LAPORAN PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK (PBO)

PRAKTIKUM 11

2411102441052

Angga Maulana Saputra

FAKULTAS SAINS DAN TEKNOLOGI

PROGRAM STUDI S1 TEKNIK INFORMATIKA

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

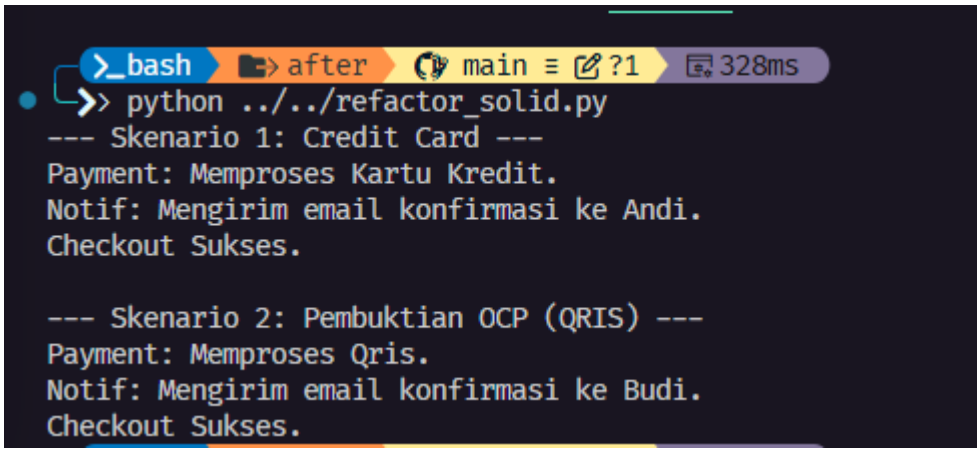
EMAIL : 2411102441052@umkt.ac.id

Link Github: https://github.com/nikamushi/praktikum-pbo/tree/main/pertemuan11

a. Screenshoot kode latihan

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

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

# === KODE BURUK (SEBELUM REFACTOR) ===
class OrderManager: # Melanggar SRP, OCP, DIP
    def process_checkout(self, order: Order, payment_method: str):
        print(f"Memulai checkout untuk {order.customer_name}...")

        # LOGIKA PEMBAYARAN (Pelanggaran OCP/DIP)
        if payment_method == "credit_card":
            # Logika detail implementasi hardcoded di sini
            print("Processing Credit Card...")
        elif payment_method == "bank_transfer":
            # Logika detail implementasi hardcoded di sini
            print("Processing Bank Transfer...")
        else:
            print("Metode tidak valid.")
            return False

        # LOGIKA NOTIFIKASI (Pelanggaran SRP)
        print(f"Mengirim notifikasi ke {order.customer_name}...")
        order.status = "paid"
        return True

# --- ABSTRAKSI (Kontrak untuk OCP/DIP) ---
class IPaymentProcessor(ABC):
    """Kontrak: Semua prosesor pembayaran hrus punya method 'process'."""
    @abstractmethod
    def process(self, order: Order) -> bool:
        pass

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

#     IMPLEMENTASI KONKRIT (Plug in)
class CreditCardProcessor(IPaymentProcessor):
    def process(self, order: Order) -> bool:
        print("Payment: Memproses Kartu Kredit.")
        return True

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

# --- KELAS KOORDINASI (SRP & DIP) ---
class CheckoutService: # Tanggung jawab tunggal: Mengkoordinasikan Checkout
    def __init__(self, payment_processor: IPaymentProcessor, notifier: INotificationService):
        # Dependency Injection (DIP): Bergantung pada Abstraksi, bukan Konkrit
        self.payment_processor = payment_processor
        self.notifier = notifier

    def run_checkout(self, order: Order):
        payment_success = self.payment_processor.process(order) # Delegasi 1

        if payment_success:
            order.status = "paid"
            self.notifier.send(order) # Delegasi 2
            print("Checkout Sukses.")
            return True
        return False

#     PROGRAM UTAMA
# Setup Dependencies
andi_order = Order("Andi", 500000)
email_service = EmailNotifier()

# 1. Inject implementasi 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)

# 2. Pembuktian OCP: Menambahkan Metode Pembayaran QRIS (Tanpa Mengubah CheckoutService)
class QrisProcessor(IPaymentProcessor):
    def process(self, order: Order) -> bool:
        print("Payment: Memproses Qris.")
        return True

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

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

b. Output


>_bash
after
main ≡ ?1
328ms
>> python ../../refactor_solid.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.

c. Screenshoot kode latihan mandiri

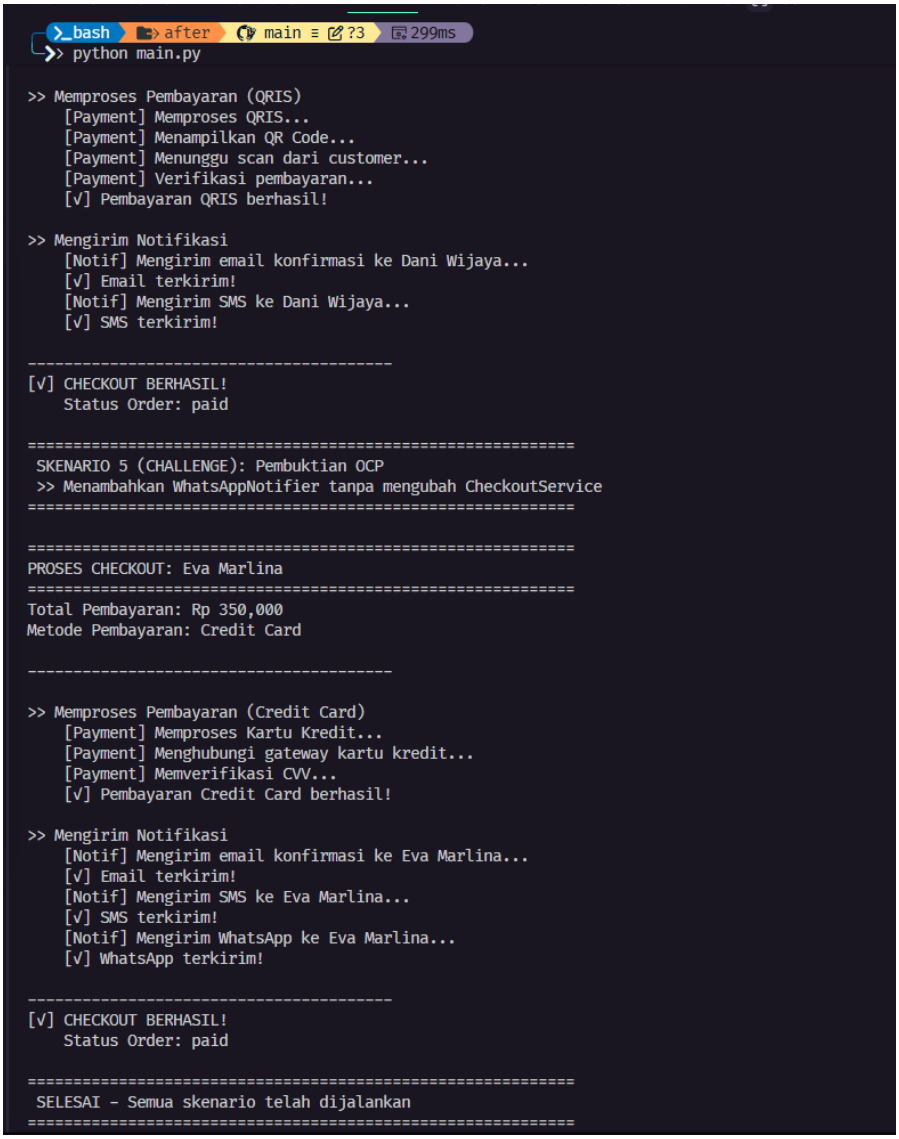
```python
class Pelanggan:
    def __init__(self, nama, saldo):
        self.__nama = nama
        self.__saldo = saldo

    def get_nama(self):
        return self.__nama

    def get_saldo(self):
        return self.__saldo

    def set_saldo(self, saldo_baru):
        if saldo_baru >= 0:
            self.__saldo = saldo_baru
        else:
            print("Saldo tidak boleh negatif!")

    def kurangi_saldo(self, jumlah):
        if jumlah <= self.__saldo:
            self.__saldo -= jumlah
            print(f"{self.__nama} membayar sebesar Rp{jumlah}")
        else:
            print("Saldo tidak cukup!")

    def __str__(self):
        return f"Pelanggan: {self.__nama}, Saldo: Rp{self.__saldo}"

class PelangganVIP(Pelanggan):
  def __init__(self, nama, saldo, diskon):
      super().__init__(nama, saldo)
      self.__diskon = diskon

  def get_diskon(self):
        return self.__diskon

  def kurangi_saldo(self, jumlah):
        jumlah_diskon = jumlah - (jumlah * self.__diskon / 100)
        if jumlah_diskon <= self.get_saldo():
            self.set_saldo(self.get_saldo() - jumlah_diskon)
            print(f"{self.get_nama()} (VIP) membayar Rp{jumlah_diskon} dengan diskon {self.__diskon}%")
        else:
            print("Saldo tidak cukup!")

  def __str__(self):
        return f"Pelanggan VIP: {self.get_nama()}, Saldo: Rp{self.get_saldo()}, Diskon: {self.__diskon}%"
```

d. Screenshoot kode latihan mandiri

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

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

class IPaymentProcessor(ABC):
    @property
    @abstractmethod
    def payment_name(self) -> str:
        """Nama metode pembayaran untuk ditampilkan"""
        pass

    @abstractmethod
    def process(self, order: Order) -> bool:
        """
        Memproses pembayaran untuk order.
        Return True jika berhasil, False jika gagal.
        """
        pass


class INotificationService(ABC):
    @property
    @abstractmethod
    def service_name(self) -> str:
        """Nama layanan notifikasi"""
        pass

    @abstractmethod
    def send(self, order: Order) -> bool:
        """Mengirim notifikasi untuk order."""
        pass

class CreditCardProcessor(IPaymentProcessor):
    @property
    def payment_name(self) -> str:
        return "Credit Card"

    def process(self, order: Order) -> bool:
        print(f"    [Payment] Memproses Kartu Kredit...")
        print(f"    [Payment] Menghubungi gateway kartu kredit...")
        print(f"    [Payment] Memverifikasi CVV...")
        print(f"    [v] Pembayaran Credit Card berhasil!")
        return True


class BankTransferProcessor(IPaymentProcessor):
    @property
    def payment_name(self) -> str:
        return "Bank Transfer"

    def process(self, order: Order) -> bool:
        print(f"    [Payment] Memproses Transfer Bank...")
        print(f"    [Payment] Mengecek rekening tujuan...")
        print(f"    [Payment] Menunggu konfirmasi bank...")
        print(f"    [v] Pembayaran Bank Transfer berhasil!")
        return True


class EWalletProcessor(IPaymentProcessor):
    @property
    def payment_name(self) -> str:
        return "E-Wallet"

    def process(self, order: Order) -> bool:
        print(f"    [Payment] Memproses E-Wallet...")
        print(f"    [Payment] Mengirim request ke provider...")
        print(f"    [v] Pembayaran E-Wallet berhasil!")
        return True

class EmailNotifier(INotificationService):
    @property
    def service_name(self) -> str:
        return "Email"

    def send(self, order: Order) -> bool:
        print(f"    [Notif] Mengirim email konfirmasi ke {order.customer_name}...")
        print(f"    [v] Email terkirim!")
        return True


class SmsNotifier(INotificationService):
    @property
    def service_name(self) -> str:
        return "SMS"

    def send(self, order: Order) -> bool:
        print(f"    [Notif] Mengirim SMS ke {order.customer_name}...")
        print(f"    [v] SMS terkirim!")
        return True
```

```python
class CheckoutService:
    def __init__(self, payment_processor: IPaymentProcessor,
                 notifiers: List[INotificationService] = None):
        """
        Dependency Injection: IPaymentProcessor,
        dan List[INotificationService].
        """
        self._payment_processor = payment_processor
        self._notifiers = notifiers or []

    def run_checkout(self, order: Order) -> bool:
        """Menjalankan proses checkout."""

        print(f"\n{'='*50}")
        print(f"[PROSES CHECKOUT] {order.customer_name}")
        print(f"{'='*50}")
        print(f"Total Pembayaran: Rp {order.total_price:,.0f}")
        print(f"Metode Pembayaran: {self._payment_processor.payment_name}")
        print(f"\n{'-' * 50}")

        # Delegasi ke Proses Pembayaran
        print(f"\n>>> Memproses Pembayaran: {self._payment_processor.payment_name}")
        payment_success = self._payment_processor.process(order)

        if not payment_success:
            print(f"\n{'=' * 50}")
            print(f"[X] CHECKOUT GAGAL - Pembayaran ditolak")
            print(f"{'=' * 50}")
            return False

        # Update status
        order.status = "paid"

        # Delegasi ke semua notifikasi
        if self._notifiers:
            print(f"\n>>> Mengirim Notifikasi")
            for notifier in self._notifiers:
                notifier.send(order)

        print(f"\n{'=' * 50}")
        print(f"[v] CHECKOUT BERHASIL!")
        print(f"    Status Order: {order.status}")
        print(f"{'=' * 50}\n")
        return True


if __name__ == "__main__":
    print("=" * 60)
    print("  SISTEM CHECKOUT - SEBELUM REFACTORING")
    print("  Tanpa Interface, ABC, DIP, atau Dependency Injection")
    print("=" * 60)

    email_notifier = EmailNotifier()
    sms_notifier = SmsNotifier()

    print("\n" + "=" * 60)
    print("  SKENARIO 1: Checkout dengan Credit Card")
    print("=" * 60)

    order1 = Order("Andi Pratama", 500000)
    cc_processor = CreditCardProcessor()

    checkout_cc = CheckoutService(
        payment_processor=cc_processor,
        notifiers=[email_notifier]
    )
    checkout_cc.run_checkout(order1)

    print("\n" + "=" * 60)
    print("  SKENARIO 2: Checkout dengan Bank Transfer")
    print("=" * 60)

    order2 = Order("Budi Santoso", 750000)
    bank_processor = BankTransferProcessor()

    checkout_bank = CheckoutService(
        payment_processor=bank_processor,
        notifiers=[email_notifier, sms_notifier]  # multiple notifiers
    )
    checkout_bank.run_checkout(order2)

    print("\n" + "=" * 60)
    print("  SKENARIO 3: Checkout dengan E-Wallet")
    print("=" * 60)

    order3 = Order("Citra Dewi", 250000)
    ewallet_processor = EWalletProcessor()

    checkout_ewallet = CheckoutService(
        payment_processor=ewallet_processor,
        notifiers=[sms_notifier]
    )
    checkout_ewallet.run_checkout(order3)

    print("\n" + "=" * 60)
    print("  SKENARIO 4: Checkout dengan QRIS (Extensibility)")
    print("  -> Menambahkan QrisProcessor tanpa mengubah CheckoutService")
    print("=" * 60)

    class QrisProcessor(IPaymentProcessor):
        @property
        def payment_name(self) -> str:
            return "QRIS"

        def process(self, order: Order) -> bool:
            print(f"    [Payment] Memproses QRIS...")
            print(f"    [Payment] Generating QR Code...")
            print(f"    [Payment] Menunggu scan dari customer...")
            print(f"    [Payment] Verifikasi pembayaran...")
            print(f"    [v] Pembayaran QRIS berhasil!")
            return True

    order4 = Order("Dewi Lestari", 100000)
    qris_processor = QrisProcessor()

    checkout_qris = CheckoutService(
        payment_processor=qris_processor,
        notifiers=[email_notifier, sms_notifier]
    )
    checkout_qris.run_checkout(order4)

    print("\n" + "=" * 60)
    print("  SKENARIO 5: Tanpa Notifikasi (Opsional)")
    print("  -> Menambahkan WhatsAppNotifier tanpa mengubah CheckoutService")
    print("=" * 60)

    class WhatsAppNotifier(INotificationService):
        @property
        def service_name(self) -> str:
            return "WhatsApp"

        def send(self, order: Order) -> bool:
            print(f"    [Notif] Mengirim WhatsApp ke {order.customer_name}...")
            print(f"    [v] WhatsApp terkirim!")
            return True

    order5 = Order("Eko Prasetyo", 300000)
    wa_notifier = WhatsAppNotifier()

    checkout_full = CheckoutService(
        payment_processor=cc_processor,
        notifiers=[email_notifier, sms_notifier, wa_notifier]  # 3 notifiers
    )
    checkout_full.run_checkout(order5)

    print("\n" + "=" * 60)
    print("  Selesai - Semua skenario telah dijalankan")
    print("=" * 60)
```

e. Output



f. Refleksi Singkat: Pendekatan Dependency Injection (DI) lebih efektif mencegah Code Smell dibandingkan if/else karena beberapa alasan. Pertama, DI menghilangkan "Long Method" karena method koordinator tetap pendek dan fokus, tidak membengkak seiring bertambahnya fitur seperti yang terjadi pada if/else yang terus menambah blok elif baru. Kedua, DI mencegah "Shotgun Surgery" karena menambah fitur baru cukup dengan membuat class baru tanpa mengubah kode existing, sehingga tidak berisiko merusak fitur yang sudah berjalan. Ketiga, DI mengatasi "God Class" dengan memisahkan tanggung jawab ke class-class kecil yang masing-masing fokus pada satu tugas sesuai prinsip Single Responsibility. Keempat, DI menghasilkan Loose Coupling dimana class koordinator hanya bergantung pada abstraksi (interface), bukan implementasi konkrit, sehingga mudah diganti dan di-test menggunakan mock object. Kesimpulannya, DI menghasilkan kode yang modular, fleksibel, dan mudah di-maintain karena mengikuti prinsip Open/Closed dimana sistem terbuka untuk ekstensi namun tertutup untuk modifikasi, berbeda dengan if/else yang menciptakan kode monolitik dan rigid.