LAPORAN PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK (PBO)

PRAKTIKUM 11

Nim : 2411102441242

Nama : Muhammad Rifqi Ihsan

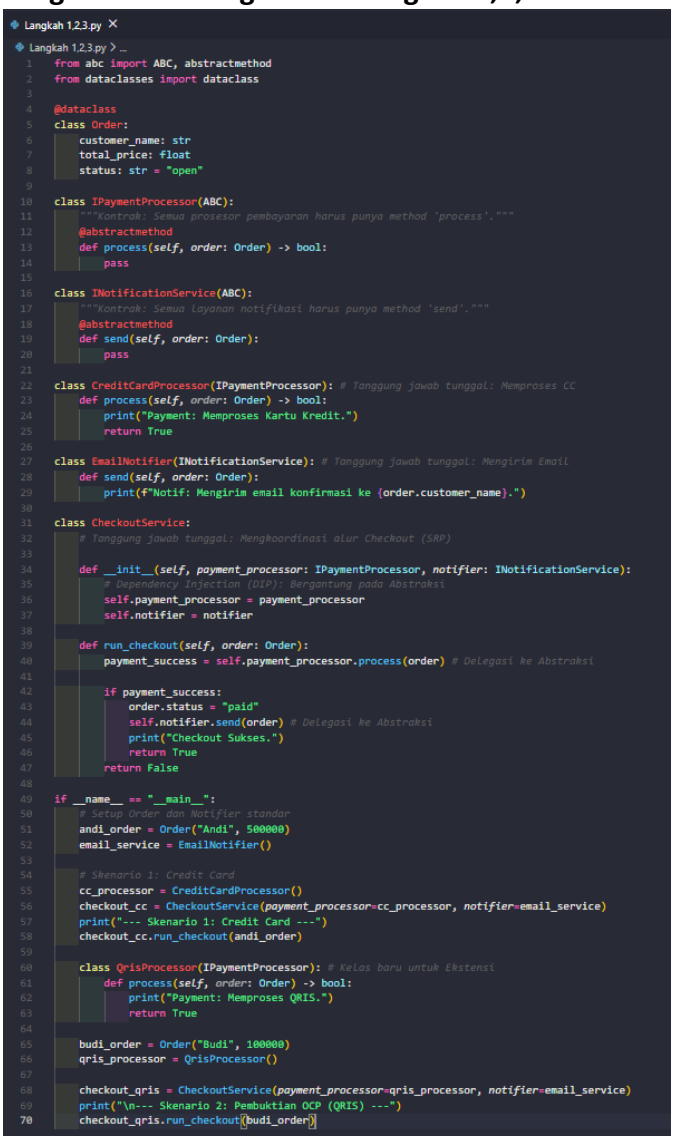
FAKULTAS SAINS DAN TEKNOLOGI

PROGRAM STUDI S1 TEKNIK INFORMATIKA

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

EMAIL : 2411102441242@umkt.ac.id

Langkah 1: Gabungan Dari Langkah 1,2,3



```python
from abc import ABC, abstractmethod
from dataclasses import dataclass

@dataclass
class Order:
    customer_name: str
    total_price: float
    status: str = "open"

class IPaymentProcessor(ABC):
    """Kontrak: Semua prosesor pembayaran harus punya method 'process'."""
    @abstractmethod
    def process(self, order: Order) -> bool:
        pass

class INotificationService(ABC):
    """Kontrak: Semua layanan notifikasi harus punya method 'send'."""
    @abstractmethod
    def send(self, order: Order):
        pass

class CreditCardProcessor(IPaymentProcessor): # Tanggung jawab tunggal: Memproses CC
    def process(self, order: Order) -> bool:
        print("Payment: Memproses Kartu Kredit.")
        return True

class EmailNotifier(INotificationService): # Tanggung jawab tunggal: Mengirim Email
    def send(self, order: Order):
        print(f"Notif: Mengirim email konfirmasi ke {order.customer_name}.")

class CheckoutService:
    # Tanggung jawab tunggal: Mengkoordinasi alur Checkout (SRP)

    def __init__(self, payment_processor: IPaymentProcessor, notifier: INotificationService):
        # Dependency Injection (DIP): Bergantung pada Abstraksi
        self.payment_processor = payment_processor
        self.notifier = notifier

    def run_checkout(self, order: Order):
        payment_success = self.payment_processor.process(order) # Delegasi ke Abstraksi

        if payment_success:
            order.status = "paid"
            self.notifier.send(order) # Delegasi ke Abstraksi
            print("Checkout Sukses.")
            return True
        return False

if __name__ == "__main__":
    # Setup Order dan Notifier standar
    andi_order = Order("Andi", 500000)
    email_service = EmailNotifier()

    # Skenario 1: Credit Card
    cc_processor = CreditCardProcessor()
    checkout_cc = CheckoutService(payment_processor=cc_processor, notifier=email_service)
    print("--- Skenario 1: Credit Card ---")
    checkout_cc.run_checkout(andi_order)

    class QrisProcessor(IPaymentProcessor): # Kelas baru untuk Ekstensi
        def process(self, order: Order) -> bool:
            print("Payment: Memproses QRIS.")
            return True

    budi_order = Order("Budi", 100000)
    qris_processor = QrisProcessor()

    checkout_qris = CheckoutService(payment_processor=qris_processor, notifier=email_service)
    print("\n--- Skenario 2: Pembuktian OCP (QRIS) ---")
    checkout_qris.run_checkout(budi_order)
```

Output:

```
[Running] python -u "c:\Users\Asus\Downloads\PBO 11,12,13\Langkah 1,2,3.py"
--- Skenario 1: Credit Card ---
Payment: Memproses Kartu Kredit.
Notif: Mengirim email konfirmasi ke Andi.
Checkout Sukses.

--- Skenario 2: Pembuktian OCP (QRIS) ---
Payment: Memproses QRIS.
Notif: Mengirim email konfirmasi ke Budi.
Checkout Sukses.

[Done] exited with code=0 in 0.17 seconds
```

EMAIL : 2411102441242@umkt.ac.id

Studi Kasus: Kalkulator Gaji Karyawan

(Latihan Mandiri) Kalkulator Gaji Karyawan.py

```python
from abc import ABC, abstractmethod
from dataclasses import dataclass

# --- MODEL SEDERHANA ---
@dataclass
class Karyawan:
    nama: str
    gaji_pokok: float
    jam_kerja_bulan: int = 0
    tunjangan: float = 0.0

# --- ABSTRAKSI (Kontrak untuk OCP/DIP) ---
class ISalaryCalculator(ABC):
    """Kontrak: Semua kalkulator harus punya method 'calculate'."""
    @abstractmethod
    def calculate(self, karyawan: Karyawan) -> float:
        pass

# --- IMPLEMENTASI KONKRIT (Plug-in - OCP) ---
class PermanentSalaryCalculator(ISalaryCalculator):
    """SRP: Hanya bertanggung jawab menghitung gaji Karyawan Tetap."""
    def calculate(self, karyawan: Karyawan) -> float:
        gaji_bersih = karyawan.gaji_pokok + karyawan.tunjangan
        return gaji_bersih

class ContractSalaryCalculator(ISalaryCalculator):
    """SRP: Hanya bertanggung jawab menghitung gaji Karyawan Kontrak."""
    def calculate(self, karyawan: Karyawan) -> float:
        # Asumsi perhitungan: Gaji Pokok + (Bonus 10% dari Gaji Pokok)
        gaji_bersih = karyawan.gaji_pokok * 1.10
        return gaji_bersih

# --- KELAS KOORDINATOR (SRP & DIP) ---
class PayrollProcessor:
    """SRP: Tanggung jawab tunggal adalah memproses gaji menggunakan kalkulator yang di-inject."""
    def __init__(self, calculator: ISalaryCalculator):
        # DIP: Bergantung pada Abstraksi ISalaryCalculator
        self.calculator = calculator

    def process_payroll(self, karyawan: Karyawan) -> float:
        gaji = self.calculator.calculate(karyawan)
        print(f"Gaji {karyawan.nama} (menggunakan {self.calculator.__class__.__name__}): Rp {gaji:,.2f}")
        return gaji

# --- PROGRAM UTAMA & PEMBUKTIAN OCP ---
if __name__ == "__main__":
    # Karyawan
    joko = Karyawan("Joko (Tetap)", 8000000, tunjangan=1000000)
    sinta = Karyawan("Sinta (Kontrak)", 4500000)

    # 1. Proses Karyawan Tetap
    permanent_calc = PermanentSalaryCalculator()
    payroll_joko = PayrollProcessor(calculator=permanent_calc)
    payroll_joko.process_payroll(joko)

    # 2. Proses Karyawan Kontrak
    contract_calc = ContractSalaryCalculator()
    payroll_sinta = PayrollProcessor(calculator=contract_calc)
    payroll_sinta.process_payroll(sinta)

    class PartTimeSalaryCalculator(ISalaryCalculator):
        """Aturan Baru: Gaji = Jam Kerja * Tarif Per Jam (Rp 50.000)"""
        def calculate(self, karyawan: Karyawan) -> float:
            gaji_bersih = karyawan.jam_kerja_bulan * 50000
            return gaji_bersih

    rudi = Karyawan("Rudi (Paruh Waktu)", gaji_pokok=0, jam_kerja_bulan=160)

    parttime_calc = PartTimeSalaryCalculator()
    payroll_rudi = PayrollProcessor(calculator=parttime_calc)

    print("\n" + "-"*50)
    print("DEMO OCP: Menggunakan Kalkulator Baru (PartTimeSalaryCalculator)")
    print("-"*50)
    payroll_rudi.process_payroll(rudi)
```

EMAIL : 2411102441242@umkt.ac.id

Output:

```
[Running] python -u "c:\Users\Asus\Downloads\PBO 11,12,13\(Latihan Mandiri) Kalkulator Gaji Karyawan.py"
Gaji Joko (Tetap) (menggunakan PermanentSalaryCalculator): Rp 9,000,000.00
Gaji Sinta (Kontrak) (menggunakan ContractSalaryCalculator): Rp 4,950,000.00

================================================
DEMO OCP: Menggunakan Kalkulator Baru (PartTimeSalaryCalculator)
================================================
Gaji Rudi (Paruh Waktu) (menggunakan PartTimeSalaryCalculator): Rp 8,000,000.00

[Done] exited with code=0 in 0.331 seconds
```

Refleksi: DI memisahkan logika menjadi unit-unit kecil yang independen (misalnya, SKSLimitRule atau PermanentSalaryCalculator). Setiap unit memiliki tanggung jawab tunggalnya. Kelas koordinator hanya bertugas mengorkestrasi alur kerja, bukan menampung semua logika. Ini secara radikal menghilangkan *The God Class*.

EMAIL : 2411102441242@umkt.ac.id