**LAPORAN PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK (PBO)**

**PRAKTIKUM 12**

**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 dan 2**

Hasil Langkah 1 dan 2.py > ...

```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 AWAL BERMASALAH (Langkah 1: The God Class) ===
# ----------------------------------------------------------------------
# Kode ini melanggar SRP (karena menangani pembayaran, notifikasi, dan checkout
# dan OCP/DIP (karena menggunakan if/else untuk logika pembayaran).
#
# class OrderManager:
#     def process_checkout(self, order: Order, payment_method: str):
#         print(f"Memulai checkout untuk {order.customer_name}...")
#
#         if payment_method == "credit_card":
#             print("Processing Credit Card...")
#         elif payment_method == "bank_transfer":
#             print("Processing Bank Transfer...")
#         else:
#             print("Metode tidak valid.")
#             return False
#
#         print(f"Mengirim notifikasi ke {order.customer_name}...")
#         order.status = "paid"
#         return True

# ----------------------------------------------------------------------
# === SOLUSI REFACTORING (Langkah 2: Abstraksi, SRP, DIP) ===
# ----------------------------------------------------------------------

# --- ABSTRAKSI (Kontrak untuk OCP/DIP) ---
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

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

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

**EMAIL : 2411102441242@umkt.ac.id**

```python
# --- KELAS KOORDINATOR (SRP & DIP) ---
class CheckoutService:
    """
    Kelas high-level untuk mengkoordinasi proses transaksi pembayaran.
    Kelas ini memisahkan logika pembayaran dan notifikasi (memenuhi SRP).
    """
    def __init__(self, payment_processor: IPaymentProcessor, notifier: INotificationService):
        """
        Args:
            payment_processor (IPaymentProcessor): Implementasi Interface pembayaran.
            notifier (INotificationService): Implementasi Interface notifikasi.
        """
        # Dependency Injection (DIP): Bergantung pada Abstraksi
        self.payment_processor = payment_processor
        self.notifier = notifier

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

        Args:
            order (Order): Objek pesanan yang akan diproses.

        Returns:
            bool: True jika checkout sukses, False jika gagal.
        """
        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

# ----------------------------------------------------------------------
# === EKSEKUSI DAN PEMBUKTIAN OCP (Langkah 3) ===
# ----------------------------------------------------------------------
if __name__ == "__main__":
    # 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: Menambah Metode Pembayaran QRIS (Tanpa Mengubah CheckoutService)
    # Ini membuktikan Open/Closed Principle: Buka untuk ekstensi, Tutup untuk modifikasi.
    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)
```

**EMAIL : 2411102441242@umkt.ac.id**

**Output:**

```
[Running] python -u "c:\Users\Asus\Downloads\Pertemuan 12\Hasil Langkah 1 dan 2.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 0.194 seconds
```

**validasi_registrasi.py:**

validasi_registrasi.py > ...

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

# ----------------------------------------------------------------------
# === KONFIGURASI AWAL LOGGING ===
# ----------------------------------------------------------------------
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(name)s - %(message)s'
)
# Logger untuk Registrasi
LOGGER = logging.getLogger('registration')

# ----------------------------------------------------------------------
# --- MODEL SEDERHANA ---
# ----------------------------------------------------------------------
@dataclass
class Mahasiswa:
    """Model data sederhana untuk Mahasiswa."""
    nama: str
    total_sks_diambil: int
    lulus_prasyarat: bool
    status_pembayaran: str = "lunas"

# ----------------------------------------------------------------------
# === IMPLEMENTASI SOLID ===
# ----------------------------------------------------------------------

# --- ABSTRAKSI (IValidationRule) ---
class IValidationRule(ABC):
    """
    Kontrak (Interface) yang harus diimplementasikan oleh setiap aturan validasi.
    Memastikan OCP (Open/Closed Principle) terpenuhi.
    """
    @abstractmethod
    def validate(self, mahasiswa: Mahasiswa) -> bool:
        """
        Memvalidasi aturan spesifik pada objek Mahasiswa.

        Args:
            mahasiswa (Mahasiswa): Objek mahasiswa yang akan divalidasi.

        Returns:
            bool: True jika validasi lolos, False jika gagal.
        """
        pass

# --- IMPLEMENTASI KONKRIT (Mematuhi SRP dan sudah ada Logging) ---
class SKSLimitRule(IValidationRule):
    """Aturan validasi batas maksimum SKS (24 SKS)."""
    def validate(self, mahasiswa: Mahasiswa) -> bool:
        if mahasiswa.total_sks_diambil > 24:
            LOGGER.warning(f"❌ Validasi SKS Gagal untuk {mahasiswa.nama}: Melebihi batas 24 SKS.")
            return False
        LOGGER.info(f"✅ Validasi SKS Lolos untuk {mahasiswa.nama}.")
        return True

class PrasyaratRule(IValidationRule):
    """Aturan validasi kelulusan mata kuliah prasyarat."""
    def validate(self, mahasiswa: Mahasiswa) -> bool:
        if not mahasiswa.lulus_prasyarat:
            LOGGER.warning(f"❌ Validasi Prasyarat Gagal untuk {mahasiswa.nama}.")
            return False
        LOGGER.info(f"✅ Validasi Prasyarat Lolos untuk {mahasiswa.nama}.")
        return True
```

**EMAIL : 2411102441242@umkt.ac.id**

```python
# --- KELAS KOORDINATOR (RegistrationValidator) ---
class RegistrationValidator:
    """
    Kelas high-level untuk mengkoordinasi eksekusi aturan validasi registrasi.
    Bergantung pada Abstraksi (IValidationRule) melalui Dependency Injection (DIP).
    """
    def __init__(self, rules: list[IValidationRule]):
        """
        Menginisialisasi validator dengan daftar aturan yang di-inject.

        Args:
            rules (list[IValidationRule]): List aturan validasi yang akan dijalankan.
        """
        self.rules = rules

    def is_valid(self, mahasiswa: Mahasiswa) -> bool:
        """
        Menjalankan semua aturan validasi yang di-inject secara berurutan.

        Args:
            mahasiswa (Mahasiswa): Objek mahasiswa yang akan divalidasi.

        Returns:
            bool: True jika semua validasi Lolos, False jika ada satu yang gagal.
        """
        LOGGER.info(f"\n--- Memulai Validasi Registrasi untuk {mahasiswa.nama} ---")
        for rule in self.rules:
            if not rule.validate(mahasiswa):
                LOGGER.error(f"Registrasi Gagal Total! Gagal pada aturan: {rule.__class__.__name__}")
                return False

        LOGGER.info(f"🎉 Registrasi Sukses! {mahasiswa.nama} disetujui.")
        return True

# ----------------------------------------------------------------------
# === EKSEKUSI DAN PEMBUKTIAN OCP ===
# ----------------------------------------------------------------------
if __name__ == "__main__":

    # SETUP AWAL
    rina = Mahasiswa("Rina", 20, True)
    basic_rules = [SKSLimitRule(), PrasyaratRule()]
    validator_basic = RegistrationValidator(basic_rules)

    print("--- Skenario 1: Validasi Normal (Rina - Lolos) ---")
    validator_basic.is_valid(rina)

    # PEMBUKTIAN OCP: Menambah Aturan Pembayaran
    class PembayaranRule(IValidationRule):
        """Aturan validasi status pembayaran."""
        def validate(self, mahasiswa: Mahasiswa) -> bool:
            if mahasiswa.status_pembayaran != "lunas":
                LOGGER.warning(f"❌ Validasi Pembayaran Gagal untuk {mahasiswa.nama}: Belum lunas.")
                return False
            LOGGER.info(f"✅ Validasi Pembayaran Lolos untuk {mahasiswa.nama}.")
            return True

    budi = Mahasiswa("Budi", 26, True, status_pembayaran="belum_lunas")

    # Menggunakan aturan baru (PembayaranRule) tanpa mengubah RegistrationValidator
    advanced_rules = [SKSLimitRule(), PrasyaratRule(), PembayaranRule()]
    validator_advanced = RegistrationValidator(advanced_rules)

    print("\n--- Skenario 2: Pembuktian OCP (Budi - Gagal) ---")
    validator_advanced.is_valid(budi)
```

**Output:**

```
[Running] python -u "c:\Users\Asus\Downloads\Pertemuan 12\validasi_registrasi.py"
--- Skenario 1: Validasi Normal (Rina - Lolos) ---
2025-12-12 21:17:25,512 - INFO - registration -
--- Memulai Validasi Registrasi untuk Rina ---
2025-12-12 21:17:25,512 - INFO - registration - \u2705 Validasi SKS Lolos untuk Rina.

--- Skenario 2: Pembuktian OCP (Budi - Gagal) ---
2025-12-12 21:17:25,513 - INFO - registration - \u2705 Validasi Prasyarat Lolos untuk Rina.
2025-12-12 21:17:25,513 - INFO - registration - \U0001f389 Registrasi Sukses! Rina disetujui.
2025-12-12 21:17:25,513 - INFO - registration -
--- Memulai Validasi Registrasi untuk Budi ---
2025-12-12 21:17:25,513 - WARNING - registration - \u274c Validasi SKS Gagal untuk Budi: Melebihi batas 24 SKS.
2025-12-12 21:17:25,513 - ERROR - registration - Registrasi Gagal Total! Gagal pada aturan: SKSLimitRule

[Done] exited with code=0 in 0.171 seconds
```

**EMAIL : 2411102441242@umkt.ac.id**

**Studi Kasus: Refactoring Pembayaran & Notifikasi**

Welcome | Hasil Langkah 1 dan 2.py | Studi Kasus Refactoring Pembayaran & Notifikasi.py

Studi Kasus Refactoring Pembayaran & Notifikasi.py > CheckoutService > run_checkout

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

# ------------------------------------------------------------------------
# === KONFIGURASI AWAL (Langkah 2 Logging) ===
# ------------------------------------------------------------------------
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(name)s - %(message)s'
)
LOGGER = logging.getLogger('checkout')

# ------------------------------------------------------------------------
# --- MODEL SEDERHANA ---
# ------------------------------------------------------------------------
@dataclass
class Order:
    """Model data sederhana untuk Order."""
    customer_name: str
    total_price: float
    status: str = "open"

# ------------------------------------------------------------------------
# === IMPLEMENTASI SOLID ===
# ------------------------------------------------------------------------

# --- ABSTRAKSI (Kontrak untuk OCP/DIP) ---
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

# --- IMPLEMENTASI KONKRIT (Mematuhi SRP) ---
class CreditCardProcessor(IPaymentProcessor):
    def process(self, order: Order) -> bool:
        LOGGER.info("Payment: Memproses Kartu Kredit.")
        return True

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

**EMAIL : 2411102441242@umkt.ac.id**

```python
# --- KELAS KOORDINATOR (Mematuhi SRP & DIP) ---
class CheckoutService:
    """
    Kelas high-level untuk mengkoordinasi proses transaksi pembayaran.
    Memisahkan logika pembayaran dan notifikasi (memenuhi SRP).
    """
    def __init__(self, payment_processor: IPaymentProcessor, notifier: INotificationService):
        """
        Args:
            payment_processor (IPaymentProcessor): Implementasi Interface pembayaran.
            notifier (INotificationService): Implementasi Interface notifikasi.
        """
        self.payment_processor = payment_processor
        self.notifier = notifier

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

        Args:
            order (Order): Objek pesanan yang akan diproses.

        Returns:
            bool: True jika checkout sukses, False jika gagal.
        """
        LOGGER.info(f"Memulai checkout untuk {order.customer_name}. Total: {order.total_price}")
        payment_success = self.payment_processor.process(order)

        if payment_success:
            order.status = "paid"
            self.notifier.send(order)
            LOGGER.info("Checkout Sukses. Status pesanan: PAID.")
            return True
        else:
            LOGGER.error("Pembayaran gagal. Transaksi dibatalkan.")
            return False
```

```python
# ----------------------------------------------------------------------
# === EKSEKUSI DAN PEMBUKTIAN OCP (Langkah 3) ===
# ----------------------------------------------------------------------
if __name__ == "__main__":
    # Skenario 1: Credit Card
    andi_order = Order("Andi", 500000)
    email_service = EmailNotifier()
    cc_processor = CreditCardProcessor()

    checkout_cc = CheckoutService(payment_processor=cc_processor, notifier=email_service)
    print("\n--- Skenario 1: Credit Card (Log akan muncul di atas) ---")
    checkout_cc.run_checkout(andi_order)

    # PEMBUKTIAN OCP: Menambah Metode Pembayaran QRIS (Tanpa Mengubah CheckoutService)
    class QrisProcessor(IPaymentProcessor):
        def process(self, order: Order) -> bool:
            LOGGER.info("Payment: Memproses QRIS.")
            return True

    # Skenario 2: QRIS
    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)
```

**EMAIL : 2411102441242@umkt.ac.id**

**Output:**

```
[Running] python -u "c:\Users\Asus\Downloads\Pertemuan 12\Studi Kasus Refactoring Pembayaran & Notifikasi.py"

--- Skenario 1: Credit Card (Log akan muncul di atas) ---
2025-12-12 20:30:49,515 - INFO - checkout - Memulai checkout untuk Andi. Total: 500000
2025-12-12 20:30:49,515 - INFO - checkout - Payment: Memproses Kartu Kredit.
2025-12-12 20:30:49,515 - INFO - checkout - Notif: Mengirim email konfirmasi ke Andi.
2025-12-12 20:30:49,515 - INFO - checkout - Checkout Sukses. Status pesanan: PAID.

--- Skenario 2: Pembuktian OCP (QRIS) ---
2025-12-12 20:30:49,515 - INFO - checkout - Memulai checkout untuk Budi. Total: 100000
2025-12-12 20:30:49,515 - INFO - checkout - Payment: Memproses QRIS.
2025-12-12 20:30:49,515 - INFO - checkout - Notif: Mengirim email konfirmasi ke Budi.
2025-12-12 20:30:49,515 - INFO - checkout - Checkout Sukses. Status pesanan: PAID.

[Done] exited with code=0 in 0.187 seconds
```

**Refleksi:** Penggunaan *Docstring* dengan format terstruktur (seperti Google Style) sangat krusial dalam kolaborasi karena berfungsi sebagai **dokumentasi *on-the-fly***. *Docstring* menjelaskan secara eksplisit apa yang dilakukan oleh setiap class dan method—termasuk deskripsi parameter (Args) dan nilai kembalian (Returns)—tanpa pengembang harus membaca seluruh implementasi kode di dalamnya. Hal ini secara signifikan **mengurangi *cognitive load*** bagi anggota tim yang baru atau yang sedang bekerja di bagian kode yang berbeda. Dengan dokumentasi yang konsisten ini, pengembang dapat memahami kontrak *interface* (IPaymentProcessor atau IValidationRule) dengan cepat, memungkinkan mereka untuk membuat implementasi baru (sesuai OCP) tanpa perlu berkomunikasi secara berlebihan dengan penulis kode aslinya.

**EMAIL : 2411102441242@umkt.ac.id**