LAPORAN PRAKTIKUM PEMOGRAMAN BERORIERNTASI OBJEK (PBO)
PRAKTIKUM 2

2411102441249
HERVINO ISLAMI FASHA

FAKULTAS SAINS DAN TEKNOLOGI
PROGRAM STUDI S1 TEKNIK INFORMATIKA
UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

2411102441249@umkt.ac.id

1. Menyiapkan Struktur Inheritanc

```python
#menyiapkan struktur Inheritance

#PARENT CLASS
class Bentuk:
    def gambar(self):
        #Method sengaja dibuat umum dan akan di override
        raise NotImplementedError("Subclass harus mengimplementasikan methode ini!")

#CHILD CLASS 1
class Persegi(Bentuk):
    def gambar(self):
        print("Menggambar Persegi: [][][]")

#CHILD CLASS 2
class Lingkaran(Bentuk):
    def gambar(self):
        print("menggambar Lingkaran: 000000")
```

- Definisi bentuk untuk method umum (gambar()).
- Setiap kelas turunannya (Persegi, Lingkaran) mengimplementasikan method yang sama dengan cara yang berbeda.
- Akibatnya, saat memanggil obj. gambar(), perilakunya dapat bervariasi berdasarkan objek yang digunakan.

2. Mendemonstrasikan Polymorphism dalam Aksi

```python
# --- Bagian Utama Program ---
#Membuat list yang berisi objek-objek dari class yang berbeda
daftar_bentuk = [Persegi(), Lingkaran(), Persegi(), Lingkaran()]

print("--- Memanggil method yang sama pada objek yang berbeda ---")
#iterasi melalui list dan panggil method .gambar() pada setiap objek
for bentuk in daftar_bentuk:
    bentuk.gambar() #Satu pemanggilan, banyak bentuk/perilaku!
```

Kode ini menghasilkan sebuah daftar yang berisi objek dari kelas yang berbeda (Persegi dan Lingkaran). Selanjutnya, digunakan loop for untuk memanggil metode. gambar() di setiap objek. Meskipun metode yang diakses adalah sama, hasil yang diperoleh berbeda tergantung pada kelasnya masing-masing. Ini adalah contoh dari polymorphism: satu metode, banyak aksi.

OUTPUT:

```
D:\smt 3\PBO\P5>C:/Python313/python.exe "d:/smt 3/PBO/P5/pertemuan_5.py"
--- Memanggil method yang sama pada objek yang berbeda ---
Menggambar Persegi: [][][]
menggambar Lingkaran: OOOOOO
Menggambar Persegi: [][][]
menggambar Lingkaran: OOOOOO
```

3. Polymorphism dengan fungsi

```python
#CHILD CLASS 3
class Segitiga(Bentuk):
    def gambar(self):
        print("Mengambar Segitiga: /\\")

#CLASS YANG TIDAK BERHUBUNGAN (untuk duck typing)
class Teks:
    def gambar(self):
        print("Menulis Teks: 'Hello, Polymorphism!'")

#Sebuah fungsi yang menunjukkan perilaku polimorfik
def render_objek(objek_untuk_digambar):
    print("Mencoba me-render objek...")
    objek_untuk_digambar.gambar()

#bagian utama program
persegi = Persegi()
lingkaran = Lingkaran()
segitiga = Segitiga()
teks_biasa = Teks() #objek ini Bukan dari turunan bentuk

print("\n--- Menggunakan fungsi polimorfik ---")
render_objek(persegi)
render_objek(lingkaran)
render_objek(segitiga)

print("\n--- Demonstrasi Duck Typing ---")
render_objek(teks_biasa) #fungsi tetap bekerja
```

Kode ini menambahkan kelas Segitiga yang merupakan turunan dari kelas Bentuk dan kelas Teks yang tidak memiliki hubungan. Kemudian, dibuatlah sebuah fungsi bernama render_objek() yang dapat menerima objek apa saja, asalkan objek tersebut memiliki metode gambar().

- Ketika fungsi ini dipanggil dengan objek Persegi, Lingkaran, dan Segitiga, fungsi akan menampilkan gambar sesuai dengan bentuk masing-masing.
- Ketika fungsi ini dipanggil dengan Teks, fungsi tetap berfungsi meskipun Teks bukan merupakan turunan dari Bentuk.

Hal ini menggambarkan konsep polymorphism (satu fungsi dapat digunakan oleh banyak objek turunan) serta duck typing dalam Python (selama objek memiliki metode yang tepat, fungsi masih dapat dijalankan).

Output:

```
D:\smt 3\PBO\P5>C:/Python313/python.exe "d:/smt 3/PBO/P5/pertemuan_5.py"
--- Memanggil method yang sama pada objek yang berbeda ---
Menggambar Persegi: [][][]
menggambar Lingkaran: OOOOOO
Menggambar Persegi: [][][]
menggambar Lingkaran: OOOOOO

--- Menggunakan fungsi polimorfik ---
Mencoba me-render objek...
Menggambar Persegi: [][][]
Mencoba me-render objek...
menggambar Lingkaran: OOOOOO
Mencoba me-render objek...
Mengambar Segitiga: /\

--- Demonstrasi Duck Typing ---
Mencoba me-render objek...
Menulis Teks: 'Hello, Polymorphism!'
```

TUGAS

A. Code

```python
from abc import ABC, abstractmethod

# Parent Class abstrak
class Notifikasi(ABC):
    @abstractmethod
    def kirim(self, pesan):
        raise NotImplementedError("Method kirim() harus diimplementasikan di class turunan")

# Child Class: Email
class Email(Notifikasi):
    def kirim(self, pesan):
        print(f"[EMAIL] Mengirim: '{pesan}'")

# Child Class: SMS
class SMS(Notifikasi):
    def kirim(self, pesan):
        print(f"[SMS] Mengirim: '{pesan}'")

# Child Class: PushNotif
class PushNotif(Notifikasi):
    def kirim(self, pesan):
        print(f"[PUSH] Mengirim: '{pesan}'")

# Bagian utama program
if __name__ == "__main__":
    # Membuat list objek dari setiap kanal
    daftar_notif = [
        Email(),
        SMS(),
        PushNotif()
    ]

    # Pesan yang akan dikirim
    pesan = "Diskon Spesial! Hanya untuk Anda!"

    # Loop polymorphism
    for notif in daftar_notif:
        notif.kirim(pesan)
```

B. Output

```
D:\smt 3\PBO\P5>C:/Python313/python.exe "d:/smt 3/PBO/P5/tugas_5.py"
[EMAIL] Mengirim: 'Diskon Spesial! Hanya untuk Anda!'
[SMS] Mengirim: 'Diskon Spesial! Hanya untuk Anda!'
[PUSH] Mengirim: 'Diskon Spesial! Hanya untuk Anda!'
```

C. Refleksi

Dengan penerapan desain polimorfik, kita tidak perlu menulis banyak cabang if-elif-else untuk membedakan tipe notifikasi. Cukup gunakan metode kirim() pada objek, dan masing-masing kelas turunan akan mengurus caranya sendiri. Ini membuat kode menjadi lebih ringkas, lebih

mudah dipahami, serta lebih fleksibel untuk pengembangan jika ada saluran notifikasi baru yang muncul di masa mendatang.