LAPORAN PRAKTIKUM PEMOGRAMAN BERORIERNTASI OBJEK (PBO) PRAKTIKUM 3

2411102441207

RIVOLDY ADRIAN PRAWIRA MAKATITA

FAKULTAS SAINS DAN TEKNOLOGI

PROGRAM STUDI S1 TEKNIK INFORMATIKA

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

2411102441207@umkt.ac.id

BAB I
PENDAHULUAN

1.1 Latar Belakang

Enkapsulasi adalah konsep fundamental dalam pemrograman berorientasi objek (OOP) yang berperan penting dalam menjaga integritas data dan keamanan perangkat lunak. Pada dasarnya, enkapsulasi mengatur cara data dan fungsi-fungsi yang mengakses data tersebut dibungkus bersama dalam sebuah objek, dengan tujuan utama menyembunyikan detail implementasi dari pengguna objek tersebut. Pendekatan ini memungkinkan pengembang untuk membatasi akses terhadap atribut dan metode tertentu agar hanya bisa diakses atau diubah melalui mekanisme yang telah ditetapkan, yakni getter dan setter.

Dalam pengembangan perangkat lunak, sering ditemukan masalah terkait akses langsung ke data internal objek yang dapat menyebabkan inkonsistensi data, kesulitan dalam pemeliharaan kode, dan rawan terjadi kesalahan baik sengaja maupun tidak sengaja. Ketidaksesuaian ini muncul ketika atribut kelas dapat diakses secara bebas dari luar kelas tanpa kontrol yang memadai. Oleh karena itu, praktikum ini bertujuan untuk mengeksplorasi bagaimana enkapsulasi dapat diterapkan untuk menyembunyikan informasi dan melindungi data dari manipulasi yang tidak diinginkan.

Menurut Fowler (2004), enkapsulasi bukan hanya sekadar membungkus data, melainkan juga mekanisme pengendalian akses yang krusial dalam OOP untuk mengurangi kompleksitas sistem dan meningkatkan keamanan data. Selain itu, literatur dari Liskov dan Guttag (2001) menekankan bahwa enkapsulasi memungkinkan modularitas kode yang lebih baik dan pemeliharaan yang lebih efisien. Studi lain oleh Sommerville (2011) menunjukkan bahwa penggunaan enkapsulasi mengurangi risiko bug terkait modifikasi data secara tidak terkendali dalam pengembangan perangkat lunak berskala besar.

Melalui praktikum ini, peserta dapat memahami penerapan teknik enkapsulasi secara langsung dengan menerapkan prinsip penyembunyian informasi dan proteksi data dalam pembuatan kelas dan objek. Hal ini penting karena pengetahuan dan keterampilan tersebut memungkinkan pengembangan solusi perangkat lunak yang lebih robust, aman, dan mudah dirawat. Selain itu, pemahaman yang baik mengenai enkapsulasi menjadi dasar bagi konsep-konsep lanjutan dalam OOP seperti inheritance dan polymorphism, sehingga praktikum ini

membantu membangun fondasi yang kuat bagi pembelajaran pemrograman yang lebih mendalam.

1.2 Tujuan

- Mampu menjelaskan konsep enkapsulasi dalam pemrograman berorientasi objek secara rinci dan teknis, sehingga peserta memahami prinsip dasar penyembunyian informasi.
- Mampu menerapkan konsep enkapsulasi dengan membuat kelas yang menggunakan atribut privat dan publik sesuai kaidah OOP.
- Mampu membuat dan mengimplementasikan metode getter dan setter untuk mengakses dan memodifikasi atribut privat secara terkontrol.
- Mengukur keberhasilan praktikum berdasarkan kemampuan peserta dalam menulis kode yang benar dan menjalankan program dengan fungsionalitas enkapsulasi yang sesuai.
- Menjamin relevansi tujuan dengan penyelesaian masalah akses data yang tidak terkendali dalam pengembangan perangkat lunak melalui penerapan enkapsulasi.
- Memastikan tujuan praktikum realistis dan dapat dicapai mengingat sumber daya seperti perangkat lunak pemrograman, waktu praktikum, dan bimbingan instruktur tersedia secara memadai.

1.3 Tinjauan Pustaka

Enkapsulasi merupakan salah satu pilar utama dalam pemrograman berorientasi objek (OOP) yang berfokus pada pengemasan data dan fungsi-fungsi terkait dalam satu kesatuan objek serta penyembunyian detail implementasi dari pengguna objek tersebut. Konsep ini dikenal juga dengan istilah "information hiding" yang bertujuan untuk menjaga keamanan dan integritas data dengan membatasi akses langsung ke atribut melalui mekanisme kontrol seperti getter dan setter (Fowler, 2004).

Dalam studi-studi terdahulu, Fowler (2004) menegaskan bahwa enkapsulasi tidak hanya menyembunyikan data, tetapi juga menyediakan interface yang terdefinisi dengan baik untuk interaksi dengan objek, sehingga memudahkan pemeliharaan dan pengembangan perangkat lunak. Pendekatan ini memperkecil risiko kerusakan data akibat akses tidak sah atau modifikasi yang tidak terkontrol. Selain itu, Liskov dan Guttag (2001) menyoroti peran enkapsulasi dalam

modularitas dan keamanan kode, memperjelas batas tanggung jawab objek yang terenkapsulasi.

Analisis perbandingan menunjukkan bahwa semua literatur setuju bahwa enkapsulasi meningkatkan keamanan dan keterbacaan kode, sekaligus mengurangi efek samping perubahan data yang tidak diinginkan dalam sistem yang kompleks. Namun, beberapa studi juga mengidentifikasi tantangan dalam implementasi enkapsulasi yang optimal, terutama pada bahasa pemrograman yang kurang mendukung aturan akses data secara ketat (Sommerville, 2011).

Penelitian dan kajian ini menjadi dasar penting bagi praktikum yang bertujuan memberikan pemahaman langsung dan keterampilan penerapan enkapsulasi dalam OOP, sehingga peserta dapat mengatasi masalah akses data yang tidak terkontrol serta meningkatkan kualitas pengembangan perangkat lunak yang lebih aman dan terstruktur. Dengan demikian, tinjauan pustaka ini mendukung perumusan masalah dan menegaskan pentingnya praktikum dalam menguji konsep enkapsulasi secara aplikatif (Fowler, 2004; Liskov & Guttag, 2001; Sommerville, 2011).

BAB II
ALAT DAN BAHAN

2.1 Alat

a. Komputer atau Laptop
b. Koneksi Internet
c. Python
d. Visual Studio Code

2.2 Bahan

a. Modul Praktikum

BAB III
PROSEDUR KERJA

3.1 Membuat File

1. Buat file baru bernama pertemuan_3.py.

3.2 Mengenali Masalahnya (Kode yang Tidak Aman)

1. Ketik kode berikut. Ini adalah ***class*** User yang sangat sederhana dan rentan.

```python
class User:
    def __init__(self, username, level):
        self.username = username
        self.level = level

    def info(self):
        print(f"Username: {self.username}, Level: {self.level}")

user_1 = User("admin_ganteng", "super Admin")
user_1.info()

print("\n[Merusak data dari luar class]")
user_1.level = 12345
user_1.username = ""
user_1.info()
```

2. Jalankan kode di atas. Perhatikan betapa mudahnya kita merusak data di dalam objek user_1

3.3 Menerapkan Atribut Privat

1. Ubah *class* User menjadi seperti ini. Perhatikan penambahan __ pada nama atribut.

```python
class User:
    def __init__(self, username, level):
        self.__username = username
        self.__level = level

    def info(self):
        print(f"Username: {self.__username}, Level: {self.__level}")

user_1 = User("admin_ganteng", "super Admin")
user_1.info()

try:
    print(user_1.__username)
except AttributeError as e:
    print(f"\nError: {e}")
    print("Attribut __username tidak bisa diakses langsung")
```

2. Jalankan kode tersebut. Anda akan mendapatkan AttributeError. Ini membuktikan bahwa atribut dengan _ _ tidak bisa diakses secara langsung dari luar *class*. Data kita sekarang aman dari sentuhan langsung! Tapi... bagaimana cara kita membacanya?

3.4 Membuat Getter untuk Membaca Data

1. Tambahkan ***method getter*** ke dalam *class* User.

```python
class User:
    def __init__(self, username, level):
        self.__username = username
        self.__level = level

    def info(self):
        print(f"Username: {self.__username}, Level: {self.__level}")

    def get_username(self):
        return self.__username

    def get_level(self):
        return self.__level

user_1 = User("admin_ganteng", "super Admin")

print("\n--- Mengakses data via Getter ---")
nama_user = user_1.get_username()
level_user = user_1.get_level()
print(f"Username dari getter: {nama_user}")
print(f"Level dari getter: {level_user}")
```

2. Jalankan kodenya. Sekarang kita bisa membaca data tanpa bisa mengubahnya.

3.5 Membuat Setter dengan Validasi untuk Mengubah Data

1. Lengkapi *class* User dengan ***method setter***.

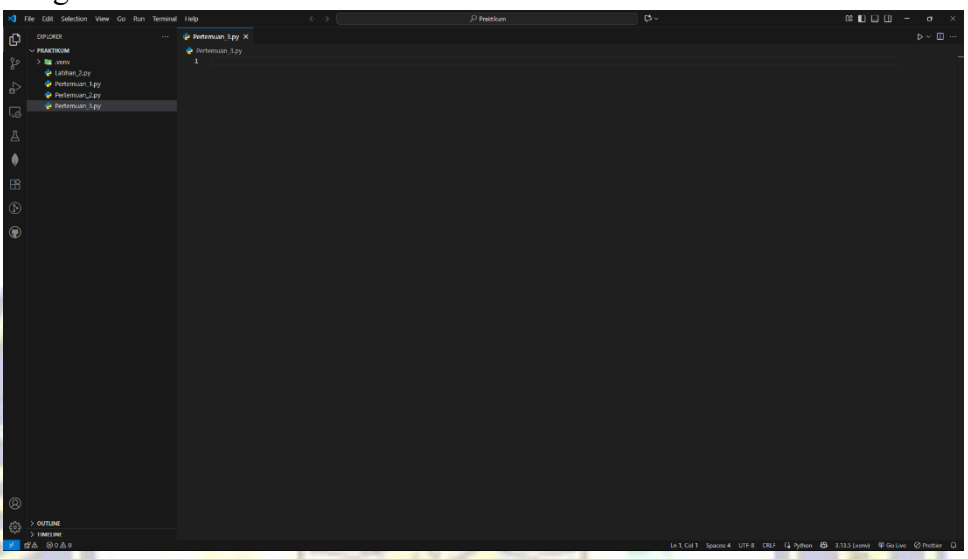
```python
class User:
    def __init__(self, username, level):
        self.__username = username
        self.__level = level
        self.set_username(username)
        self.set_level(level)

    def info(self):
        print(f"Username: {self.__username}, Level: {self.__level}")

    def get_username(self):
        return self.__username

    def get_level(self):
        return self.__level

    def set_username(self, username_baru):
        if len(username_baru) > 5:
            self.__username = username_baru
        else:
            print("Username terlalu pendek! Minimal 6 karakter")

    def set_level(self, level_baru):
        allowed_level = ["User", "Admin", "Super Admin"]
        if level_baru in allowed_level:
            self.__level = level_baru
            print("Level berhasil diubah.")
        else:
            print(f"Error: Level '{level_baru}' tidak valid!")

user_1 = User("pengguna_baru", "User")
user_1.info()

print("\n--- Mencoba mengubah data via Setter ---")
user_1.set_username("admin")
user_1.set_level("Moderator")
user_1.info()

print("\n--- Mencoba lagi dengan data yang valid ---")
user_1.set_username("administrator_sistem")
user_1.set_level("Admin")
user_1.info()
```

2. Jalankan dan amati outputnya. Perhatikan bagaimana *setter* menolak data yang tidak valid dan hanya menerima data yang sesuai dengan aturan. Objek kita sekarang jauh lebih kuat dan dapat diandalkan.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil

- Langkah 3.1



File Python dengan nama file Pertemuan_3.py

- Langkah 3.2

```
PS E:\University's Stuff\3rd\Pemrograman Berorientasi Objek\Pr
aktikum> & "E:/University's Stuff/3rd/Pemrograman Berorientasi
 Objek/Praktikum/.venv/Scripts/python.exe" "e:/University's St
uff/3rd/Pemrograman Berorientasi Objek/Praktikum/Pertemuan_3.p
y"
Username: admin_ganteng, Level: super Admin

[Merusak data dari luar class]
Username: , Level: 12345
```

Output

- Langkah 3.3

```
PS E:\University's Stuff\3rd\Pemrograman Berorientasi Objek\Pr
aktikum> & "E:/University's Stuff/3rd/Pemrograman Berorientasi
 Objek/Praktikum/.venv/Scripts/python.exe" "e:/University's St
uff/3rd/Pemrograman Berorientasi Objek/Praktikum/Pertemuan_3.p
y"
Username: admin_ganteng, Level: super Admin

Error: 'User' object has no attribute '__username'
Attribut __username tidak bisa diakses langsung
```

Output

- Langkah 3.4

```
PS E:\University's Stuff\3rd\Pemrograman Berorientasi Objek\Pr
aktikum> & "E:/University's Stuff/3rd/Pemrograman Berorientasi
 Objek/Praktikum/.venv/Scripts/python.exe" "e:/University's St
uff/3rd/Pemrograman Berorientasi Objek/Praktikum/Pertemuan_3.p
y"

--- Mengakses data via Getter ---
Username dari getter: admin_ganteng
Level dari getter: super Admin
```

Output

- Langkah 3.5

```
PS E:\University's Stuff\3rd\Pemrograman Berorientasi Objek\Pr
aktikum> & "E:/University's Stuff/3rd/Pemrograman Berorientasi
 Objek/Praktikum/.venv/Scripts/python.exe" "e:/University's St
uff/3rd/Pemrograman Berorientasi Objek/Praktikum/Pertemuan_3.p
y"
Level berhasil diubah.
Username: pengguna_baru, Level: User

--- Mencoba mengubah data via Setter ---
Username terlalu pendek! Minimal 6 karakter
Error: Level 'Moderator' tidak valid!
Username: pengguna_baru, Level: User

--- Mencoba lagi dengan data yang valid ---
Level berhasil diubah.
Username: administrator_sistem, Level: Admin
```

Output

4.2 Pembahasan

Praktikum ini membahas penerapan konsep enkapsulasi dalam paradigma pemrograman berorientasi objek (OOP) menggunakan bahasa Python. Enkapsulasi

berfungsi sebagai mekanisme pembatasan akses ke atribut dan method suatu objek dengan tujuan utama menjaga integritas dan keamanan data internal objek tersebut. Pada Python, enkapsulasi dicapai melalui mekanisme access modifier yang terdiri atas tiga tingkat akses: public, protected, dan private.

Atribut atau metode dengan akses public dapat diakses langsung dari luar kelas tanpa batasan. Namun hal ini berpotensi menimbulkan masalah karena atribut dapat diubah secara sembarangan sehingga menyebabkan inkonsistensi data. Untuk mengatasinya, atribut harus dideklarasikan sebagai private menggunakan konvensi penamaan dengan dua garis bawah (misalnya, __gaji), yang mengaktifkan mekanisme name mangling Python untuk memproteksi atribut agar tidak dapat diakses langsung dari luar kelas.

Untuk mengakses dan memodifikasi atribut privat tersebut, digunakan metode getter dan setter. Getter merupakan metode publik yang bertugas mengambil atau membaca nilai atribut privat dengan cara yang aman, sedangkan setter berfungsi mengubah nilai atribut privat dengan menambahkan lapisan validasi dan proteksi agar hanya nilai valid yang diizinkan. Sebagai contoh, setter atribut gaji di modul memverifikasi bahwa nilai gaji yang akan diset adalah angka positif, sehingga mencegah nilai negatif yang akan merusak integritas data. Validasi ini merupakan komponen krusial dalam enkapsulasi untuk menjaga konsistensi dan kelayakan data.

Pada implementasi praktikum, dibuat kelas User dan Karyawan yang masing-masing memiliki atribut privat yang tidak bisa diakses langsung dari luar kelas. Override akses langsung ini menciptakan barrier yang mencegah modifikasi data yang tidak terkendali serta meningkatkan keamanan data. Adapun interaksi dengan atribut dilakukan hanya melalui getter dan setter yang sudah diimplementasikan dengan validasi logis sesuai kebutuhan aplikasi.

Selain aspek keamanan data, enkapsulasi juga berperan dalam modularitas kode program. Dengan menyembunyikan detail implementasi internal kelas, pengembang dapat melakukan perubahan internal tanpa memengaruhi kode luar selama kontrak antarmuka getter dan setter dipertahankan. Hal ini meningkatkan fleksibilitas, mengurangi ketergantungan kode, dan mempermudah pemeliharaan program dalam jangka panjang.

Dengan demikian, praktikum ini secara efektif mengajarkan penerapan enkapsulasi sebagai metode menyembunyikan informasi dan melindungi data pada objek, serta menanamkan pemahaman bahwa akses data yang terkontrol dan validasi input/output merupakan aspek fundamental dalam pengembangan perangkat lunak yang robust dan aman.

BAB V
PENUTUP

5.1 Kesimpulan

Enkapsulasi merupakan prinsip fundamental dalam pemrograman berorientasi objek yang menjalankan peran krusial dalam pengendalian akses terhadap data internal objek melalui pembatasan akses atribut dengan modifier akses, khususnya pada atribut privat. Implementasi enkapsulasi di Python menggunakan mekanisme name mangling melalui penggunaan atribut dengan prefix double underscore untuk menyembunyikan data dari akses langsung eksternal, sehingga menjaga integritas dan konsistensi data terhadap modifikasi yang tidak sah.

Pemanfaatan metode getter dan setter sebagai interface publik dalam mengakses dan memodifikasi atribut privat memungkinkan penerapan validasi yang efektif untuk memastikan data yang masuk memenuhi kriteria validitas sesuai kebutuhan bisnis atau logika program. Pendekatan ini tidak hanya meningkatkan keamanan data tetapi juga mendukung modularitas dan pemisahan tanggung jawab antara interface dan implementasi internal objek, sehingga mempermudah pemeliharaan dan pengembangan perangkat lunak.

Melalui praktikum ini, pemahaman konsep enkapsulasi diterapkan secara langsung dan sistematis dengan membatasi akses atribut dan memvalidasi perubahan data menggunakan getter dan setter, yang mempertegas pentingnya enkapsulasi sebagai mekanisme kunci untuk menyembunyikan informasi dan melindungi data dalam pengembangan perangkat lunak berorientasi objek.

5.2 Refleksi

Enkapsulasi disebut sebagai salah satu pilar fundamental dalam pemrograman berorientasi objek karena secara esensial mengontrol akses terhadap data dan perilaku objek melalui pembatasan akses atribut dan metode. Dengan menyembunyikan detail implementasi internal dan hanya menyediakan interface publik yang terdefinisi dengan jelas, enkapsulasi memastikan bahwa data objek tidak dapat diakses atau dimodifikasi secara langsung dari luar, sehingga menjaga integritas dan konsistensi data.

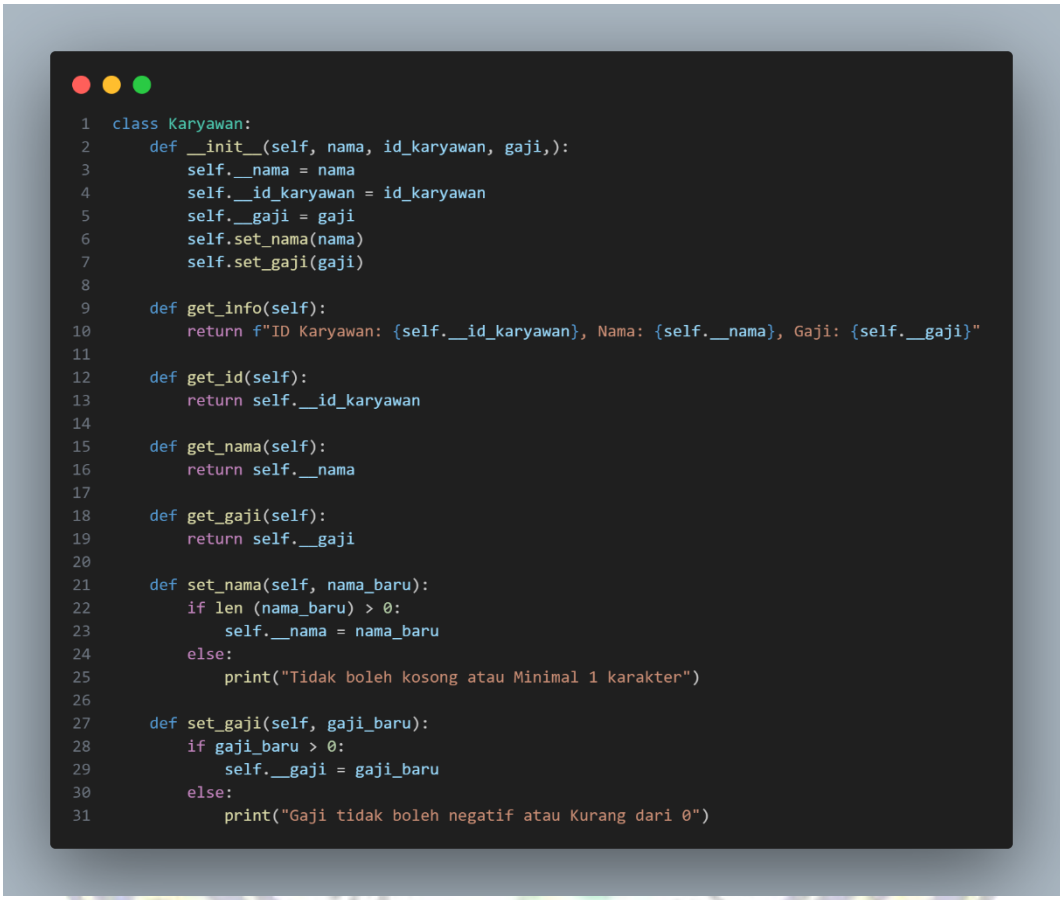
Selain itu, enkapsulasi mendukung modularitas dan abstraksi dengan memisahkan antara bagaimana data diorganisasikan dan dioperasikan secara

internal dengan bagaimana pengguna atau objek lain berinteraksi dengan objek tersebut. Hal ini memungkinkan perubahan internal dapat dilakukan tanpa memengaruhi bagian lain dari sistem selama kontrak antarmuka tetap terjaga, yang sangat penting untuk pemeliharaan dan skalabilitas perangkat lunak.

Secara keseluruhan, enkapsulasi menyediakan dasar keamanan data, mengurangi kompleksitas sistem, dan meningkatkan fleksibilitas desain perangkat lunak, sehingga menjadi pilar utama yang mendasari paradigma OOP yang terstruktur dan efisien.

LATIHAN MANDIRI

A. Code



```python
class Karyawan:
    def __init__(self, nama, id_karyawan, gaji,):
        self.__nama = nama
        self.__id_karyawan = id_karyawan
        self.__gaji = gaji
        self.set_nama(nama)
        self.set_gaji(gaji)

    def get_info(self):
        return f"ID Karyawan: {self.__id_karyawan}, Nama: {self.__nama}, Gaji: {self.__gaji}"

    def get_id(self):
        return self.__id_karyawan

    def get_nama(self):
        return self.__nama

    def get_gaji(self):
        return self.__gaji

    def set_nama(self, nama_baru):
        if len (nama_baru) > 0:
            self.__nama = nama_baru
        else:
            print("Tidak boleh kosong atau Minimal 1 karakter")

    def set_gaji(self, gaji_baru):
        if gaji_baru > 0:
            self.__gaji = gaji_baru
        else:
            print("Gaji tidak boleh negatif atau Kurang dari 0")
```

B. Input

```python
k1 = Karyawan("Budi", "K001", 5000000)

print("Data Awal:", k1.get_info())


print("\nCoba ubah gaji ke -5000000")
k1.set_gaji(-5000000)
print("Setelah dicoba:", k1.get_info())


print("\nCoba ubah nama jadi kosong")
k1.set_nama("")
print("Setelah dicoba:", k1.get_info())


print("\nUbah gaji ke 6000000")
k1.set_gaji(6000000)


print("\nData Terbaru:", k1.get_info())
```

C. Output

```
PS E:\University's Stuff\3rd\Pemrograman Berorientasi Objek\Pr
aktikum> & "E:/University's Stuff/3rd/Pemrograman Berorientasi
 Objek/Praktikum/.venv/Scripts/python.exe" "e:/University's St
uff/3rd/Pemrograman Berorientasi Objek/Praktikum/Latihan_3.py"

Data Awal: ID Karyawan: K001, Nama: Budi, Gaji: 5000000

Coba ubah gaji ke -5000000
Gaji tidak boleh negatif atau Kurang dari 0
Setelah dicoba: ID Karyawan: K001, Nama: Budi, Gaji: 5000000

Coba ubah nama jadi kosong
Tidak boleh kosong atau Minimal 1 karakter
Setelah dicoba: ID Karyawan: K001, Nama: Budi, Gaji: 5000000

Ubah gaji ke 6000000

Data Terbaru: ID Karyawan: K001, Nama: Budi, Gaji: 6000000
```

DAFTAR PUSTAKA

Fowler, M. (2004). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Guttag, J. V. (2016). *Introduction to Computation and Programming Using Python*. MIT Press. (Bab 2-3).

Jamal, A. (2014). Fitur Pemrograman Berorientasi Objek. *Jurnal AL-AZHAR INDONESIA SERI SAINS DAN TEKNOLOGI*, 2(4).

Liskov, B., & Guttag, J. (2001). *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*. Addison-Wesley.

Sommerville, I. (2011). *Software Engineering* (9th ed.). Addison-Wesley.