LAPORAN PRAKTIKUM PEMOGRAMAN BERORIERNTASI OBJEK (PBO)
PRAKTIKUM 13

2411102441211

Muhammad Nabiel

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

2411102441211@umkt.ac.id

Nama :Muhammad Nabiel

Kelas : B

NIM :2411102441211

B. Link:

https://github.com/mhmmdnabiel24/Pemrograman-Berbasis-Objek/tree/main/P13_B_2411102441211_Muhammad_Nabiel

C. Screenshot

Output:

Memilih opsi 1

```
Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 1
MAIN_APP - INFO -
--- DAFTAR PRODUK ---
MAIN_APP - INFO - [P001] Laptop Gaming - Rp15,000,000
MAIN_APP - INFO - [P002] Mouse Wireless - Rp250,000
MAIN_APP - INFO - [P003] Keyboard Mech - Rp800,000
```

Memilih opsi 2

```
Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 2
Masukkan ID Produk: P002
Jumlah (default 1): 2
SERVICES - INFO - Added 2x Mouse Wireless to cart.
```

Memilih opsi 3

```
Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 3
MAIN_APP - INFO -
Total Belanja: Rp500,000
SERVICES - INFO - Memproses KARTU DEBIT sejumlah: Rp500,000
SERVICES - INFO - Menghubungkan ke Bank... Verifikasi Berhasil!
MAIN_APP - INFO - TRANSAKSI BERHASIL.
MAIN_APP - INFO -
--- STRUK PEMBELIAN ---
MAIN_APP - INFO - Mouse Wireless x2 = Rp500,000
MAIN_APP - INFO - --------------------
MAIN_APP - INFO - TOTAL AKHIR: Rp500,000
MAIN_APP - INFO - --------------------
```

Memilih opsi 4

```
Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 4
MAIN_APP - INFO - Aplikasi dihentikan.
```

Code:

1.models.py

```python
from dataclasses import dataclass
from typing import List

@dataclass
class Product:
    id: str
    name: str
    price: float

@dataclass
class CartItem:
    product: Product
    quantity: int

    @property
    def subtotal(self) -> float:
        """Menghitung subtotal untuk item ini."""
        return self.product.price * self.quantity
```

2. repositories.py

```python
import logging
from models import Product # Wajib diimpor dari models.py

LOGGER = logging.getLogger('REPOSITORY')

class ProductRepository:
    """Mengambil data produk (simulasi database)."""

    def __init__(self):
        # Data hardcoded untuk simulasi:
        self._products = {
            "P001": Product(id="P001", name="Laptop Gaming", price=15000000),
            "P002": Product(id="P002", name="Mouse Wireless", price=250000),
            "P003": Product(id="P003", name="Keyboard Mech", price=800000),
        }
        LOGGER.info("ProductRepository initialized with 3 products.")

    def get_all(self) -> list[Product]:
        """Mengambil semua produk yang tersedia."""
        return list(self._products.values())

    def get_by_id(self, product_id: str) -> Product | None:
        """Mencari produk berdasarkan ID."""
        return self._products.get(product_id)
```

3. Services.py

```python
import logging
from abc import ABC, abstractmethod
from typing import List
from models import Product, CartItem # Wajib diimpor dari models.py

LOGGER = logging.getLogger('SERVICES')

# --- INTERFACE PEMBAYARAN (Diperlukan untuk DIP/OCP) ---
class IPaymentProcessor(ABC):
    @abstractmethod
    def process(self, amount: float) -> bool:
        pass

# --- IMPLEMENTASI PEMBAYARAN TUNAI ---
class CashPayment(IPaymentProcessor):
    def process(self, amount: float) -> bool:
        LOGGER.info(f"Menerima TUNAI sejumlah: Rp{amount:,.0f}")
        return True

# --- IMPLEMENTASI PEMBAYARAN KARTU DEBIT (Challenge Sesi 13) ---
class DebitCardPayment(IPaymentProcessor):
    def process(self, amount: float) -> bool:
        # Simulasi proses verifikasi kartu debit
        LOGGER.info(f"Memproses KARTU DEBIT sejumlah: Rp{amount:,.0f}")
        LOGGER.info("Menghubungkan ke Bank... Verifikasi Berhasil!")
        return True

# --- SERVICE KERANJANG BELANJA (Logika Inti Bisnis) ---
class ShoppingCart:
    """Mengelola item, kuantitas, dan total harga pesanan (SRP)."""

    def __init__(self):
        self._items: dict[str, CartItem] = {}

    def add_item(self, product: Product, quantity: int = 1):
        if product.id in self._items:
            self._items[product.id].quantity += quantity
        else:
            self._items[product.id] = CartItem(product=product, quantity=quantity)
        LOGGER.info(f"Added {quantity}x {product.name} to cart.")

    def get_items(self) -> List[CartItem]:
        return list(self._items.values())

    @property
    def total_price(self) -> float:
        return sum(item.subtotal for item in self._items.values())
```

4. Main_app.py

```python
import logging
from repositories import ProductRepository
from services import IPaymentProcessor, ShoppingCart, CashPayment
from models import Product # Diperlukan untuk type hint

LOGGER = logging.getLogger('MAIN_APP')

class PosApp:
    """Kelas Orchestrator (Aplikasi Utama). Hanya mengkoordinasi flow dan menerapkan DI."""

    def __init__(self, repository: ProductRepository, payment_processor: IPaymentProcessor):
        self.repository = repository
        self.payment_processor = payment_processor
        self.cart = ShoppingCart()
        LOGGER.info("POS Application Initialized.")

    def _display_menu(self):
        LOGGER.info("\n--- DAFTAR PRODUK ---")
        for p in self.repository.get_all():
            LOGGER.info(f"[{p.id}] {p.name} - Rp{p.price:,.0f}")

    def _handle_add_item(self):
        product_id = input("Masukkan ID Produk: ").strip().upper()
        product = self.repository.get_by_id(product_id)
        if not product:
            LOGGER.warning("Produk tidak ditemukan.")
            return

        try:
            qty_input = input("Jumlah (default 1): ")
            quantity = int(qty_input) if qty_input else 1
            if quantity <= 0: raise ValueError
            self.cart.add_item(product, quantity)
        except ValueError:
            LOGGER.error("Jumlah tidak valid.")

    def _handle_checkout(self):
        total = self.cart.total_price
        if total == 0:
            LOGGER.warning("Keranjang kosong.")
            return

        LOGGER.info(f"\nTotal Belanja: Rp{total:,.0f}")
        success = self.payment_processor.process(total)
        if success:
            LOGGER.info("TRANSAKSI BERHASIL.")
            self._print_receipt()
            self.cart = ShoppingCart() # Reset cart
        else:
            LOGGER.error("TRANSAKSI GAGAL.")

    def _print_receipt(self):
        LOGGER.info("\n--- STRUK PEMBELIAN ---")
        for item in self.cart.get_items():
            LOGGER.info(f"{item.product.name} x{item.quantity} = Rp{item.subtotal:,.0f}")
        LOGGER.info("-" * 20)
        LOGGER.info(f"TOTAL AKHIR: Rp{self.cart.total_price:,.0f}")
        LOGGER.info("-" * 20)

# TITIK MASUK UTAMA (Orchestration)
if __name__ == "__main__":
    # Setup Logging
    logging.basicConfig(level=logging.INFO, format='%(name)s - %(levelname)s - %(message)s')

    # 1. Instantiate Lapisan Data
    repo = ProductRepository()

    # 2. CHALLENGE: Mengganti CashPayment dengan DebitCardPayment
    # Perhatikan bahwa kita hanya mengubah baris ini saja
    from services import DebitCardPayment
    payment_method = DebitCardPayment()

    # 3. Inject Dependencies (PosApp tetap menggunakan payment_method yang baru)
    # Ini membuktikan fleksibilitas sistem karena tidak mengubah isi class PosApp
    app = PosApp(repository=repo, payment_processor=payment_method)


    # Loop CLI untuk interaksi
    while True:
        print("\nMenu:")
        print("1. Tampilkan Produk")
        print("2. Tambah ke Keranjang")
        print("3. Checkout")
        print("4. Keluar")

        choice = input("Pilih opsi (1-4): ")

        if choice == "1":
            app._display_menu()
        elif choice == "2":
            app._handle_add_item()
        elif choice == "3":
            app._handle_checkout()
        elif choice == "4":
            LOGGER.info("Aplikasi dihentikan.")
            break
        else:
            LOGGER.warning("Pilihan tidak valid.")
```

Refleksi Singkat

Berdasarkan modul tersebut, berikut adalah poin-poin refleksi yang bisa Anda masukkan ke laporan:

1. **Penerapan Dependency Injection (DI):** PosApp tidak lagi membuat objek pembayarannya sendiri (misal: self.payment = CashPayment()), melainkan "menerimanya" melalui constructor.
2. **Keuntungan OCP (Open-Closed Principle):** Kita bisa menambah fitur pembayaran baru (seperti Debit Card) tanpa mengubah kode utama di PosApp. Kelas tersebut tetap "tertutup" dari modifikasi tapi "terbuka" untuk pengembangan fungsionalitas.
3. **Struktur Arsitektur:** Dengan memisahkan Model, Repository, dan Service, kode menjadi lebih rapi, mudah diuji secara terpisah, dan siap untuk dikembangkan menjadi proyek yang lebih besar