

Program Studi Teknik Informatika Institut Teknologi Sumatera

2024

**Pemrograman Berorientasi Objek**

**Laporan Praktikum**

|  |  |
| --- | --- |
| **Modul :** | **Abstraksi & Enkapsulasi** |
| **Nama :** | **Muhammad Yusuf** |
| **NIM :** | **122140193** |
| **Kelas (Kelas Asal) :** | **RA** |

Instruksi sederhana :

* Disarankan kepada **Praktikan Pemrograman Berorientasi Objek** untuk mengeditnya menggunakan Google Docs agar tidak berantakan dan rapi,
* Silahkan mengganti **Nama Modul** baik yang ada pada **Cover** dan **Header** sesuai dengan materi praktikum,
* Gunakan text styling seperti **Heading 1**, **Normal Text** yang telah terformat / Text Style lainnya yang digunakan untuk menjaga estetika laporan,
* Gunakan [**Syntax Highlighter**](https://highlight.hohli.com/index.php)untuk merapikan kode yang sudah Praktikan buat ke dalam Laporan Praktikum.

# Materi Praktikum

Abstraksi dan enkapsulasi adalah dua konsep penting dalam pemrograman berorientasi objek yang digunakan untuk merancang dan mengimplementasikan kelas dan objek. Abstraksi memungkinkan kita untuk menyembunyikan detail implementasi yang kompleks dari pengguna kelas atau objek, sehingga hanya menampilkan informasi yang relevan atau yang diperlukan. Di sisi lain, enkapsulasi memungkinkan kita untuk mengelompokkan data dan metode yang berhubungan bersama dalam satu unit yang disebut kelas, dan mengatur akses ke data tersebut dengan menentukan tingkat aksesibilitas, seperti public, private, atau protected. Dalam Python, abstraksi dapat dicapai dengan menggunakan metode dan properti untuk mengakses dan memanipulasi data dalam kelas, sedangkan enkapsulasi dilakukan dengan cara memberi aksesibilitas private pada atribut dan metode yang hanya dapat diakses dari dalam kelas itu sendiri. Dengan menerapkan kedua konsep ini, kita dapat menciptakan kode yang lebih bersih, modular, dan mudah dimengerti.

# Link Source Code Tugas 1

<https://onlinegdb.com/suwN_4pdbk>

# Source Code Tugas 1

class AkunBank:

    list\_pelanggan = []

    def \_\_init\_\_(*self*, *no\_pelanggan*, *nama\_pelanggan*, *jumlah\_saldo*):

*self*.no\_pelanggan = *no\_pelanggan*

*self*.nama\_pelanggan = *nama\_pelanggan*

*self*.\_\_jumlah\_saldo = *jumlah\_saldo*

        AkunBank.list\_pelanggan.append(*self*)

    def lihat\_menu(*self*):

        print("Selamat datang di Bank Jago")

        print(f"Halo {*self*.nama\_pelanggan}, ingin melakukan apa?")

        print("1. Lihat saldo")

        print("2. Tarik tunai")

        print("3. Transfer saldo")

        print("4. Keluar")

    def lihat\_saldo(*self*):

        print(f"{*self*.nama\_pelanggan} memiliki saldo Rp {*self*.\_\_jumlah\_saldo}")

    def tarik\_tunai(*self*):

        print("Masukkan jumlah nominal yang ingin ditarik:")

        jumlah\_tarik = int(input())

        if jumlah\_tarik > *self*.\_\_jumlah\_saldo:

            print("Nominal saldo yang Anda punya tidak cukup!")

        else:

*self*.\_\_jumlah\_saldo -= jumlah\_tarik

            print("Saldo berhasil ditarik!")

    def transfer(*self*):

        print("Masukkan nominal yang ingin ditransfer:")

        nominal\_transfer = int(input())

        print("Masukkan no rekening tujuan:")

        no\_rekening\_tujuan = int(input())

        tujuan = None

        for akun in AkunBank.list\_pelanggan:

            if akun.no\_pelanggan == no\_rekening\_tujuan:

                tujuan = akun

                break

        if tujuan is None:

            print("No rekening tujuan tidak dikenal!")

        else:

            if *self*.\_\_jumlah\_saldo >= nominal\_transfer:

*self*.\_\_jumlah\_saldo -= nominal\_transfer

                tujuan.\_\_jumlah\_saldo += nominal\_transfer

                print(f"Transfer Rp {nominal\_transfer} ke {tujuan.nama\_pelanggan} sukses!")

            else:

                print("Saldo tidak mencukupi!")

*# Membuat instansi AkunBank*

Akun1 = AkunBank(12345678, "Ujang", 1000000000000)

Akun2 = AkunBank(87654321, "Jamal", 9999999999999999)

Akun3 = AkunBank(12348765, "Mamank Recing", 500000)

*# Simulasi penggunaan fungsi-fungsi*

Akun1.lihat\_menu()

while True:

    print("Masukkan nomor input:")

    input\_nomor = int(input())

    if input\_nomor == 1:

        Akun1.lihat\_saldo()

    elif input\_nomor == 2:

        Akun1.tarik\_tunai()

    elif input\_nomor == 3:

        Akun1.transfer()

    elif input\_nomor == 4:

        print("Terima kasih telah menggunakan layanan kami!")

        break

# Dokumentasi Hasil Running Tugas 1

|  |
| --- |
| **Gambar 1. Output Tugas 1 Abstraksi & Enkapsulasi Kasus Akun Bank** |
|  |

Kode di atas mendefinisikan sebuah kelas `AkunBank` yang memiliki atribut untuk nomor pelanggan, nama pelanggan, dan jumlah saldo (yang dienkapsulasi). Metode-metode dalam kelas tersebut memungkinkan pengguna untuk melihat saldo, menarik tunai, dan mentransfer saldo antar akun. Setelah membuat beberapa instansi dari kelas `AkunBank`, sebuah simulasi interaksi dengan pengguna dibuat menggunakan perulangan yang memungkinkan pengguna memilih aksi yang ingin dilakukan (lihat saldo, tarik tunai, transfer, atau keluar). Seluruh logika diimplementasikan dengan benar untuk menangani input pengguna dan mengubah saldo serta melakukan transfer antar akun. Overall, program tersebut memungkinkan pengguna untuk mengelola akun bank dengan berbagai operasi yang umum dilakukan dalam sebuah layanan perbankan.

# Link Source Code Tugas 2

https://onlinegdb.com/FBWkloMer

# Source Code Tugas 2

class Pengguna:

*# Atribut private, hanya dapat diakses dari dalam kelas*

    \_\_username = "admin"

    \_\_password = "12345"

*# Atribut protected, dapat diakses dari dalam kelas dan kelas turunannya*

    \_level = "admin"

*# Atribut public, dapat diakses dari luar kelas*

    email = "admin@example.com"

    def \_\_init\_\_(*self*, *nama*):

*self*.nama = *nama*

*# Metode private, hanya dapat dipanggil dari dalam kelas*

    def \_\_ubah\_password(*self*, *new\_password*):

*self*.\_\_password = *new\_password*

        print("Password berhasil diubah")

*# Metode public, dapat dipanggil dari luar kelas*

    def tampilkan\_info(*self*):

        print(f"Nama: {*self*.nama}")

        print(f"Email: {*self*.email}")

        print(f"Level: {*self*.\_level}")

*# Metode untuk mengubah password, dapat dipanggil dari luar kelas*

    def ganti\_password(*self*, *current\_password*, *new\_password*):

        if *current\_password* == *self*.\_\_password:

*self*.\_\_ubah\_password(*new\_password*)

        else:

            print("Password salah")

*# Membuat objek pengguna*

user1 = Pengguna("Alice")

*# Mengakses atribut public dari luar kelas*

print("Email pengguna:", user1.email)

*# Mengakses atribut protected dari luar kelas*

print("Level pengguna:", user1.\_level)

*# Percobaan mengakses atribut private dari luar kelas akan menghasilkan error*

*# print("Username:", user1.\_\_username)  # Error: AttributeError*

*# Memanggil metode public untuk menampilkan informasi pengguna*

user1.tampilkan\_info()

*# Percobaan memanggil metode private dari luar kelas akan menghasilkan error*

*# user1.\_\_ubah\_password("newpassword")  # Error: AttributeError*

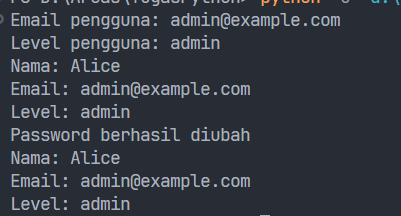
*# Memanggil metode public untuk mengubah password*

user1.ganti\_password("12345", "newpassword")

*# Memanggil metode public untuk menampilkan informasi pengguna setelah mengubah password*

user1.tampilkan\_info()

# Dokumentasi Hasil Running Tugas 2



**Gambar 2. Output Tugas 1 Abstraksi & Enkapsulasi Demontrsasi Private dan Publik Attribute/Method**

Pada kode di atas, kita mendefinisikan sebuah kelas `Pengguna` dengan beberapa atribut dan metode yang menerapkan konsep abstraksi dan enkapsulasi. Atribut `\_\_username` dan `\_\_password` dideklarasikan sebagai private, yang berarti hanya dapat diakses dari dalam kelas. Atribut `\_level` dideklarasikan sebagai protected, sehingga dapat diakses dari dalam kelas dan kelas turunannya. Sedangkan atribut `email` dideklarasikan sebagai public, sehingga dapat diakses dari luar kelas. Metode `\_\_ubah\_password()` didefinisikan sebagai private untuk mengubah password, sementara metode `tampilkan\_info()` dan `ganti\_password()` didefinisikan sebagai public untuk menampilkan informasi pengguna dan mengubah password. Pada saat membuat objek `user1` dari kelas `Pengguna`, kita dapat mengakses atribut dan memanggil metode yang sesuai dengan tingkat aksesibilitasnya. Misalnya, atribut `email` dapat diakses langsung dari luar kelas, sedangkan metode `ganti\_password()` hanya dapat diakses melalui pemanggilan objek dari luar kelas. Dengan menerapkan konsep abstraksi dan enkapsulasi ini, kita dapat membatasi akses ke data dan metode yang sensitif, sehingga meningkatkan keamanan dan modularitas kode.