuk mencentang opsi "Add Python to PATH" saat melakukan instalasi.



Step 2. Instalasi PyQt

Masuk ke Command Prompt kemudian ketikan perintah berikut untuk melakukan instalasi

```
1. pip install pyqt5
```

Step 3. Verifikasi Instalasi

Setelah instalasi selesai ketikkan perintah berikut pada python interpreter

```
|1. import PyQt5
```

Jika tidak ada error maka proses instalasi PyQt5 sudah selesai

```
C:\Users\AIO ASUS>python
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 ]
Type "help", "copyright", "credits" or "license" for more information.
>>> import PyQt5
>>> |
```

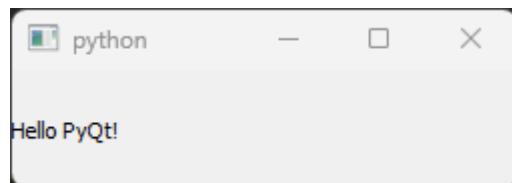
8.2.3 Konsep Dasar PyQt

Dalam PyQt, aplikasi GUI dibuat dengan membuat sebuah instance dari QApplication. Setelah itu, widget (seperti jendela, tombol, atau label) dibuat dan dimodifikasi sesuai kebutuhan. Setelah semua widget disiapkan, aplikasi memasuki loop event dengan memanggil metode exec_() dari QApplication. Loop event ini yang bertanggung jawab untuk menangani semua event seperti klik mouse atau ketukan keyboard.

Berikut adalah contoh sederhana dari aplikasi PyQt:

```
1. from PyQt5.QtWidgets import QApplication, QLabel
2.
3. app = QApplication([])
4. label = QLabel('Hello PyQt!')
5. label.show()
6. app.exec_()
```

Dalam contoh ini, sebuah aplikasi dan label dibuat. Label tersebut kemudian ditampilkan, dan aplikasi memasuki loop event. Ketika kode dijalankan maka akan menampilkan output seperti berikut



8.2.4 Struktur Aplikasi PyQt

Aplikasi PyQt biasanya memiliki struktur berikut:

- Import modul yang diperlukan: Melakukan import terhadap modul-modul yang diperlukan termasuk modul PyQt5.QtWidgets yang berisi kelas dasar yang diperlukan untuk membuat aplikasi GUI.
- Membuat aplikasi: Aplikasi dibuat dengan membuat instance dari QApplication.
- Membuat widget: Widget adalah elemen-elemen GUI seperti jendela, tombol, label, dan lainnya. Kita dapat membuat widget dengan membuat instance dari kelas-kelas seperti QWidget, QPushButton, QLabel, dan lainnya.
- Menyiapkan widget: Setelah widget dibuat, kita bisa mengubah propertinya seperti ukuran, posisi, dan teks. Kita juga bisa mengatur tata letak widget dengan menggunakan kelas-kelas seperti QVBoxLayout, QHBoxLayout, dan QGridLayout.
- Menampilkan widget: Widget ditampilkan dengan memanggil metode show().
- Memasuki loop event: Aplikasi memasuki loop event dengan memanggil metode exec_() dari QApplication. Loop event ini yang bertanggung jawab untuk menangani semua event seperti klik mouse atau ketukan keyboard.

Berikut adalah contoh struktur kode PyQt5 menggunakan konsep pemrograman berorientasi objek (OOP):

```
1. # Import necessary modules from PyQt5
2. from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel
3.
4. # Define a MainWindow class that inherits from QMainWindow
5. class MainWindow(QMainWindow):
6.     def __init__(self):
7.         # Call the constructor of the parent class
8.         super().__init__()
9.
10.        # Set some basic attributes for the window
```

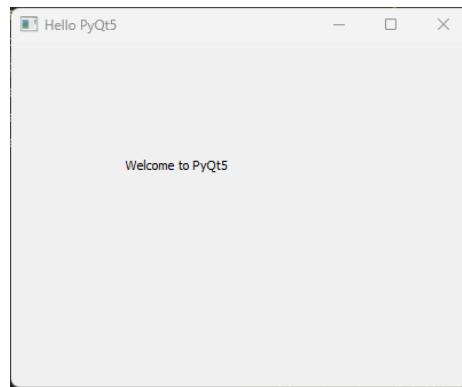
```

11.         self.title = "Hello PyQt5"
12.         self.top = 100
13.         self.left = 100
14.         self.width = 400
15.         self.height = 300
16.
17.         # Call the method that initializes the window
18.         self.initWindow()
19.
20.     def initWindow(self):
21.         # Create a QLabel widget and set it as the central widget o
f the window
22.         self.label = QLabel("Welcome to PyQt5", self)
23.         self.label.adjustSize() # Adjust the size of the Label to
fit the text
24.         self.label.move(100, 100) # Move the Label to the position
(100, 100)
25.
26.         # Set the title of the window
27.         self.setWindowTitle(self.title)
28.         # Set the position and size of the window
29.         self.setGeometry(self.top, self.left, self.width, self.heig
ht)
30.         # Show the window
31.         self.show()
32.
33.# Create an instance of QApplication
34.app = QApplication([])
35.# Create an instance of MainWindow
36.window = MainWindow()
37.# Start the event loop
38.app.exec_()

```

Dalam contoh ini, kita membuat kelas `MainWindow` yang mewarisi dari `QMainWindow`. Kelas ini memiliki metode `__init__()` di mana kita mengatur beberapa atribut dasar seperti judul, posisi, dan ukuran jendela. Kita juga memanggil metode `initWindow()` di mana kita mengatur lebih banyak properti jendela dan menampilkan jendela.

Kemudian, kita membuat instance dari `QApplication` dan `MainWindow`, dan memulai loop event dengan memanggil `app.exec_()`.



8.2.5 Widget dan Layout

8.2.5.1 QLabel

QLabel adalah widget yang digunakan untuk menampilkan teks atau gambar

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         self.label = QLabel('Hello PyQt!', self)
8.         self.label.adjustSize()
9.
10. app = QApplication([])
11. window = MainWindow()
12. window.show()
13. app.exec_()
```

8.2.5.2 QPushButton

QPushButton adalah tombol yang bisa diklik oleh pengguna

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
```

```
6.  
7.         self.button = QPushButton('Click Me!', self)  
8.         self.button.clicked.connect(self.on_button_clicked)  
9.  
10.    def on_button_clicked(self):  
11.        print('Button clicked!')  
12.  
13.app = QApplication([])  
14.window = MainWindow()  
15.window.show()  
16.app.exec_()
```

8.2.5.3 QLineEdit

QLineEdit adalah kotak teks satu baris yang bisa digunakan pengguna untuk memasukkan teks..

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QLineEdit  
2.  
3. class MainWindow(QMainWindow):  
4.     def __init__(self):  
5.         super().__init__()  
6.  
7.         self.lineEdit = QLineEdit(self)  
8.         self.lineEdit.textChanged.connect(self.on_text_changed)  
9.  
10.    def on_text_changed(self, text):  
11.        print('Text changed:', text)  
12.  
13.app = QApplication([])  
14.window = MainWindow()  
15.window.show()  
16.app.exec_()
```

8.2.5.4 QTextEdit

QTextEdit adalah kotak teks multi-baris yang bisa digunakan pengguna untuk memasukkan teks.

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QTextEdit
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         self.textEdit = QTextEdit(self)
8.         self.textEdit.textChanged.connect(self.on_text_changed)
9.
10.    def on_text_changed(self):
11.        print('Text changed')
12.
13.app = QApplication([])
14.window = MainWindow()
15.window.show()
16.app.exec_()
17.
```

8.2.5.5 QCheckBox

QCheckBox adalah kotak centang yang bisa dicentang atau tidak dicentang oleh pengguna

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QCheckBox
2. from PyQt5.QtCore import Qt
3.
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.
8.         self.checkBox = QCheckBox('Check Me', self)
9.         self.checkBox.stateChanged.connect(self.on_state_changed)
10.
11.    def on_state_changed(self, state):
12.        if state == Qt.Checked:
13.            print('Checked')
14.        else:
15.            print('Unchecked')
16.
17.app = QApplication([])
18.window = MainWindow()
19.window.show()
```

```
20. app.exec_()
21.
```

8.2.5.6 QRadioButton

QRadioButton adalah tombol radio yang biasanya digunakan dalam grup di mana pengguna bisa memilih satu dari beberapa pilihan

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QRadioButton
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         self.radioButton = QRadioButton('Select Me', self)
8.         self.radioButton.toggled.connect(self.on_toggled)
9.
10.    def on_toggled(self, is_checked):
11.        if is_checked:
12.            print('Selected')
13.
14. app = QApplication([])
15. window = MainWindow()
16. window.show()
17. app.exec_()
18.
```

8.2.5.7 QComboBox

QComboBox adalah kotak kombinasi yang memungkinkan pengguna memilih satu dari beberapa pilihan

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QComboBox
2.
```

```

3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         self.comboBox = QComboBox(self)
8.         self.comboBox.addItems(['Option 1', 'Option 2', 'Option 3'])
9.         self.comboBox.currentIndexChanged.connect(self.on_index_changed)
10.
11.    def on_index_changed(self, index):
12.        print('Selected option:', self.comboBox.itemText(index))
13.
14. app = QApplication([])
15.window = MainWindow()
16.window.show()
17.app.exec_()
18.

```

8.2.5.8 QSlider

QSlider adalah slider yang memungkinkan pengguna memilih nilai dari rentang nilai dengan menggeser penanda.

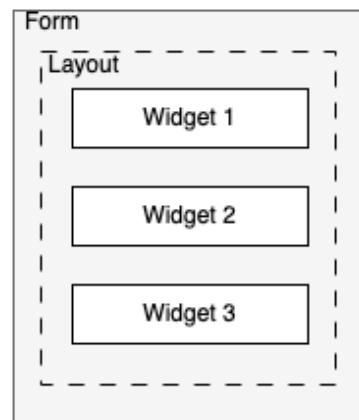
```

1. from PyQt5.QtWidgets import QApplication, QMainWindow, QSlider
2. from PyQt5.QtCore import Qt
3.
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.
8.         self.slider = QSlider(Qt.Horizontal, self)
9.         self.slider.setMinimum(0)
10.        self.slider.setMaximum(100)
11.        self.slider.valueChanged.connect(self.on_value_changed)
12.
13.    def on_value_changed(self, value):
14.        print('Slider value:', value)
15.
16. app = QApplication([])
17.window = MainWindow()
18.window.show()
19.app.exec_()

```

| 20.

8.2.5.9 QVBoxLayout

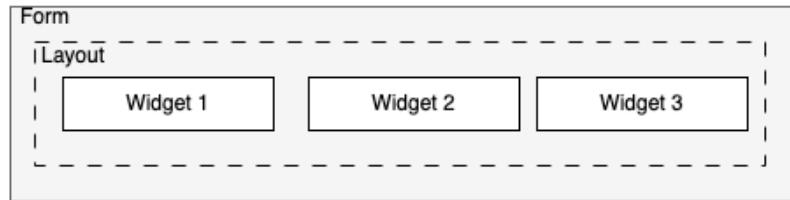


QVBoxLayout adalah layout yang mengatur widget secara vertikal. Berikut adalah contoh penggunaannya:

```
1. from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton
2.
3. class MainWindow(QWidget):
4.     def __init__(self):
5.         super().__init__()
6.
7.         layout = QVBoxLayout()
8.
9.         layout.addWidget(QPushButton('Button 1'))
10.        layout.addWidget(QPushButton('Button 2'))
```

```
11.         layout.addWidget(QPushButton('Button 3'))
12.
13.         self.setLayout(layout)
14.
15.app = QApplication([])
16.window = MainWindow()
17.window.show()
18.app.exec_()
```

8.2.5.10 QBoxLayout



QBoxLayout adalah layout yang mengatur widget secara horizontal. Berikut adalah contoh penggunaannya:

```
1. from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout, QPushButton
2.
3. class MainWindow(QWidget):
4.     def __init__(self):
5.         super().__init__()
6.
7.         layout = QHBoxLayout()
8.
9.         layout.addWidget(QPushButton('Button 1'))
10.        layout.addWidget(QPushButton('Button 2'))
11.        layout.addWidget(QPushButton('Button 3'))
12.
13.        self.setLayout(layout)
14.
15.app = QApplication([])
16.window = MainWindow()
17.window.show()
18.app.exec_()
```

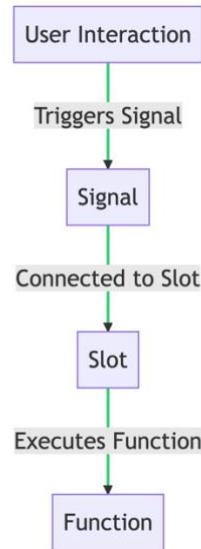
8.2.5.11 QGridLayout



QGridLayout adalah layout yang mengatur widget dalam bentuk grid. Berikut adalah contoh penggunaannya:

```
1. from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout, QPushButton
2.
3. class MainWindow(QWidget):
4.     def __init__(self):
5.         super().__init__()
6.
7.         layout = QGridLayout()
8.
9.         layout.addWidget(QPushButton('Button 1'), 0, 0)
10.        layout.addWidget(QPushButton('Button 2'), 0, 1)
11.        layout.addWidget(QPushButton('Button 3'), 1, 0)
12.        layout.addWidget(QPushButton('Button 4'), 1, 1)
13.        layout.addWidget(QPushButton('Button 5'), 2, 0, 1, 2)
14.        self.setLayout(layout)
15.
16.app = QApplication([])
17.window = MainWindow()
18.window.show()
19.app.exec_()
```

8.2.6 Signal dan Slot



Signal dan Slot adalah fitur fundamental dari PyQt5 dan merupakan mekanisme yang memungkinkan komunikasi antara objek dalam aplikasi GUI.

Signal

Signal dalam PyQt5 adalah sebuah event yang bisa dipancarkan oleh objek tertentu. Event ini bisa berupa aksi pengguna seperti klik tombol, mengubah nilai slider, memilih item dari dropdown, dan lainnya. Signal juga bisa dipancarkan oleh program itu sendiri, misalnya ketika timer berakhir atau sebuah file selesai didownload.

Slot

Slot adalah fungsi atau metode yang dipanggil sebagai respons terhadap sebuah signal. Slot bisa berupa fungsi apa saja, termasuk fungsi yang dibuat oleh pengguna. Slot bisa memiliki parameter, dan parameter ini bisa dipasok oleh signal.

Mekanisme Penggunaan

Untuk menggunakan signal dan slot, kita perlu melakukan dua hal:

1. Membuat slot, yaitu fungsi yang akan dipanggil ketika signal dipancarkan.

2. Menghubungkan signal ke slot menggunakan metode `connect()`.

Berikut adalah contoh penggunaan signal dan slot dalam PyQt5:

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton  
2.  
3. class MainWindow(QMainWindow):  
4.     def __init__(self):  
5.         super().__init__()  
6.  
7.         self.button = QPushButton('Click Me!', self)  
8.         self.button.clicked.connect(self.on_button_clicked)  
9.  
10.    def on_button_clicked(self):  
11.        print('Button clicked!')  
12.  
13.app = QApplication([])  
14.window = MainWindow()  
15.window.show()  
16.app.exec_()
```

Dalam contoh ini, kita membuat sebuah QPushButton dan menghubungkan signal `clicked` dari tombol tersebut ke slot `on_button_clicked`. Ketika tombol diklik, signal `clicked` akan dipancarkan dan slot `on_button_clicked` akan dipanggil, mencetak 'Button clicked!' ke console.

8.3 Kegiatan Praktikum

8.3.1 Kegiatan 1 : Layout

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

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QPushButton, QLineEdit, QVBoxLayout, QHBoxLayout, QGridLayout, QWidget  
2.  
3. class MainWindow(QMainWindow):  
4.     def __init__(self):  
5.         super().__init__()  
6.  
7.         self.setWindowTitle('Layouts Example')
```

```

8.
9.         # Create some widgets
10.        label1 = QLabel('Label 1')
11.        label2 = QLabel('Label 2')
12.        label3 = QLabel('Label 3')
13.        button1 = QPushButton('Button 1')
14.        button2 = QPushButton('Button 2')
15.        button3 = QPushButton('Button 3')
16.        textbox1 = QLineEdit()
17.        textbox2 = QLineEdit()
18.        textbox3 = QLineEdit()
19.
20.        # Create a QVBoxLayout
21.        vbox_layout = QVBoxLayout()
22.        vbox_layout.addWidget(QLabel('Vertical Layout'))
23.        vbox_layout.addWidget(label1)
24.        vbox_layout.addWidget(button1)
25.        vbox_layout.addWidget(textbox1)
26.
27.        # Create a QHBoxLayout
28.        hbox_layout = QHBoxLayout()
29.        hbox_layout.addWidget(QLabel('Horizontal Layout'))
30.        hbox_layout.addWidget(label2)
31.        hbox_layout.addWidget(button2)
32.        hbox_layout.addWidget(textbox2)
33.
34.        # Create a QGridLayout
35.        grid_layout = QGridLayout()
36.        grid_layout.addWidget(QLabel('Grid Layout'), 0, 0, 1, 2)
37.        grid_layout.addWidget(label3, 1, 0)
38.        grid_layout.addWidget(button3, 1, 1)
39.        grid_layout.addWidget(textbox3, 2, 0, 1, 2)
40.
41.        # Create a QVBoxLayout for the main layout and add the other layouts
42.        main_layout = QVBoxLayout()
43.        main_layout.addLayout(vbox_layout)
44.        main_layout.addLayout(hbox_layout)
45.        main_layout.addLayout(grid_layout)
46.
47.        # Create a QWidget, set its layout, and set it as the central widget
48.        container = QWidget()
49.        container.setLayout(main_layout)
50.        self.setCentralWidget(container)
51.

```

```
52.app = QApplication([])
53.window = MainWindow()
54.window.show()
55.app.exec_()
56.
```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

8.3.2 Kegiatan 2 : Widget

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

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QPushButton, QLineEdit, QVBoxLayout, QWidget
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         # Set the window title
8.         self.setWindowTitle('My App')
9.
10.        # Create a QLabel, QPushButton, and QLineEdit
11.        self.label = QLabel()
12.        self.button = QPushButton('Show Text')
13.        self.textbox = QLineEdit()
14.
15.        # Connect the button's clicked signal to the slot
16.        self.button.clicked.connect(self.on_button_clicked)
17.
18.        # Create a QVBoxLayout and add the widgets
19.        layout = QVBoxLayout()
20.        layout.addWidget(self.textbox)
21.        layout.addWidget(self.button)
22.        layout.addWidget(self.label)
23.
24.        # Create a QWidget, set its layout, and set it as the central widget
25.        container = QWidget()
26.        container.setLayout(layout)
27.        self.setCentralWidget(container)
28.
```

```

29. # Define the slot that will be called when the button is clicked
30. def on_button_clicked(self):
31.     # Get the text from the QLineEdit and set it as the QLabel's text
32.     text = self.textbox.text()
33.     self.label.setText(text)
34.
35. # Create a QApplication, create a MainWindow, show the main window,
   and start the event loop
36.app = QApplication([])
37.window = MainWindow()
38.window.show()
39.app.exec_()

```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

8.3.3 Kegiatan 3 : Signal dan Slot

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

```

1. from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QPushButton, QLineEdit, QVBoxLayout, QWidget
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         # Set the window title
8.         self.setWindowTitle('Kalkulator Sederhana')
9.
10.        # Create two QLineEdit widgets for the user to enter numbers
11.        self.first_number = QLineEdit()
12.        self.second_number = QLineEdit()
13.
14.        # Create a QLabel to display input and the result
15.        self.labelinput1 = QLabel("Masukkan Angka 1")
16.        self.labelinput2 = QLabel("Masukkan Angka 2")
17.        self.hasil_label = QLabel("Hasil Perhitungan ")
18.        self.result_label = QLabel()

```

```

19.
20.      # Create four QPushButton widgets for the mathematical operations
21.      self.add_button = QPushButton('Tambah')
22.      self.subtract_button = QPushButton('Kurang')
23.      self.multiply_button = QPushButton('Kali')
24.      self.divide_button = QPushButton('Bagi')
25.
26.      # Connect each button's clicked signal to the appropriate slot
27.      self.add_button.clicked.connect(self.add_numbers)
28.      self.subtract_button.clicked.connect(self.subtract_numbers)
29.      self.multiply_button.clicked.connect(self.multiply_numbers)
30.      self.divide_button.clicked.connect(self.divide_numbers)
31.
32.      # Create a QVBoxLayout and add the widgets
33.      layout = QVBoxLayout()
34.      layout.addWidget(self.labelinput1)
35.      layout.addWidget(self.first_number)
36.      layout.addWidget(self.labelinput2)
37.      layout.addWidget(self.second_number)
38.      layout.addWidget(self.add_button)
39.      layout.addWidget(self.subtract_button)
40.      layout.addWidget(self.multiply_button)
41.      layout.addWidget(self.divide_button)
42.      layout.addWidget(self.hasil_label)
43.      layout.addWidget(self.result_label)
44.
45.      # Create a QWidget, set its layout, and set it as the central widget
46.      container = QWidget()
47.      container.setLayout(layout)
48.      self.setCentralWidget(container)
49.
50.      # Define the slots that will be called when the buttons are clicked
51.      def add_numbers(self):
52.          # Add the numbers entered by the user and display the result
53.          result = float(self.first_number.text()) + float(self.second_number.text())
54.          self.result_label.setText(str(result))
55.
56.      def subtract_numbers(self):

```

```

57.      # Subtract the second number from the first and display the result
58.      result = float(self.first_number.text()) - float(self.second_number.text())
59.      self.result_label.setText(str(result))
60.
61.  def multiply_numbers(self):
62.      # Multiply the numbers entered by the user and display the result
63.      result = float(self.first_number.text()) * float(self.second_number.text())
64.      self.result_label.setText(str(result))
65.
66.  def divide_numbers(self):
67.      # Divide the first number by the second and display the result
68.      result = float(self.first_number.text()) / float(self.second_number.text())
69.      self.result_label.setText(str(result))
70.
71.# Create a QApplication, create a MainWindow, show the main window
    , and start the event loop
72.app = QApplication([])
73.window = MainWindow()
74.window.show()
75.app.exec_()
76.

```

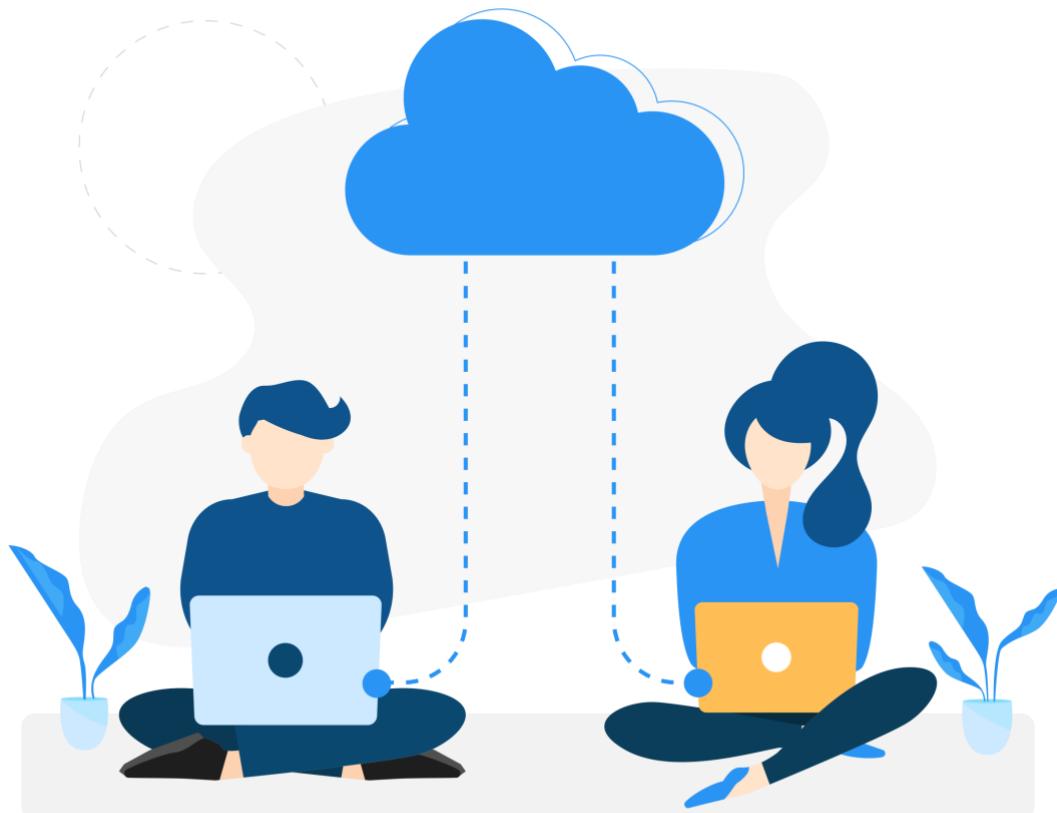
2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

8.4 Tugas

Berdasarkan pada Praktikum Kegiatan 3, tambahkan button Modulus yang akan menghasilkan sisa pembagian, dan button Pangkat yang akan menghasilkan hasil pangkat dari angka pertama dengan pangkat angka ke dua.

BAB 9

Form dan Database



9.1 Tujuan

1. Dapat memahami pembuatan form dengan PyQt5
2. Dapat mengimplementasikan operasi database dengan PyQt5

9.2 Pengantar

9.2.1 Form

Dalam konteks pengembangan aplikasi GUI (Graphical User Interface), form adalah jendela atau layar yang berisi elemen-elemen GUI seperti tombol, kotak teks, label, dan lainnya. Form digunakan untuk berinteraksi dengan pengguna, misalnya untuk menerima input dari pengguna atau menampilkan informasi kepada pengguna. Dalam PyQt5, kita dapat membuat form dengan menggunakan class QWidget atau class turunannya seperti QMainWindow dan QDialog. Secara umum, kita akan menggunakan QWidget untuk membuat widget kustom atau form sederhana, QMainWindow untuk jendela utama aplikasi, dan QDialog untuk dialog dan kotak pesan.

9.2.1.1 QWidget

QWidget adalah kelas dasar untuk semua objek antarmuka pengguna, atau widget, dalam PyQt5. QWidget adalah kelas yang sangat fleksibel yang dapat digunakan sebagai dasar untuk membuat berbagai jenis widget, termasuk form dan dialog. QWidget dapat berisi widget lain, dan kita dapat menentukan layout untuk menata widget-widget tersebut.

```
1. from PyQt5.QtWidgets import QApplication, QWidget
2.
3. class MyForm(QWidget):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle('My Form')
7.
8. app = QApplication([])
9. form = MyForm()
10.form.show()
11.app.exec_()
```

9.2.1.2 QMainWindows

QMainWindow adalah kelas khusus yang diturunkan dari QWidget dan dirancang untuk menjadi jendela utama aplikasi Anda. QMainWindow memiliki struktur yang telah ditentukan sebelumnya yang mencakup area konten utama dan beberapa panel tambahan, seperti toolbar, menu, dan status bar. Kita biasanya akan menambahkan widget lain ke area konten utama QMainWindow.

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. class MainWindow(QMainWindow):
3.     def __init__(self):
4.         super().__init__()
5.         self.setWindowTitle('Main Window')
6. app = QApplication([])
7. main_window = MainWindow()
8. main_window.show()
9. app.exec_()
```

9.2.1.3 QDialog

QDialog adalah kelas lain yang diturunkan dari QWidget dan dirancang untuk digunakan sebagai dialog. Dialog adalah jendela sekunder yang digunakan untuk berinteraksi dengan pengguna, biasanya untuk mendapatkan input atau memberikan informasi. QDialog memiliki beberapa fitur tambahan yang dirancang khusus untuk penggunaan ini, seperti kemampuan untuk menampilkan dialog secara modal (yaitu, mencegah pengguna berinteraksi dengan jendela lain sampai dialog ditutup).

```
1. from PyQt5.QtWidgets import QApplication, QDialog
2.
3. class MyDialog(QDialog):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle('My Dialog')
```

```
7.  
8. app = QApplication([])  
9. dialog = MyDialog()  
10.dialog.show()  
11.app.exec_()  
12.
```

9.2.1.4 Menghubungkan dua form atau lebih

Berikut ini contoh kode jika kita ingin menghubungkan dua form dalam PyQt5. Dalam kode ini, kita membuat dua class: MainWindow dan DialogWindow. MainWindow adalah subclass dari QMainWindow dan DialogWindow adalah subclass dari QDialog. MainWindow memiliki QPushButton, dan ketika tombol ini diklik, sebuah instance dari DialogWindow dibuat dan ditampilkan.

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton,  
   QDialog  
2.  
3. class MainWindow(QMainWindow):  
4.     def __init__(self):  
5.         super().__init__()  
6.         self.setWindowTitle('Main Window')  
7.  
8.         self.button = QPushButton("Open Dialog", self)  
9.         self.button.clicked.connect(self.open_dialog)  
10.  
11.    def open_dialog(self):  
12.        self.dialog = DialogWindow()  
13.        self.dialog.show()  
14.  
15. class DialogWindow(QDialog):  
16.     def __init__(self):  
17.         super().__init__()  
18.         self.setWindowTitle('Dialog Window')  
19.  
20. if __name__ == "__main__":  
21.     app = QApplication([])  
22.     main_window = MainWindow()  
23.     main_window.show()  
24.     app.exec_()
```

9.2.2 Database

Database adalah kumpulan data yang terorganisir. Dalam konteks aplikasi PyQt5, database biasanya digunakan untuk menyimpan data yang dihasilkan atau digunakan oleh aplikasi, seperti data pengguna, data transaksi, dan lainnya. PyQt5 mendukung berbagai jenis database melalui modul QSql, termasuk SQLite, PostgreSQL, MySQL, dan lainnya. kita dapat melakukan operasi database seperti membuat tabel, memasukkan data, mengambil data, memperbarui data, dan menghapus data dari database.

9.2.2.1 SQLite

SQLite adalah sistem manajemen database relasional (RDBMS) yang disimpan dalam satu file di disk. Ini adalah perangkat lunak open source dan salah satu sistem database yang paling banyak digunakan di dunia. Berikut adalah beberapa fitur utama SQLite:

- Serverless: SQLite tidak menggunakan arsitektur server-client seperti kebanyakan sistem manajemen database lainnya. Sebaliknya, SQLite membaca dan menulis langsung ke file disk.
- Zero Configuration: SQLite tidak memerlukan konfigurasi atau instalasi terpisah untuk mulai bekerja. Ini membuat SQLite menjadi pilihan yang baik untuk pengembangan aplikasi yang cepat dan prototyping.
- Portable: Database SQLite adalah file disk tunggal yang dapat dengan mudah ditransfer antara sistem.

- Transaksi ACID: SQLite mendukung transaksi ACID (Atomicity, Consistency, Isolation, Durability), yang berarti bahwa semua operasi database adalah atomik dan konsisten, dan mereka tetap demikian bahkan dalam kondisi kegagalan sistem atau crash.
- Mendukung SQL Standar: SQLite mendukung sebagian besar SQL standar, yang memungkinkan pengguna untuk melakukan operasi database kompleks seperti join dan subquery.

SQLite biasanya digunakan dalam aplikasi yang tidak memerlukan skala besar dan di mana kecepatan dan simplicitas lebih penting daripada fitur-fitur lanjutan seperti replikasi dan partisi. Contoh penggunaan SQLite termasuk aplikasi desktop, aplikasi mobile, dan aplikasi IoT.

9.2.2.2 Koneksi ke Database

Untuk melakukan koneksi ke database kita dapat menggunakan beberapa class yang sudah disediakan oleh PyQt5 seperti QSqlDatabase dan QSqlQuery. Berikut ini merupakan contoh kode koneksi PyQt5 ke database SQLite

```

1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. from PyQt5.QtSql import QSqlDatabase, QSqlQuery
3.
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.         self.setWindowTitle('Main Window')
8.
9.         # Membuat koneksi ke database SQLite
10.        self.db = QSqlDatabase.addDatabase("QSQLITE")
11.        self.db.setDatabaseName("database.sqlite") # Database disimpan dalam file disk
12.
13.        # Membuka koneksi ke database
14.        if not self.db.open():
15.            print("Failed to open database")
16.
17.        # Membuat tabel dalam database
18.        query = QSqlQuery()

```

```

19.         query.exec_("CREATE TABLE people (id INTEGER PRIMARY KEY,
20.                         name TEXT)")
21.         # Menambahkan data ke tabel
22.         query.exec_("INSERT INTO people (name) VALUES ('Setiawan A
23.                         rif')")
24.         # Mengambil data dari tabel
25.         query.exec_("SELECT * FROM people")
26.         while query.next():
27.             print(query.value(1))
28.
29.if __name__ == "__main__":
30.     app = QApplication([])
31.     main_window = MainWindow()
32.     main_window.show()
33.     app.exec_()

```

Dalam kode ini, kita membuat class MainWindow yang merupakan subclass dari QMainWindow. Di dalam constructor MainWindow, kita membuat koneksi ke database SQLite dengan menggunakan class QSqlDatabase. Kita kemudian membuka koneksi ke database bernama database.sqlite, membuat tabel dalam database, menambahkan data ke tabel, dan mengambil data dari tabel. Semua operasi database ini dilakukan dengan menggunakan class QSqlQuery.

9.2.2.3 Operasi CRUD

CRUD adalah singkatan dari Create, Read, Update, dan Delete. Ini adalah empat operasi dasar yang dapat dilakukan pada data dalam database.

Create: Membuat atau menambahkan data baru ke dalam database.

```
INSERT INTO people (name) VALUES ('John Doe');
```

Read: Membaca atau mengambil data dari database.

```
SELECT * FROM people;
```

Update: Memperbarui atau mengubah data yang sudah ada dalam database.

```
UPDATE people SET name = 'Jane Doe' WHERE id = 1;
```

Delete: Menghapus data dari database.

```
DELETE FROM people WHERE id = 1;
```

Berikut ini merupakan contoh aplikasi CRUD menggunakan PyQt5

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. from PyQt5.QtSql import QSqlDatabase, QSqlQuery
3.
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.         self.setWindowTitle('Main Window')
8.
9.         # Membuat koneksi ke database SQLite
10.        self.db = QSqlDatabase.addDatabase("QSQLITE")
11.        self.db.setDatabaseName("database.sqlite")
12.
13.        # Membuka koneksi ke database
14.        if not self.db.open():
15.            print("Failed to open database")
16.
17.        # Membuat tabel dalam database (Create)
18.        query = QSqlQuery()
19.        query.exec_("CREATE TABLE people (id INTEGER PRIMARY KEY,
name TEXT)")
20.
21.        # Menambahkan data ke tabel (Create)
22.        query.exec_("INSERT INTO people (name) VALUES ('Setiawan
Arif')")
23.
24.        # Mengambil data dari tabel (Read)
25.        query.exec_("SELECT * FROM people")
26.        while query.next():
27.            for i in range(2): #range(2) karena terdapat 2 kolom
dalam database
28.                print(query.value(i))
29.
30.        # Memperbarui data dalam tabel (Update)
```

```

31.         query.exec_("UPDATE people SET name = 'Jane Doe' WHERE id
   = 1")
32.
33.         # Menghapus data dari tabel (Delete)
34.         query.exec_("DELETE FROM people WHERE id = 1")
35.
36.if __name__ == "__main__":
37.     app = QApplication([])
38.     main_window = MainWindow()
39.     main_window.show()
40.     app.exec_()
41.

```

9.2.3 Widget dalam operasi CRUD

Seiring dengan meningkatnya kompleksitas aplikasi yang dibuat, maka terdapat beberapa widget tambahan yang perlu kita ketahui untuk memudahkan dalam membangun aplikasi CRUD

9.2.3.1 QTableWidget

QTableView adalah widget yang digunakan untuk menampilkan data dalam format tabel. QTableView menggunakan model untuk mengatur dan mengakses data. Model yang paling sering digunakan adalah QStandardItemModel dan QSqlTableModel. Berikut adalah contoh penggunaan QTableView:

```

1. from PyQt5.QtWidgets import QApplication, QMainWindow, QTableView
2. from PyQt5.QtGui import QStandardItemModel, QStandardItem
3.
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.
8.         # Membuat model
9.         self.model = QStandardItemModel()
10.        self.model.setHorizontalHeaderLabels(['Name', 'Age'])
11.
12.        # Menambahkan data ke model
13.        names = ['Alice', 'Bob', 'Charlie']
14.        ages = [25, 32, 18]

```

```

15.     for i in range(3):
16.         name_item = QStandardItem(names[i])
17.         age_item = QStandardItem(str(ages[i]))
18.         self.model.appendRow([name_item, age_item])
19.
20.     # Membuat QTableView dan mengatur modelnya
21.     self.table = QTableView()
22.     self.table.setModel(self.model)
23.
24.     self.setCentralWidget(self.table)
25.
26.if __name__ == "__main__":
27.     app = QApplication([])
28.     window = MainWindow()
29.     window.show()
30.     app.exec_()
31.

```

9.2.3.2 QListWidget

QListWidget adalah widget yang digunakan untuk menampilkan data dalam format daftar. Berbeda dengan QTableView, QListWidget tidak menggunakan model dan lebih sederhana untuk digunakan. Berikut adalah contoh penggunaan QListWidget :

```

1. from PyQt5.QtWidgets import QApplication, QMainWindow, QListWidget
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         # Membuat QListWidget
8.         self.list_widget = QListWidget()
9.
10.        # Menambahkan item ke QListWidget
11.        self.list_widget.addItem('Alice')
12.        self.list_widget.addItem('Bob')
13.        self.list_widget.addItem('Charlie')
14.
15.        self.setCentralWidget(self.list_widget)
16.
17.if __name__ == "__main__":

```

```
18.     app = QApplication([])
19.     window = MainWindow()
20.     window.show()
21.     app.exec_()
```

9.2.3.3 QDialog

QDialog adalah kelas dasar untuk dialog. Dialog adalah jendela independen yang digunakan untuk tugas-tugas sementara yang memerlukan interaksi pengguna.

Berikut adalah contoh penggunaan QDialog:

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog, QPushButton, QVBoxLayout
2.
3. class CustomDialog(QDialog):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle("Hello!")
7.         layout = QVBoxLayout()
8.         button = QPushButton("Close")
9.         button.clicked.connect(self.close)
10.        layout.addWidget(button)
11.        self.setLayout(layout)
12.
13. class MainWindow(QMainWindow):
14.     def __init__(self):
15.         super().__init__()
16.         button = QPushButton("Open Dialog")
17.         button.clicked.connect(self.open_dialog)
18.         self.setCentralWidget(button)
19.
20.     def open_dialog(self):
```

```
21.         dialog = CustomDialog()
22.         dialog.exec_()
23.
24.if __name__ == "__main__":
25.     app = QApplication([])
26.     window = MainWindow()
27.     window.show()
28.     app.exec_()
```

9.2.3.4 QMessageBox

QMessageBox adalah dialog modal yang menyediakan pesan standar. Ini digunakan untuk menampilkan pesan, pertanyaan, atau meminta pengguna untuk memasukkan informasi. Berikut adalah contoh penggunaan QMessageBox:

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
   , QPushButton
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.         button = QPushButton("Show Message")
7.         button.clicked.connect(self.show_message)
8.         self.setCentralWidget(button)
9.
10.    def show_message(self):
11.        QMessageBox.information(self, "Hello!", "This is a message
12.        .")
13.if __name__ == "__main__":
14.    app = QApplication([])
15.    window = MainWindow()
16.    window.show()
17.    app.exec_()
18.
```

9.2.3.5 QDateTimeEdit

QDateEdit adalah widget yang memungkinkan pengguna untuk memilih dan memasukkan tanggal dan waktu. Berikut adalah contoh penggunaan QDateEdit:

```
1. from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QDa
   teEdit
2. from PyQt5.QtCore import QDateTime
3.
4.
5. class DateTimeEditExample(QWidget):
6.     def __init__(self):
7.         super().__init__()
8.         self.setWindowTitle('QDateTimeEdit Example')
9.
10.        layout = QVBoxLayout()
11.
12.        datetime_edit = QDateTimeEdit()
13.        datetime_edit.setDateTime(QDateTime.currentDateTime()) # M
   engatur tanggal dan waktu awal ke tanggal dan waktu saat ini
14.        layout.addWidget(datetime_edit)
15.
16.        self.setLayout(layout)
17.
18.
19.if __name__ == "__main__":
20.    app = QApplication([])
21.    window = DateTimeEditExample()
22.    window.show()
23.    app.exec()
|24.
```

9.3 Kegiatan Praktikum

9.3.1 Kegiatan 1 : Data dan Form

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

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVB
   oxLayout, QLabel, QLineEdit, QPushButton
2.
3.
4. class MainWindow(QMainWindow):
5.     def __init__(self):
```

```

6.         super().__init__()
7.         self.setWindowTitle('Main Window')
8.
9.         # Membuat widget utama
10.        widget = QWidget()
11.        self.setCentralWidget(widget)
12.
13.        # Membuat Layout utama
14.        layout = QVBoxLayout()
15.        widget.setLayout(layout)
16.
17.        # Membuat input nama
18.        self.name_input = QLineEdit()
19.        layout.addWidget(self.name_input)
20.
21.        # Membuat tombol kirim
22.        send_button = QPushButton('Kirim')
23.        send_button.clicked.connect(self.send_data)
24.        layout.addWidget(send_button)
25.
26.        # Membuat instance widget
27.        self.widget = Widget()
28.
29.    def send_data(self):
30.        # Mengambil data nama dari input
31.        name = self.name_input.text()
32.        # Mengirim data nama ke widget
33.        self.widget.receive_data(name)
34.        # Menampilkan widget
35.        self.widget.show()
36.
37.
38.class Widget(QWidget):
39.    def __init__(self):
40.        super().__init__()
41.        self.setWindowTitle('Widget')
42.
43.        # Membuat Layout
44.        layout = QVBoxLayout()
45.        self.setLayout(layout)
46.
47.        # Membuat Label untuk menampilkan data nama
48.        self.name_label = QLabel()
49.        layout.addWidget(self.name_label)
50.
51.    def receive_data(self, name):

```

```
52.         # Menampilkan data nama pada Label
53.         self.name_label.setText(f'Halo, {name}!')
54.
55.
56.if __name__ == '__main__':
57.     app = QApplication([])
58.     window = MainWindow()
59.     window.show()
60.     app.exec()
```

4. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

9.3.2 Kegiatan 2 : Studi Kasus Aplikasi Kontak

1. Buat beberapa file berikut dalam satu folder

Nama File : **main.py**

```
1. import sys
2. from PyQt5.QtWidgets import QApplication
3. from contact_app import ContactApp
4.
5. if __name__ == '__main__':
6.     app = QApplication(sys.argv)
7.     window = ContactApp()
8.     window.show()
9.     sys.exit(app.exec_())
```

Nama File : **contact.py**

```
1. class Contact:
2.     def __init__(self, name, phone, email):
3.         self.name = name
4.         self.phone = phone
5.         self.email = email
```

Nama File : **contact_app.py**

```
1. import sys
2. from PyQt5.QtWidgets import QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QMessageBox, QListWidget
3. from PyQt5.QtSql import QSqlDatabase, QSqlQuery
4. from contact import Contact
5.
6. class ContactApp(QMainWindow):
7.     def __init__(self):
8.         super().__init__()
9.         self.setWindowTitle('Daftar Kontak')
10.        self.setGeometry(200, 200, 400, 300)
11.
12.        # Membuat koneksi database SQLite
13.        self.db = QSqlDatabase.addDatabase('QSQLITE')
14.        self.db.setDatabaseName('contact.sqlite')
15.        if not self.db.open():
16.            QMessageBox.critical(self, 'Error', 'Tidak dapat terhubung ke database.')
17.            sys.exit(1)
18.
19.        # Membuat tabel kontak jika belum ada
20.        query = QSqlQuery()
21.        query.exec_('CREATE TABLE IF NOT EXISTS contacts (name TEXT, phone TEXT, email TEXT)')
22.
23.        # Membuat widget utama
24.        widget = QWidget()
25.        self.setCentralWidget(widget)
26.
27.        # Membuat layout utama
28.        layout = QVBoxLayout()
29.        widget.setLayout(layout)
30.
31.        # Membuat Label dan inputan
32.        name_label = QLabel('Nama:')
33.        self.name_input = QLineEdit()
34.        phone_label = QLabel('Nomor Telepon:')
35.        self.phone_input = QLineEdit()
36.        email_label = QLabel('Email:')
37.        self.email_input = QLineEdit()
38.
39.        # Membuat tombol-tombol
40.        button_layout = QHBoxLayout()
```

```

41.         add_button = QPushButton('Tambah')
42.         add_button.clicked.connect(self.add_contact)
43.         update_button = QPushButton('Perbarui')
44.         update_button.clicked.connect(self.update_contact)
45.         delete_button = QPushButton('Hapus')
46.         delete_button.clicked.connect(self.delete_contact)
47.         button_layout.addWidget(add_button)
48.         button_layout.addWidget(update_button)
49.         button_layout.addWidget(delete_button)
50.
51.         # Menambahkan widget ke dalam Layout utama
52.         layout.addWidget(name_label)
53.         layout.addWidget(self.name_input)
54.         layout.addWidget(phone_label)
55.         layout.addWidget(self.phone_input)
56.         layout.addWidget(email_label)
57.         layout.addWidget(self.email_input)
58.         layout.addLayout(button_layout)
59.
60.         # Menambahkan daftar kontak
61.         self.contact_list = QListWidget()
62.         self.contact_list.itemClicked.connect(self.show_contact)
63.         layout.addWidget(self.contact_list)
64.
65.         # Memuat daftar kontak dari database
66.         self.load_contacts()
67.
68.     def load_contacts(self):
69.         # Mengambil daftar kontak dari database dan menampilkannya
70.         query = QSqlQuery()
71.         query.exec_('SELECT name FROM contacts')
72.         self.contact_list.clear()
73.         while query.next():
74.             name = query.value(0)
75.             self.contact_list.addItem(name)
76.
77.     def add_contact(self):
78.         # Menambahkan kontak baru ke database
79.         name = self.name_input.text()
80.         phone = self.phone_input.text()
81.         email = self.email_input.text()
82.         contact = Contact(name, phone, email)
83.
84.         query = QSqlQuery()
85.         query.prepare('INSERT INTO contacts (name, phone, email) V
   ALUES (?, ?, ?)')

```

```

86.         query.bindValue(0, name)
87.         query.bindValue(1, phone)
88.         query.bindValue(2, email)
89.
90.     if query.exec_():
91.         self.load_contacts()
92.         self.clear_inputs()
93.         QMessageBox.information(self, 'Info', 'Kontak berhasil ditambahkan.')
94.     else:
95.         QMessageBox.warning(self, 'Peringatan', 'Gagal menambahkan kontak.')
96.
97. def update_contact(self):
98.     # Mengupdate kontak yang dipilih di database
99.     current_item = self.contact_list.currentItem()
100.    if current_item:
101.        selected_name = current_item.text()
102.        name = self.name_input.text()
103.        phone = self.phone_input.text()
104.        email = self.email_input.text()
105.        contact = Contact(name, phone, email)
106.
107.        query = QSqlQuery()
108.        query.prepare('UPDATE contacts SET name=? , phone=? , email=? WHERE name=?')
109.        query.bindValue(0, name)
110.        query.bindValue(1, phone)
111.        query.bindValue(2, email)
112.        query.bindValue(3, selected_name)
113.
114.    if query.exec_():
115.        self.load_contacts()
116.        self.clear_inputs()
117.        QMessageBox.information(self, 'Info', 'Kontak berhasil diperbarui.')
118.    else:
119.        QMessageBox.warning(self, 'Peringatan', 'Gagal memperbarui kontak.')
120.    else:
121.        QMessageBox.warning(self, 'Peringatan', 'Pilih kontak terlebih dahulu.')
122.
123. def delete_contact(self):
124.     # Menghapus kontak yang dipilih dari database
125.     current_item = self.contact_list.currentItem()

```

```

126.         if current_item:
127.             selected_name = current_item.text()
128.             confirm = QMessageBox.question(self, 'Konfirmasi',
129.                 f'Anda yakin ingin menghapus kontak {selected_name}?', QMessageBox
130.                 .Yes | QMessageBox.No)
131.             if confirm == QMessageBox.Yes:
132.                 query = QSqlQuery()
133.                 query.prepare('DELETE FROM contacts WHERE name=
134. ?')
135.                 query.bindValue(0, selected_name)
136.             if query.exec_():
137.                 self.load_contacts()
138.                 self.clear_inputs()
139.                 QMessageBox.information(self, 'Info', 'Kont
140. ak berhasil dihapus.')
141.             else:
142.                 QMessageBox.warning(self, 'Peringatan', 'Ga
143. gal menghapus kontak.')
144.             else:
145.                 QMessageBox.warning(self, 'Peringatan', 'Pilih kont
146. ak terlebih dahulu.')
147. 
148.     def show_contact(self, item):
149.         # Menampilkan informasi kontak saat diklik
150.         selected_name = item.text()
151.         query = QSqlQuery()
152.         query.prepare('SELECT * FROM contacts WHERE name=?')
153.         query.bindValue(0, selected_name)
154. 
155.     def clear_inputs(self):
156.         # Mengosongkan inputan
157.         self.name_input.clear()
158.         self.phone_input.clear()
159.         self.email_input.clear()
160.

```

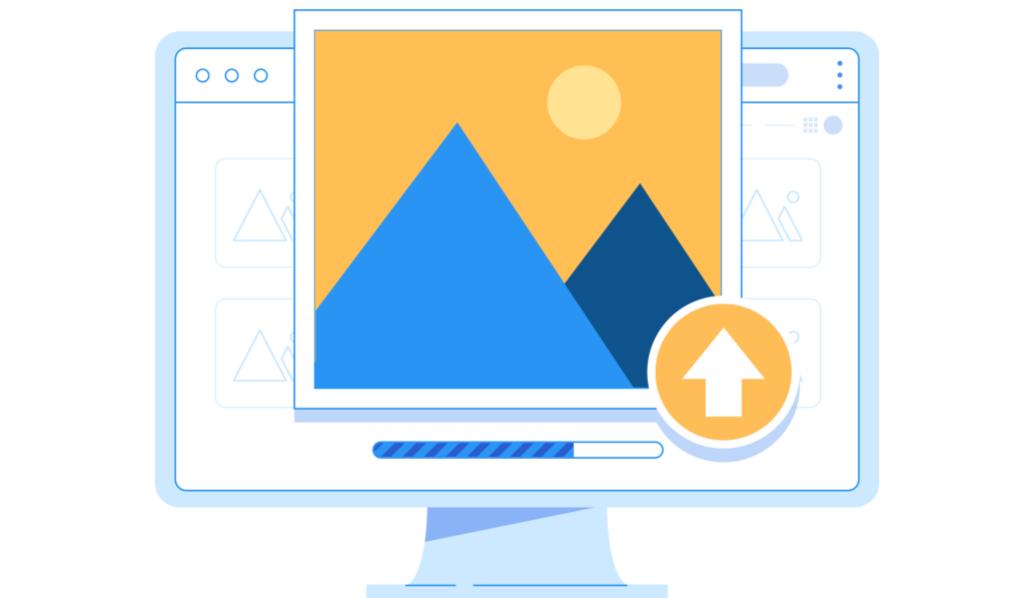
2. Klik Run pada main.py untuk menjalankan aplikasi ini

9.4 Tugas

Berdasarkan pada Praktikum Kegiatan 3, Identifikasi dan jelaskan widget, modul PyQt5 class yang digunakan dalam membangun aplikasi daftar kontak

BAB 10

PyQt Designer



10.1 Tujuan

20. Dapat memahami pembuatan aplikasi GUI menggunakan PyQt Designer
21. Dapat melakukan pemaketan aplikasi PyQt Designer dengan Python

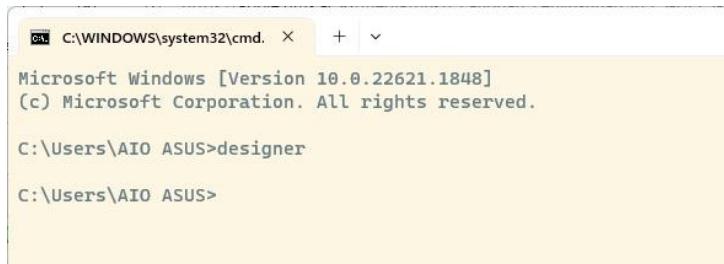
10.2 Pengantar

10.2.1 Pengenalan Qt Designer

Qt Designer adalah tool GUI yang digunakan untuk merancang dan membangun antarmuka pengguna dari aplikasi Qt. Dengan Qt Designer kita dapat merancang tata letak widget secara visual dan juga membuat skrip untuk widget tersebut. Qt Designer sangat berguna bagi mereka yang ingin merancang antarmuka pengguna tanpa harus menulis kode untuk itu.

10.2.2 Instalasi Qt Designer

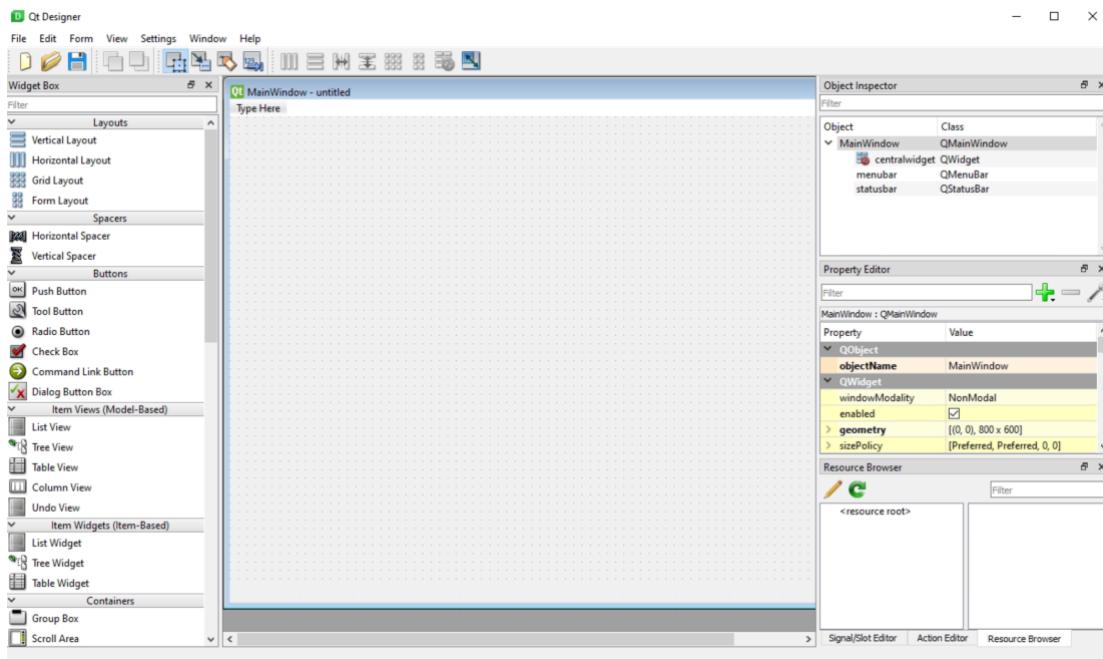
Qt Designer secara otomatis akan terinstall ketika kita melakukan instalasi PyQt5. Jika telah menginstal PyQt5 melalui pip, maka seharusnya Qt Designer juga sudah terinstall pada sistem. Untuk menjalankan Qt Designer, ketikkan perintah designer pada Command Prompt.



```
C:\WINDOWS\system32\cmd. + 
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\AIO ASUS>designer
C:\Users\AIO ASUS>
```

Tampilan dari PyQt Designer dapat dilihat pada gambar berikut



- Widget Box: Daftar semua widget yang dapat kita gunakan dalam desain Anda.
- Object Inspector: Menampilkan hierarki widget dalam desain.
- Property Editor: Menampilkan properti dari widget yang dipilih.
- Signal & Slot Editor: Memungkinkan kita untuk mengedit sinyal dan slot dari widget.
- Form Editor: Area kerja utama di mana kita merancang tampilan aplikasi.

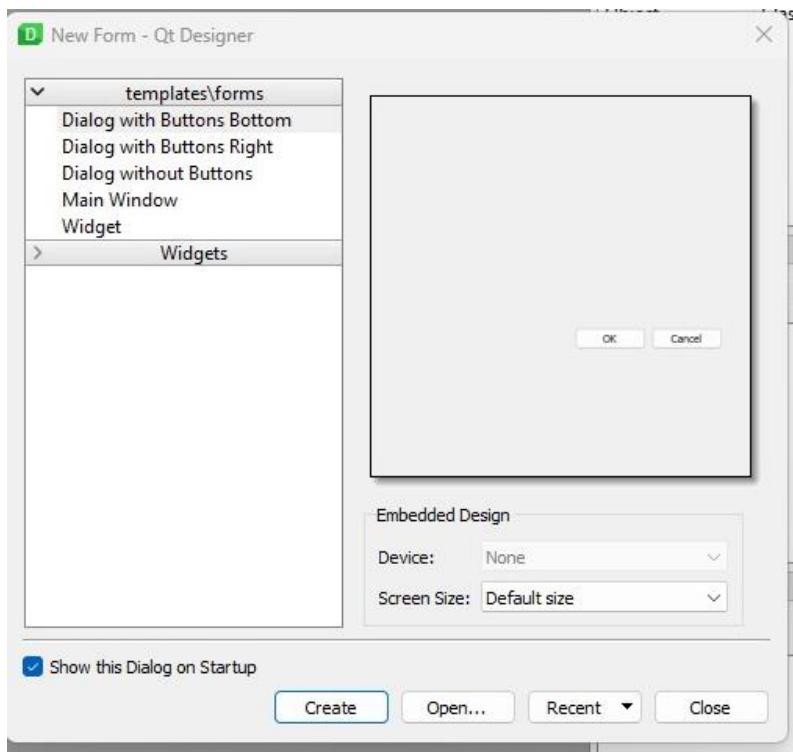
Jika muncul error, maka kita harus melakukan instalasi Qt Designer dengan menggunakan perintah

> pip install PyQt5Designer

```
C:\Users\AIO ASUS>pip install PyQt5Designer
Collecting PyQt5Designer
  Downloading PyQt5Designer-5.14.1-py3-none-win_amd64.whl (40.8 MB)
  40.8/40.8 MB 9.5 MB/s eta 0:00:00
Installing collected packages: PyQt5Designer
Successfully installed PyQt5Designer-5.14.1
```

10.2.3 Membuat Form dengan Qt Designer

Untuk membuat form baru, pilih File > New. kita akan diminta untuk memilih jenis form yang ingin dibuat.



Template form yang tersedia antara lain :

- Dialog with Buttons Bottom: template untuk dialog yang memiliki baris tombol di bagian bawah. Tombol ini biasanya digunakan untuk menerima atau membatalkan operasi.
- Dialog with Buttons Right: template untuk dialog yang memiliki kolom tombol di sisi kanan.
- Dialog without Buttons: template untuk dialog yang tidak memiliki tombol. Kita bebas menambahkan widget lain ke dalam dialog ini.
- Main Window: template untuk jendela utama aplikasi. Jendela utama biasanya memiliki menu, toolbar, dan area konten di mana widget lain dapat ditempatkan.

- Widget: template untuk widget kustom. Widget adalah elemen dasar antarmuka pengguna dalam Qt dan dapat digabungkan untuk membuat antarmuka yang lebih kompleks.

Setelah kita memilih jenis form, kita akan dibawa ke Form Editor. Di sini, kita dapat mulai menambahkan widget ke form dari Widget Box.

10.2.4 Mengubah UI menjadi Kode Python

Jika kita menyimpan file dari PyQt Designer, maka format yang diberikan adalah .ui . Untuk itu kita perlu mengubah format tersebut ke dalam kode python. Kita dapat melakukan ini dengan menggunakan tool pyuic yang disertakan dengan PyQt5. Sebagai contoh jika memiliki file ui dengan nama mainwindow.ui. Aktifkan command prompt, Jalankan perintah berikut di terminal Anda:

```
1. pyuic5 -x mainwindow.ui -o mainwindow.py
```

Setelah kita menjalankan perintah ini, kita akan mendapatkan file Python dengan nama mainwindow.py yang berisi kode untuk form. Namun kode tersebut belum dapat langsung kita jalankan, kita perlu membuat kode file baru yang berfungsi mengimpor file tersebut. Buat file baru beri nama dengan demo.py

```
1. from PyQt5.QtWidgets import QApplication
2. from mainwindow import Ui_MainWindow
3.
4. if __name__ == "__main__":
5.     app = QApplication([])
6.     window = Ui_MainWindow()
7.     window.show()
8.     app.exec_()
```

Dalam kode di atas, Ui_MainWindow adalah kelas yang dihasilkan oleh pyuic5. Perhatikan bahwa nama mainwindow pada baris 2 harus sesuai dengan nama file

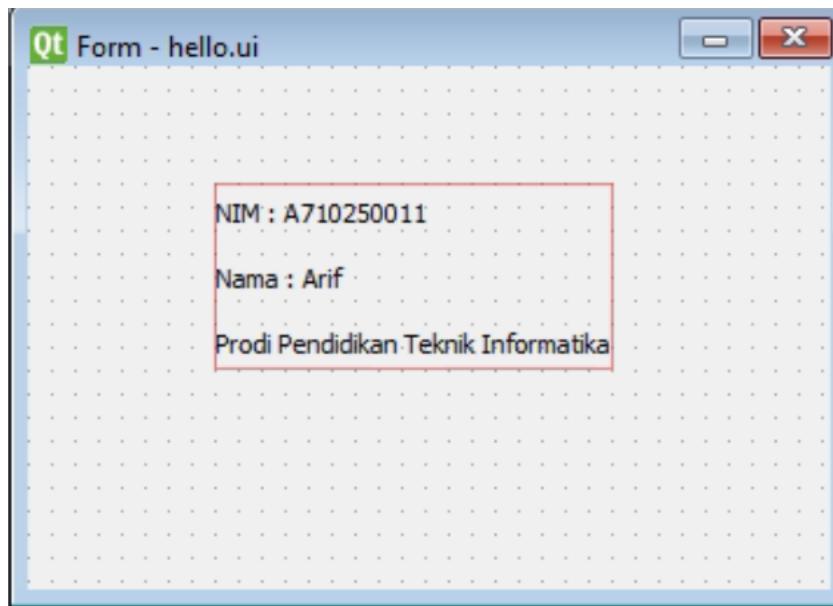
Python yang dihasilkan oleh pyuic5. Untuk menjalankannya ketik perintah berikut pada command prompt

```
1. python demo.py
```

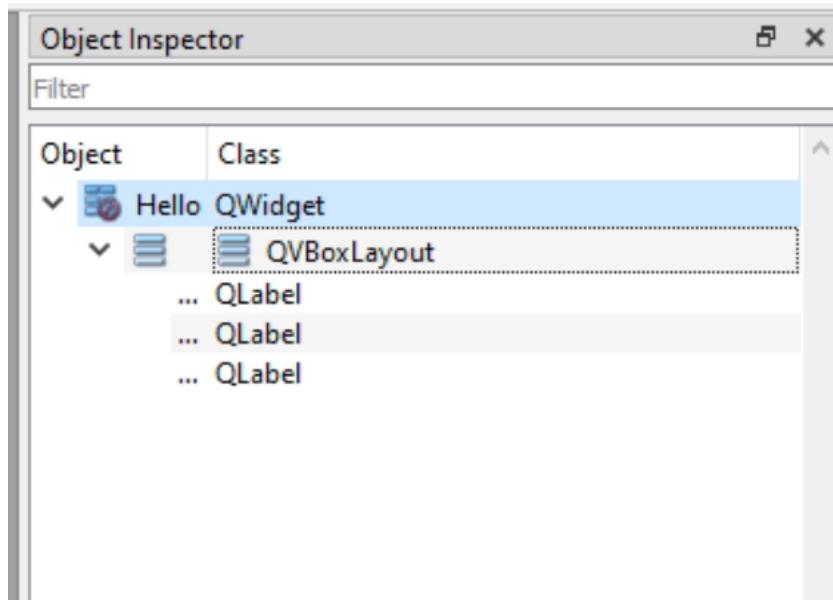
10.3 Kegiatan Praktikum

10.3.1 Kegiatan 1 : Membuat Aplikasi Pertama

5. Buat rancangan layout berikut menggunakan Qt Designer, Pilih Form dengan tipe **Widget**. simpan dengan nama hello.ui.



6. Perhatikan bahwa rancangan UI tersebut menggunakan Vertical Layout yang didalamnya terdapat 3 QLabel



7. Buka command prompt pada folder yang sama dengan file hello.ui kemudian jalankan perintah berikut

```
1. pyuic5 hello.ui -o hello.py
```

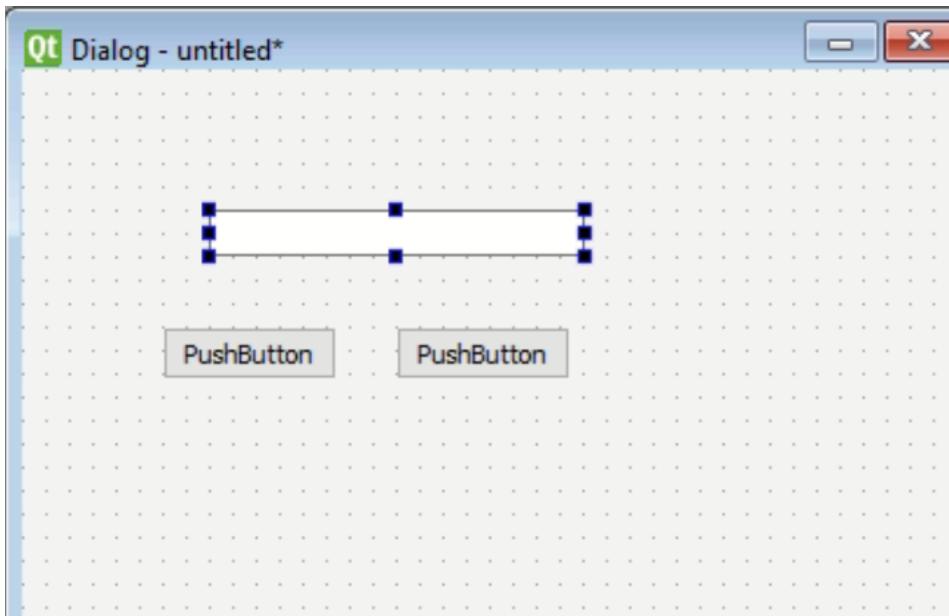
8. Buat file baru dengan nama mainhello.py

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. from hello import Ui_Hello
3.
4. class HelloWindow(QMainWindow, Ui_Hello):
5.     def __init__(self, parent=None):
6.         super(HelloWindow, self).__init__(parent)
7.         self.setupUi(self)
8.
9. app = QApplication([])
10.window = HelloWindow()
11.window.show()
12.app.exec_()
```

9. Jalankan aplikasi dengan mengetikkan perintah “ python mainhello.py ” pada command prompt

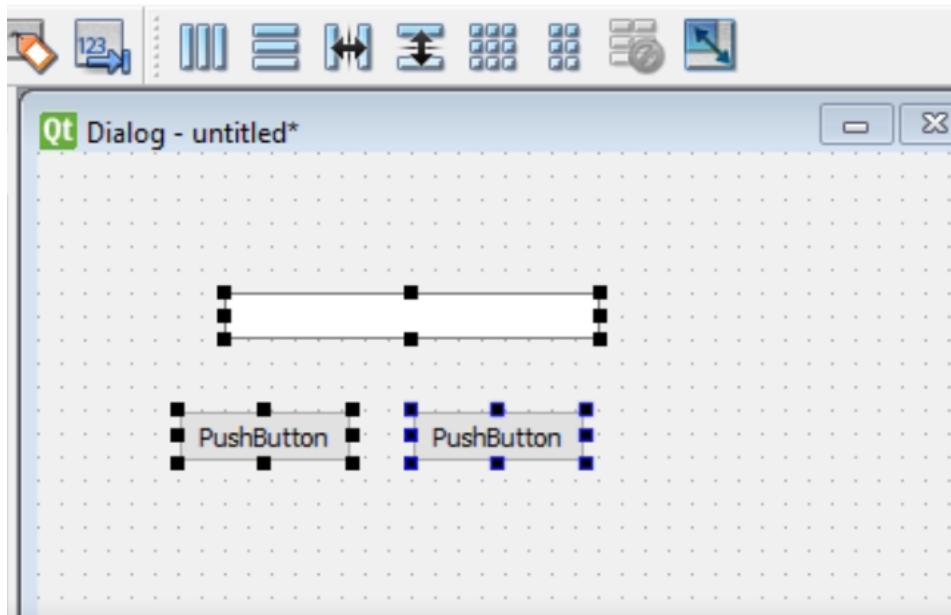
10.3.2 Kegiatan 2 : Signal dan Slot pada PyQt Designer

1. Buat rancangan layout berikut menggunakan Qt Designer, Pilih Form dengan tipe **Dialog Without Button**. simpan dengan nama signalslot.ui. Perhatikan bahwa peletakan QLineEdit dan QPushButton masih berantakan.

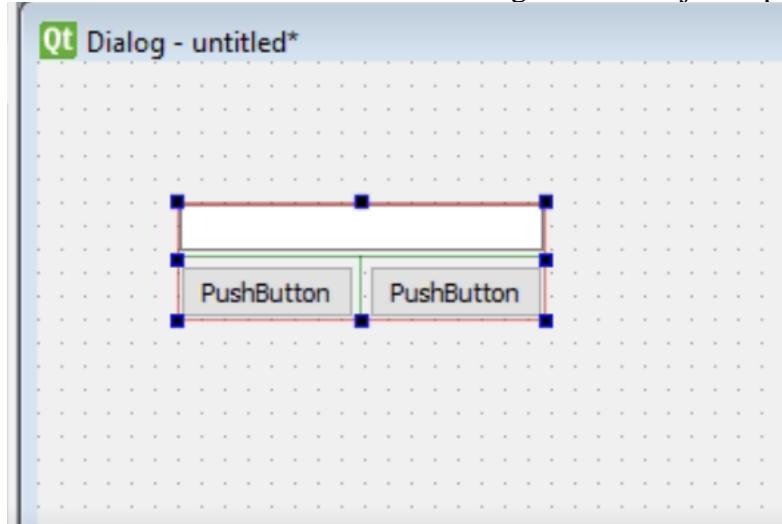


2. Seleksi semua komponen dengan cara tahan Ctrl + Klik QPushButton dan QLineEdit, kemudian pilih logo Layout in Grid

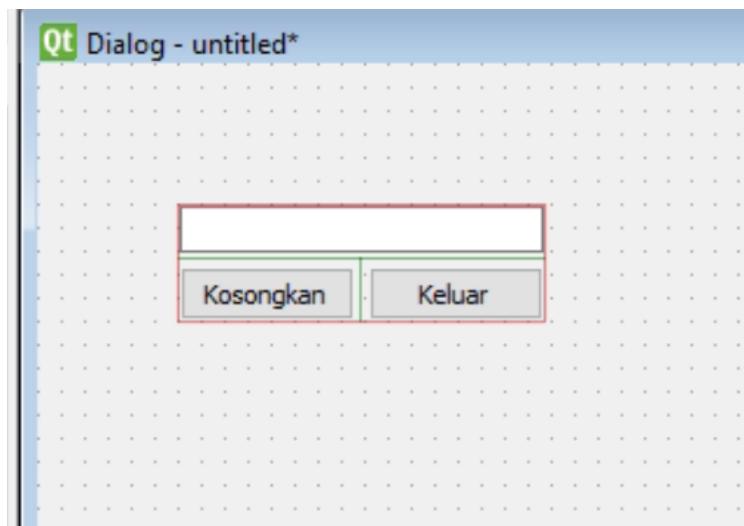




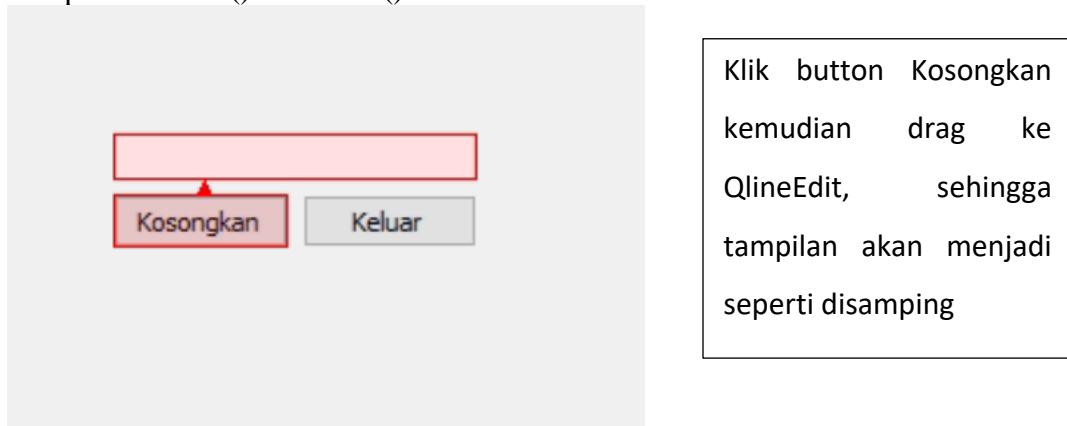
3. Secara otomatis maka susunan widget akan menjadi rapi menyesuaikan layout

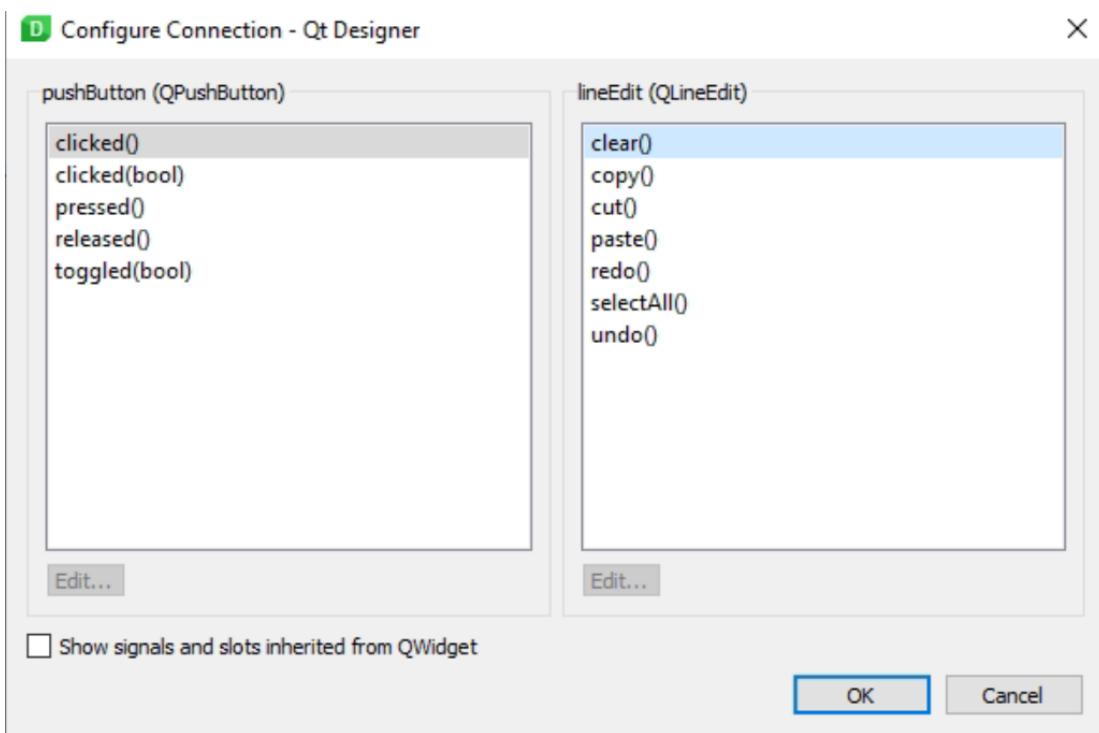


4. Ubah nama PushButton 1 dan 2 menjadi **Kosongkan** dan **Keluar**

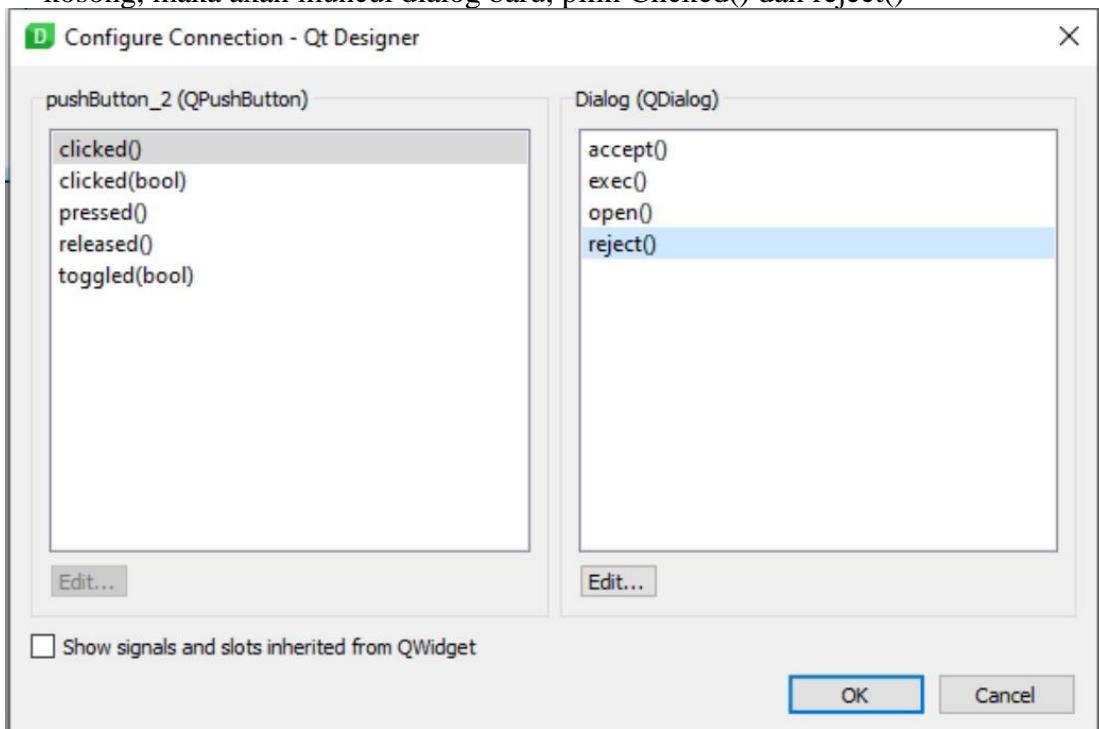


5. Masuk ke Mode Edit Signal/Slot dengan klik logo  pada toolbar. Kemudian klik button **Kosongkan** dan drag ke QLineEdit, maka akan muncul dialog baru, pilih Clicked() dan Clear()

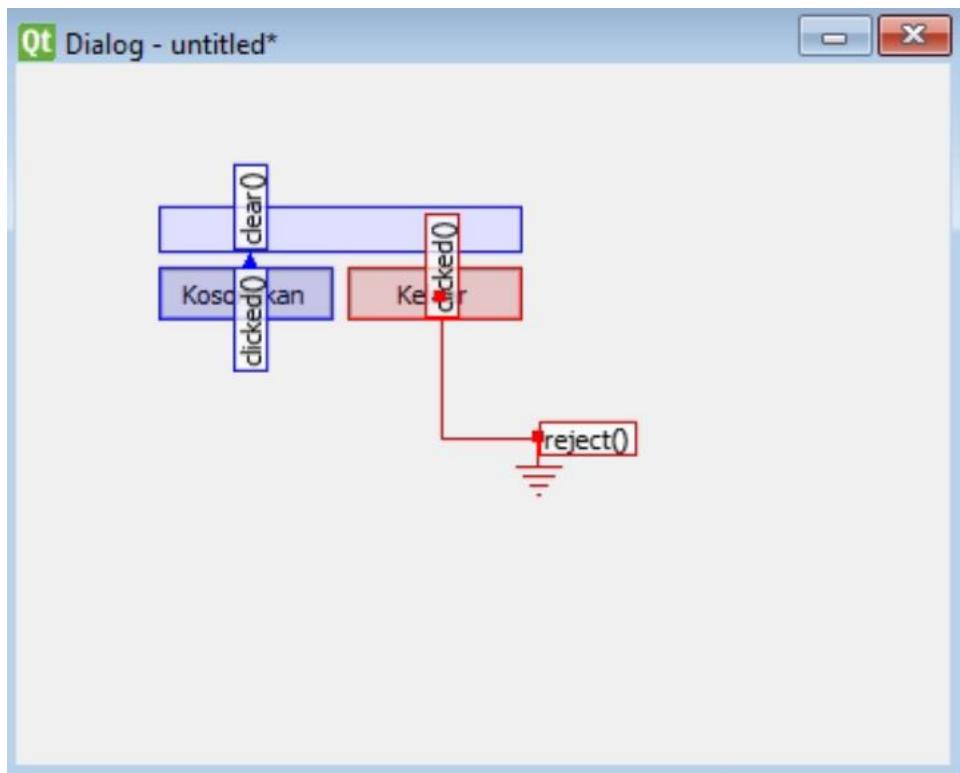




6. Lakukan hal yang sama untuk klik button **Keluar** dan namun drag ke form kosong, maka akan muncul dialog baru, pilih Clicked() dan reject()



7. Tampilan Form akan menjadi seperti berikut



10. Buka command prompt pada folder yang sama dengan file signalslot.ui kemudian jalankan perintah berikut

```
|pyuic5 signalslot.ui -o signalslot.py
```

11. Buat file baru dengan nama mainsignalslot.py

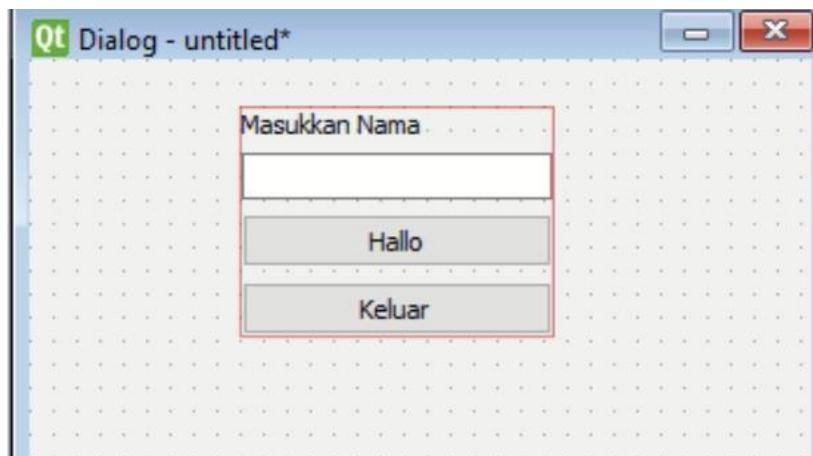
```
1. from PyQt5.QtWidgets import QApplication, QDialog
2. from signalslot import Ui_Dialog
3.
4. class DialogWindow(QDialog, Ui_Dialog):
5.     def __init__(self, parent=None):
6.         super(DialogWindow, self).__init__(parent)
7.         self.setupUi(self)
8.
9. app = QApplication([])
10.window = DialogWindow()
11.window.show()
```

```
|12.app.exec_()
```

12. Jalankan aplikasi dengan mengetikkan perintah “ python mainsignalslot.py ” pada command prompt

10.3.3 Kegiatan 3 : Memproses Input

1. Buat rancangan layout berikut menggunakan Qt Designer, Pilih Form dengan tipe **Dialog Without Button**. simpan dengan nama form.ui. Beri nama object QLineEdit menjadi nameLineEdit dan QPushButton “Halo” menjadi showMessageButton.



2. Buka command prompt pada folder yang sama dengan file form.ui kemudian jalankan perintah berikut

```
| pyuic5 form.ui -o form.py
```

3. Buat file baru dengan nama mainform.py

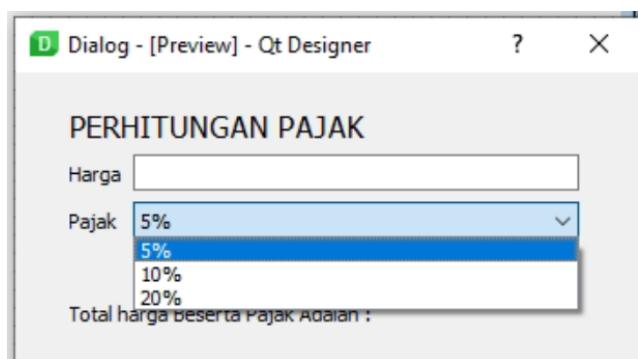
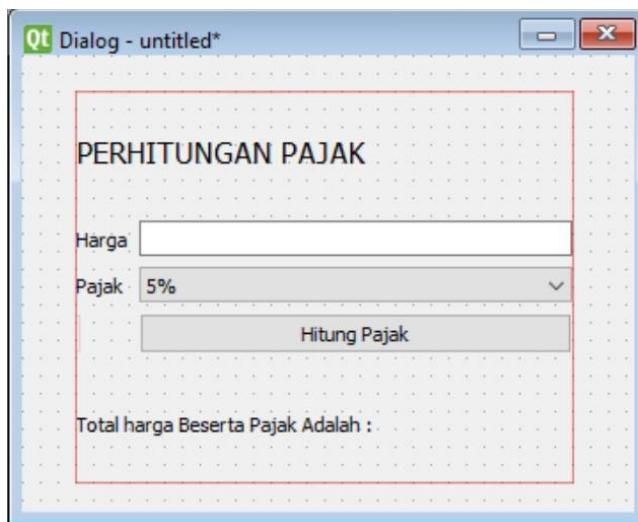
```
|1. from PyQt5.QtWidgets import QApplication, QDialog, QMessageBox  
2. from form import Ui_Dialog  
3.  
4. class Dialog(QDialog, Ui_Dialog):
```

```
5.     def __init__(self, parent=None):
6.         super(Dialog, self).__init__(parent)
7.         self.setupUi(self)
8.         self.showMessageButton.clicked.connect(self.show_message)
9.
10.    def show_message(self):
11.        name = self.nameLineEdit.text()
12.        QMessageBox.information(self, "Hello", f"Hello, {name}!")
13.
14. app = QApplication([])
15. dialog = Dialog()
16. dialog.show()
17. app.exec_()
```

4. Jalankan aplikasi dengan mengetikkan perintah “ python mainform.py ” pada command prompt

10.4 Tugas

1. Dengan menggunakan PyQt Designer, rancanglah form dengan tampilan berikut. Terdapat 2 input utama, input harga dengan nominal rupiah dan input pajak dengan nilai 5%, 10% dan 15%. Jika user melakukan klik pada tombol **Hitung Pajak** maka akan menampilkan Total harga beserta pajak yang harus dibayar



DAFTAR PUSTAKA

- Hunt, J. (2019). *A Beginners Guide to Python 3 Programming*. In Springer
- Romano, Fabrizio. (2015). *Learning Python*. Packt Publishing.
- Swastika, W. (2019). *Pengantar Algoritma dan Penerapannya pada Python*. Ma Chung Press.
- Wadi, H. *Pemrograman Python untuk Mahasiswa dan Pelajar*. TR Publisher
- Python File Handling Tutorial: How to Create, Open, Read, Write, Append*. (n.d.). Retrieved September 7, 2020, from
<https://www.softwaretestinghelp.com/python/python-file-reading-writing/>
- Learn Python Programming*. (n.d.). Retrieved September 7, 2020, from
<https://www.programiz.com/python-programming>
- Tutorial Pemrograman Python*. (n.d.). Retrieved September 5 , 2020, from
<https://www.petanikode.com/tutorial/python/>
- PY4E - Python for Everybody*. (n.d.). Retrieved September 2, 2020, from
<https://www.py4e.com/>
- Google's Python Class / Python Education / Google Developers*. (n.d.). Retrieved September 9, 2020, from <https://developers.google.com/edu/python/>
- Learn Python the Hard Way*. (n.d.). Retrieved September 2, 2020, from
<https://learnpythonthehardway.org/book/>
- Python Programming Tutorials*. (n.d.). Retrieved September 9, 2020, from
<https://pythonprogramming.net/python-fundamental-tutorials/>

LAMPIRAN

Laporan Sementara

Laporan sementara dilakukan dengan langkah sebagai berikut:

1. Buatlah screenshot atau gambar dari hasil kegiatan yang dilakukan.
2. Ganti nama file dengan format: bab-kegiatan-no.gambar-nim.jpg Misalnya file adalah gambar ke 1 pada Bab 1, Kegiatan 1, dan nim anda adalah A40009001, maka nama filenya adalah **1-1-1-A40009001.jpg**
3. Simpan file tersebut kemudian tempatkan pada folder sesuai instruksi dosen/asisten praktikum.

Laporan Praktikum

Laporan ditulis dalam kertas putih ukuran A4. Sedangkan urutan susunan laporan adalah sebagai berikut:

1. Cover depan: Berwarna sama dengan cover modul praktikum
2. Halaman Cover:
3. Kata Pengantar
4. Daftar isi
5. Laporan tiap modul (1-10) sesuai dengan format terlampir
6. Penulis: berisi biodata penulis (disertai foto), pesan dan kesan, kritik dan saran demi kemajuan praktikum berikutnya.

Format Laporan Tiap Bab

Mata Kuliah	:	Pemrograman Berorientasi Objek	Acc
NIM	:		
Nama	:		
Tgl Prakt.	:		Tgl

BAB I Judul

1. Dasar Teori

300 sampai dengan 350 kata

2. Tujuan

3. Analisa Hasil

3.1. Kegiatan 1: ...

Tampilkan hasil praktikum berupa kode yang dibuat atau hasil output, kemudian berikan analisisnya. Jika terdapat gambar, berikan juga nomor gambar.

3.2. Kegiatan 2: ...

4. Penyelesaian Tugas

Jika terdapat tugas yang dikerjakan, tuliskan disini langkah penggerjaan dan hasilnya.

5. Kesimpulan

Berikan kesimpulan yang didapatkan setelah anda menyelesaikan praktikum