LAPORAN PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK (PBO)

PRAKTIKUM 1

2411102441136

Zalfa Faris Ibrahim

FAKULTAS SAINS DAN TEKNOLOGI

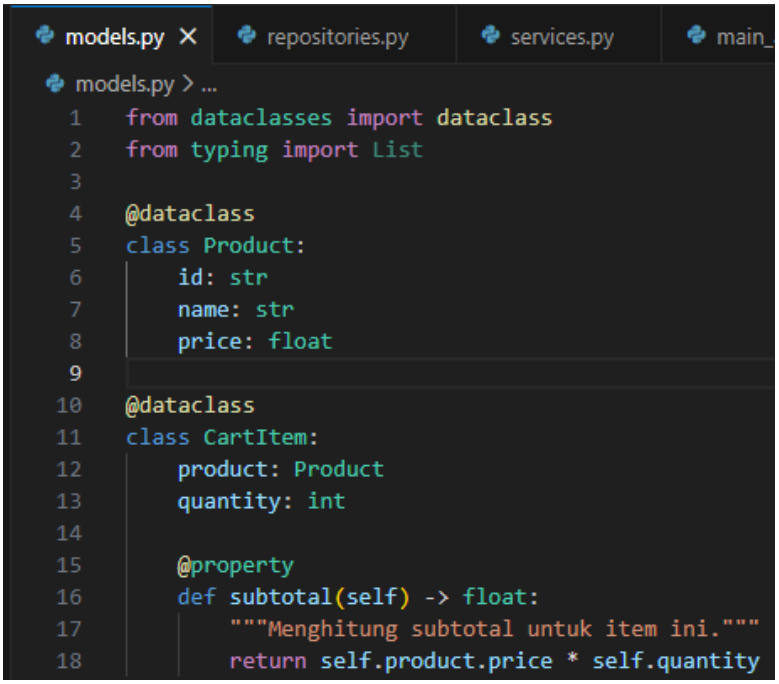
PROGRAM STUDI S1 TEKNIK INFORMATIKA

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

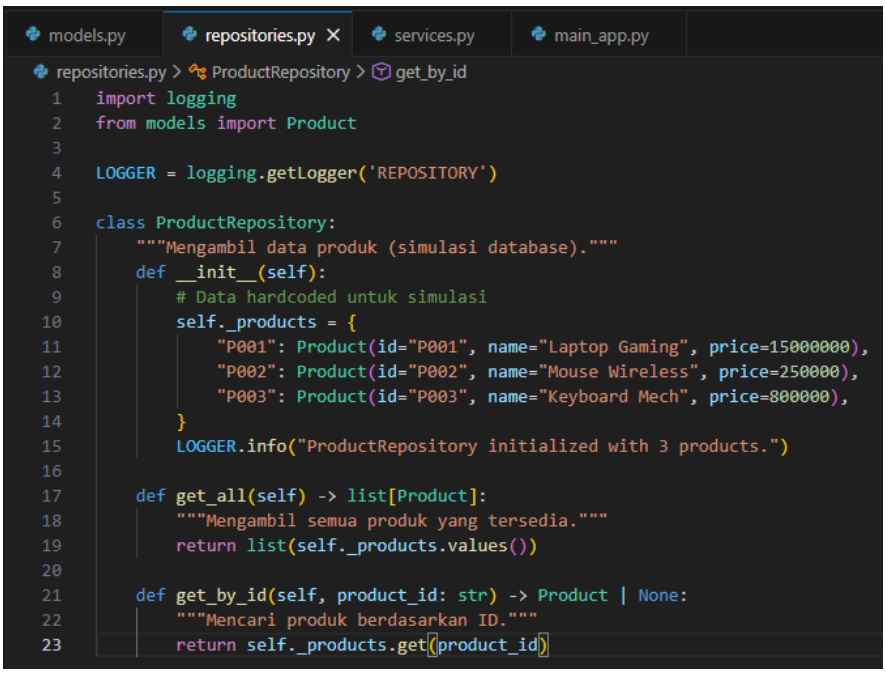
EMAIL : 2411102441136@umkt.ac.id

C. Langkah-langkah Praktikum

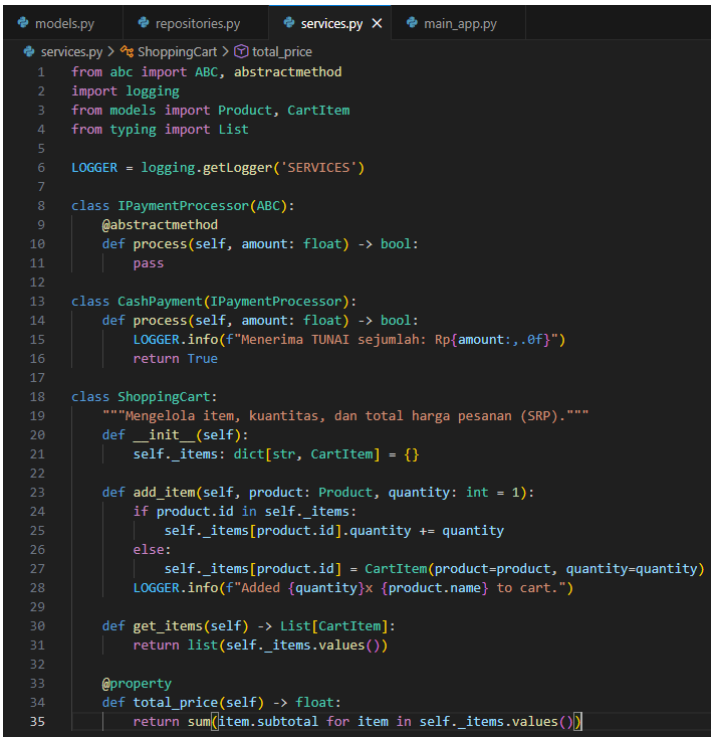
Models.py



Repositorles.py

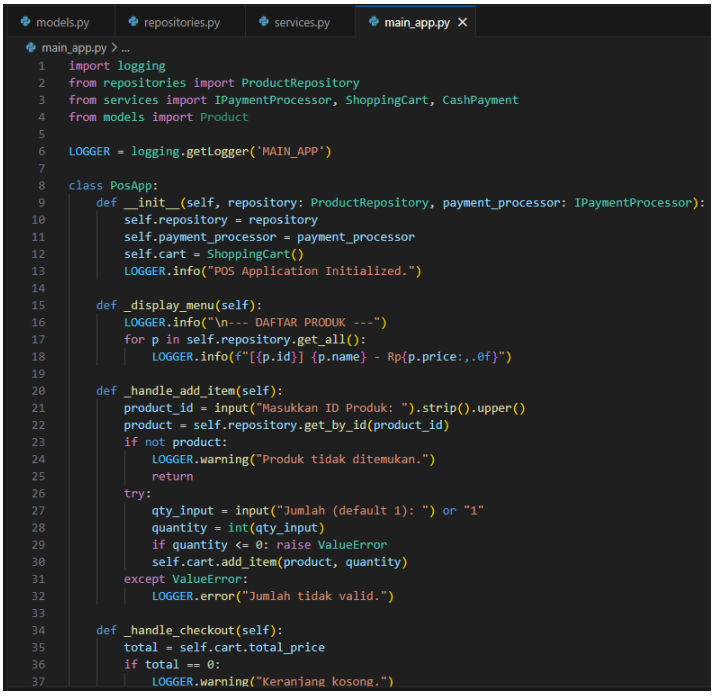


Services.py




```python
from abc import ABC, abstractmethod
import logging
from models import Product, CartItem
from typing import List

LOGGER = logging.getLogger('SERVICES')

class IPaymentProcessor(ABC):
    @abstractmethod
    def process(self, amount: float) -> bool:
        pass

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

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())
```

Main_app.py




```python
import logging
from repositories import ProductRepository
from services import IPaymentProcessor, ShoppingCart, CashPayment
from models import Product

LOGGER = logging.getLogger('MAIN_APP')

class PosApp:
    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): ") or "1"
            quantity = int(qty_input)
            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.")
```

EMAIL : 2411102441136@umkt.ac.id

```python
            return
        LOGGER.info(f"\nTotal Belanja: Rp{total:,.0f}")
        if self.payment_processor.process(total):
            LOGGER.info("TRANSAKSI BERHASIL.")
            self._print_receipt()
            self.cart = ShoppingCart()
        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(f"TOTAL AKHIR: Rp{self.cart.total_price:,.0f}")

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format='%(name)s - %(levelname)s - %(message)s')
    repo = ProductRepository()
    app = PosApp(repository=repo, payment_processor=CashPayment())

    while True:
        print("\n1. Tampilkan Produk\n2. Tambah ke Keranjang\n3. Checkout\n4. Keluar")
        choice = input("Pilih opsi: ")
        if choice == "1": app._display_menu()
        elif choice == "2": app._handle_add_item()
        elif choice == "3": app._handle_checkout()
        elif choice == "4": break
```

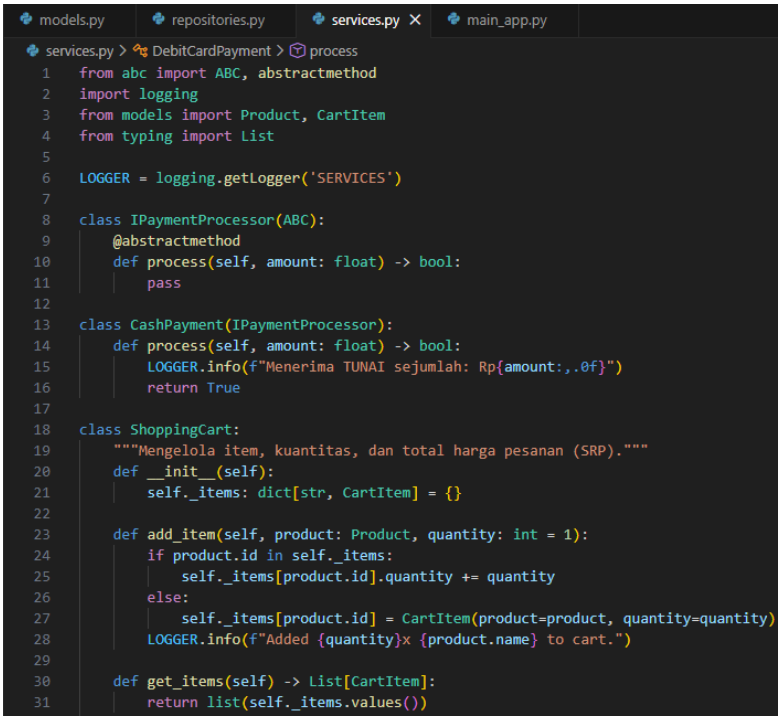
Output

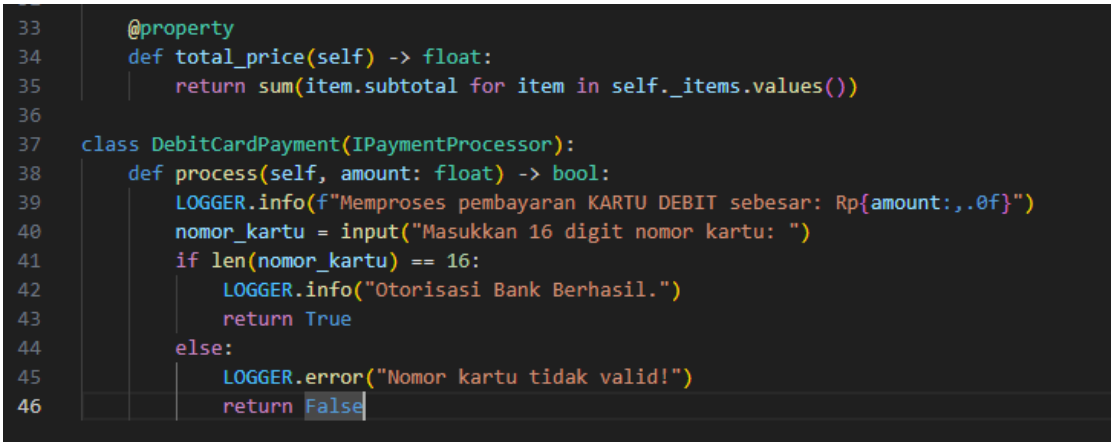
```
PS C:\xampp\htdocs\tugas pbo\Praktikum 13> python main_app.py
REPOSITORY - INFO - ProductRepository initialized with 3 products.
MAIN_APP - INFO - POS Application Initialized.

1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi: 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

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

1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi: 3
MAIN_APP - INFO -
Total Belanja: Rp500,000
SERVICES - INFO - Menerima TUNAI sejumlah: Rp500,000
MAIN_APP - INFO - TRANSAKSI BERHASIL.
MAIN_APP - INFO -
--- STRUK PEMBELIAN ---
MAIN_APP - INFO -  Mouse Wireless x2 = Rp500,000
MAIN_APP - INFO - TOTAL AKHIR: Rp500,000

1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi: 4
PS C:\xampp\htdocs\tugas pbo\Praktikum 13>
```

EMAIL : 2411102441136@umkt.ac.id

D. Tugas mandiri

Services.py





Main_app.py

main_app.py > ...

```python
import logging
from repositories import ProductRepository
from services import IPaymentProcessor, ShoppingCart, CashPayment, DebitCardPayment
from models import Product

LOGGER = logging.getLogger('MAIN_APP')

class PosApp:
    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): ") or "1"
            quantity = int(qty_input)
            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}")
        if self.payment_processor.process(total):
            LOGGER.info("TRANSAKSI BERHASIL.")
            self._print_receipt()
            self.cart = ShoppingCart()
```

models.py | repositories.py | services.py | main_app.py

main_app.py > ...

```python
class PosApp:
    def _handle_checkout(self):
            LOGGER.info("TRANSAKSI BERHASIL.")
            self._print_receipt()
            self.cart = ShoppingCart()
        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(f"TOTAL AKHIR: Rp{self.cart.total_price:,.0f}")



if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format='%(name)s - %(levelname)s - %(message)s')

    repo = ProductRepository()

    payment_method = DebitCardPayment()


    app = PosApp(repository=repo, payment_processor=payment_method)

    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.")
```

EMAIL : 2411102441136@umkt.ac.id

Output

```
PS C:\xampp\htdocs\tugas pbo\Praktikum 13> python main_app.py
REPOSITORY - INFO - ProductRepository initialized with 3 products.
MAIN_APP - INFO - POS Application Initialized.

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

Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 2
Masukkan ID Produk: P001
Jumlah (default 1): 3
SERVICES - INFO - Added 3x Laptop Gaming to cart.

Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 3
MAIN_APP - INFO -
Total Belanja: Rp45,000,000
SERVICES - INFO - Memproses pembayaran KARTU DEBIT sebesar: Rp45,000,000
Masukkan 16 digit nomor kartu: 1234567891212345
SERVICES - INFO - Otorisasi Bank Berhasil.
MAIN_APP - INFO - TRANSAKSI BERHASIL.
MAIN_APP - INFO -
--- STRUK PEMBELIAN ---
MAIN_APP - INFO -  Laptop Gaming x3 = Rp45,000,000
MAIN_APP - INFO - TOTAL AKHIR: Rp45,000,000

Menu:
1. Tampilkan Produk
2. Tambah ke Keranjang
3. Checkout
4. Keluar
Pilih opsi (1-4): 4
MAIN_APP - INFO - Aplikasi dihentikan.
PS C:\xampp\htdocs\tugas pbo\Praktikum 13>
```

Refleksi: Mengapa *Dependency Injection* Membuat Sistem Lebih Baik?

Sesuai dengan bagian **Refleksi** pada gambar, penggunaan *Dependency Injection* (DI) pada PosApp memberikan keuntungan sebagai berikut:

1. **Open/Closed Principle (OCP):** Kelas PosApp bersifat "tertutup" untuk modifikasi tetapi "terbuka" untuk perluasan. Anda bisa menambah 10 metode pembayaran baru (QRIS, Kredit, dll) tanpa perlu menyentuh satu baris kode pun di dalam PosApp.
2. **Decoupling:** PosApp tidak lagi bergantung pada implementasi spesifik (seperti CashPayment), melainkan bergantung pada abstraksi (IPaymentProcessor).
3. **Kemudahan Testing:** Anda bisa dengan mudah memasukkan "metode pembayaran palsu" (mocking) saat melakukan pengujian otomatis tanpa harus melakukan transaksi sungguhan.