Framework Layer Architecture Assessment of Learning
Kelompok 3
Nama & Nim Kelompok:

- **DIATRA AWANANDA**
  2702296792
  Computer Science
- **ILYASA FATHUR RAHMAN**
  2502030372
  Computer Science
- **REYNOLD SUHERMAN**
  2702314920
  Computer Science
- **KHANZA ALMAS SHAFIRA**
  2502055444
  Computer Science

Case Study Title

SocierMart – Inventory & Shopping Cart System with Java Design Patterns

---

Introduction

SocierMart adalah aplikasi console berbasis Java yang dirancang untuk mengelola stok barang dan transaksi pembelian secara interaktif. Sistem ini dikembangkan sebagai solusi untuk permasalahan manajemen inventori dan transaksi sederhana pada toko atau e-commerce skala kecil-menengah. Tujuan utama proyek ini adalah menerapkan arsitektur berlapis dan beberapa design pattern utama agar kode lebih modular, mudah dipelihara, scalable, dan mudah dikembangkan di masa mendatang.

Methodology

Analisis Kebutuhan

Aplikasi SocierMart membutuhkan fitur-fitur utama sebagai berikut:

- Manajemen katalog produk dan stok: Menambah, melihat, dan menghapus produk dari katalog.
- Manajemen keranjang belanja: Menambah dan menghapus barang dari keranjang, serta melihat isi keranjang.
- Proses pemesanan dan pembayaran: Membuat pesanan dan memilih metode pembayaran.
- Validasi stok: Memastikan stok tersedia sebelum transaksi dilakukan.

Pemilihan Design Pattern

Berdasarkan kebutuhan di atas, pola desain berikut dipilih dan diimplementasikan:

Proses Pemilihan Design Pattern

Berdasarkan kebutuhan di atas, tim memilih beberapa design pattern berikut untuk diimplementasikan:

1. Facade Pattern
    - Penjelasan:
      Facade Pattern digunakan untuk menyederhanakan interaksi antara UI (Main.java) dengan subsistem yang kompleks (Service, ServicePesan, ServicePembayaran). Dengan satu interface utama (`ShoppingFacade`), seluruh operasi bisnis dapat dipanggil dari satu titik, sehingga kode di layer presentasi menjadi lebih bersih dan mudah dipahami.
    - Alasan:
      Mengurangi tight coupling antara UI dan subsistem, meningkatkan maintainability, dan memudahkan pengembangan fitur baru tanpa mengubah kode di UI.
2. Singleton Pattern
    - Penjelasan:
      Singleton Pattern memastikan hanya ada satu instance dari keranjang belanja (ShoppingCart) selama satu sesi aplikasi.
    - Alasan:
      Menjamin konsistensi data keranjang dan mencegah duplikasi data pada saat transaksi berlangsung.
3. Strategy Pattern
    - Penjelasan:
      Strategy Pattern digunakan untuk memungkinkan sistem memilih strategi perhitungan harga atau diskon yang berbeda-beda secara dinamis.
    - Alasan:
      Memberikan fleksibilitas dalam menambah atau mengubah strategi diskon/metode pembayaran tanpa mengubah struktur utama kode.

4. Command Pattern
    - Penjelasan:
      Command Pattern digunakan untuk mengenkapsulasi aksi seperti tambah/hapus barang sebagai objek perintah yang dapat dieksekusi, dibatalkan, atau diulang.
    - Alasan:
      Memudahkan pengelolaan aksi user, memungkinkan fitur undo/redo pada aksi keranjang, serta meningkatkan extensibility.
5. State Pattern
    - Penjelasan:
      State Pattern mengelola status dari keranjang belanja, misal status "kosong", "diisi", atau "checkout".
    - Alasan:
      Memisahkan logika perubahan status dan transisi, sehingga kode lebih terstruktur dan mudah dikembangkan.

Formulated Solution Design Patterns

Facade Pattern

Penjelasan Implementasi:
Pada SocierMart, seluruh interaksi antara user interface (`Main.java`) dan subsistem yang kompleks (seperti pengelolaan katalog, keranjang, pemesanan, dan pembayaran) dilakukan melalui satu kelas utama, yaitu `ShoppingFacade`. Kelas ini menyediakan method-method sederhana seperti `addToCart`, `removeFromCart`, `addProductToCatalogAndCart`, `getCart`, `getProductCatalog`, `placeOrder`, `selectPaymentMethod`, dan `pay`.
Setiap method di `ShoppingFacade` akan meneruskan permintaan ke subsistem terkait (misal: `Service`, `ServicePesan`, `ServicePembayaran`), sehingga UI tidak perlu tahu detail implementasi di balik layar.
Alasan Penggunaan:

- Menyederhanakan interaksi UI dengan subsistem, sehingga kode di layer presentasi lebih bersih dan mudah dipahami.
- Mengurangi tight coupling antara UI dan subsistem, sehingga perubahan di subsistem tidak langsung memengaruhi UI.
- Memudahkan pengembangan dan penambahan fitur baru tanpa mengubah kode di UI.

Singleton Pattern

Penjelasan Implementasi:
Kelas `ShoppingCart` diimplementasikan sebagai Singleton. Artinya, hanya ada satu instance keranjang belanja yang digunakan selama aplikasi berjalan.
Konstruktor kelas ini dibuat private, dan akses instance dilakukan melalui method statis `getInstance()`.
Alasan Penggunaan:

- Menjamin konsistensi data keranjang selama sesi aplikasi.
- Menghindari duplikasi data keranjang yang bisa menyebabkan inkonsistensi transaksi.

Strategy Pattern

Penjelasan Implementasi:
SocierMart memungkinkan penggunaan berbagai strategi perhitungan harga atau diskon.
Sistem mendefinisikan interface `DiscountStrategy` dengan method `apply(double total)`. Implementasi strategi bisa berupa `NoDiscount`, `PercentageDiscount`, dll.
Saat checkout, strategi diskon yang aktif akan diterapkan pada total harga keranjang.
Alasan Penggunaan:

- Memberikan fleksibilitas dalam menambah atau mengubah strategi diskon/metode pembayaran tanpa mengubah struktur utama kode.
- Memudahkan pengembangan fitur promosi atau diskon di masa depan.

Command Pattern

Penjelasan Implementasi:
Setiap aksi pada keranjang (seperti tambah/hapus barang) di-enkapsulasi sebagai objek command, misal `AddItemCommand`, `RemoveItemCommand`.
Setiap command mengimplementasikan interface `Command` dengan method `execute()`.
Dengan pola ini, aksi user dapat dikelola sebagai objek yang dapat dieksekusi, dibatalkan (undo), atau diulang (redo).
Alasan Penggunaan:

- Memudahkan pengelolaan aksi user, terutama jika ingin menambah fitur undo/redo.
- Memisahkan logika aksi dari UI, sehingga kode lebih modular dan mudah diuji.

State Pattern

Penjelasan Implementasi:
State Pattern digunakan untuk mengelola status keranjang belanja, misal status "kosong", "diisi", atau "checkout".
Sistem mendefinisikan interface `CartState` dan implementasi seperti `EmptyState`, `FilledState`, `CheckedOutState`.
Keranjang akan berpindah state sesuai aksi user (misal: setelah checkout, state menjadi `CheckedOutState`).
Alasan Penggunaan:

- Memisahkan logika perubahan status dan transisi, sehingga kode lebih terstruktur dan mudah dikembangkan.
- Memudahkan penambahan aturan baru terkait status keranjang di masa depan.

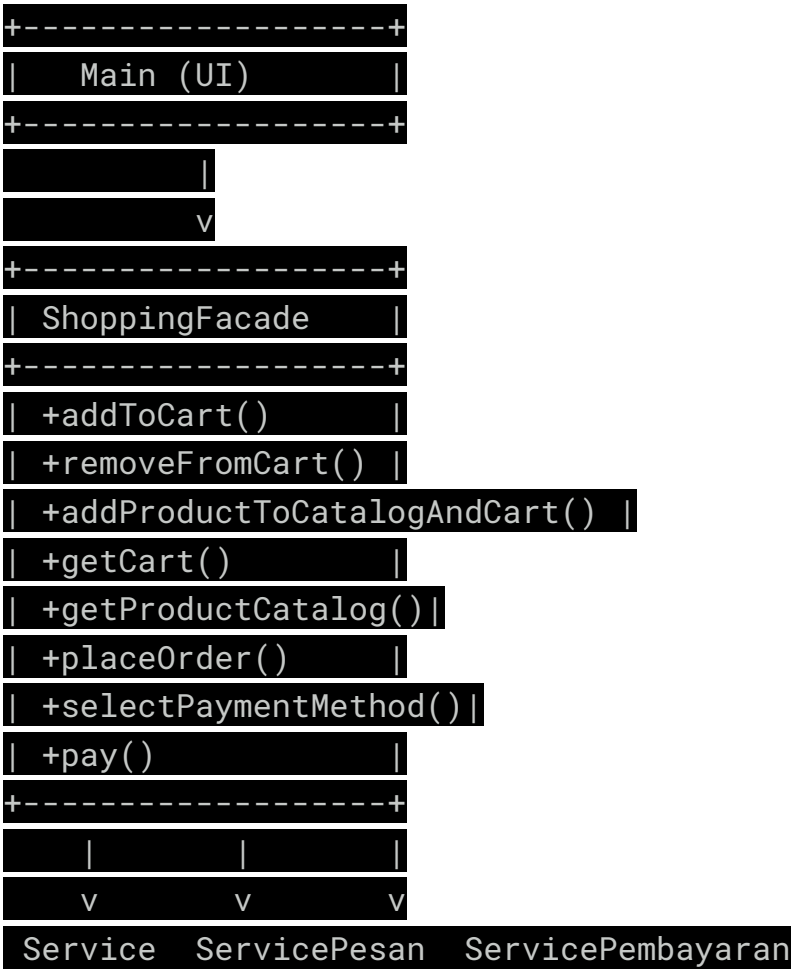
Dengan menerapkan kelima design pattern ini, SocierMart menjadi:

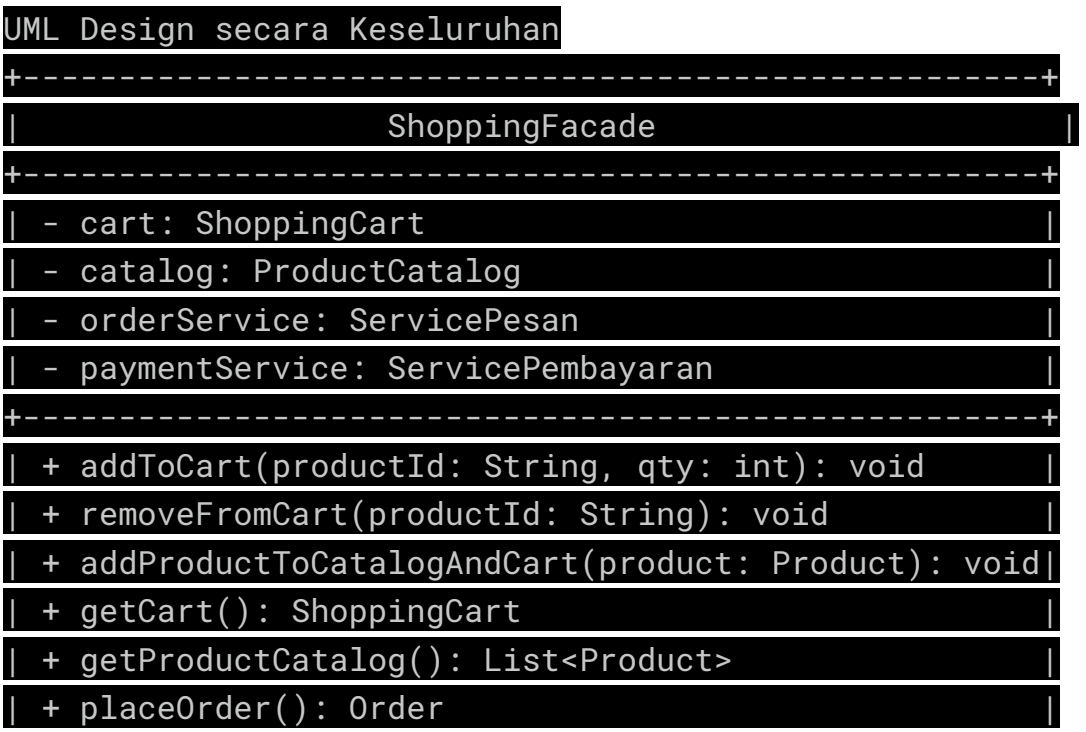
- Lebih modular dan maintainable.
- Mudah dikembangkan dan diuji.
- Siap untuk penambahan fitur baru tanpa refactor besar-besaran.

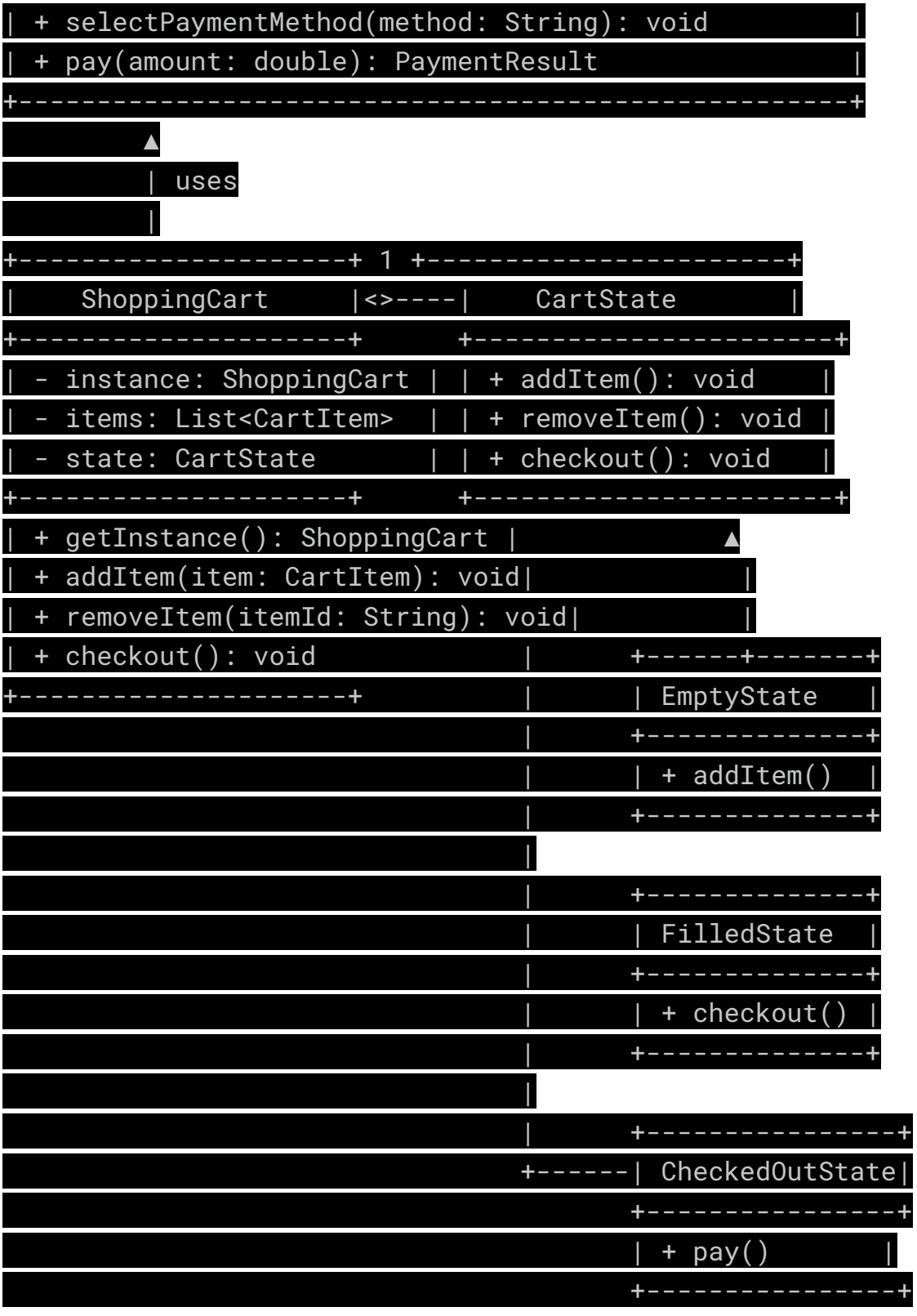
Struktur Layer dan Implementasi

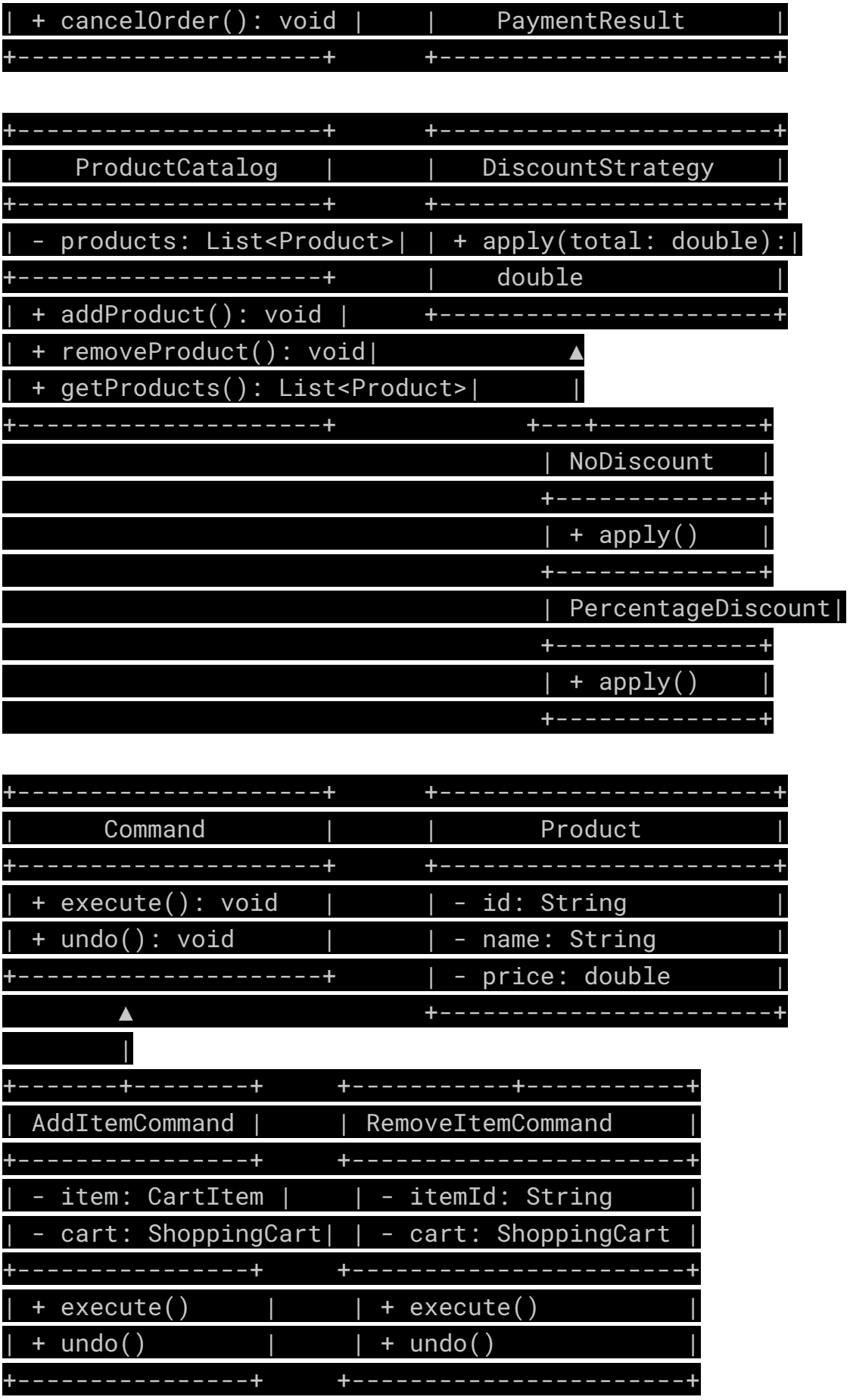
Layer	**Design Pattern**	**Implementasi Kelas**
Presentation Layer	-	Main.java
Business Logic Layer	Facade, Service	ShoppingFacade, Service, ServicePesan, ServicePembayaran
Data Access Layer	- (langsung via Map)	Map<Integer, Product> untuk katalog dan keranjang

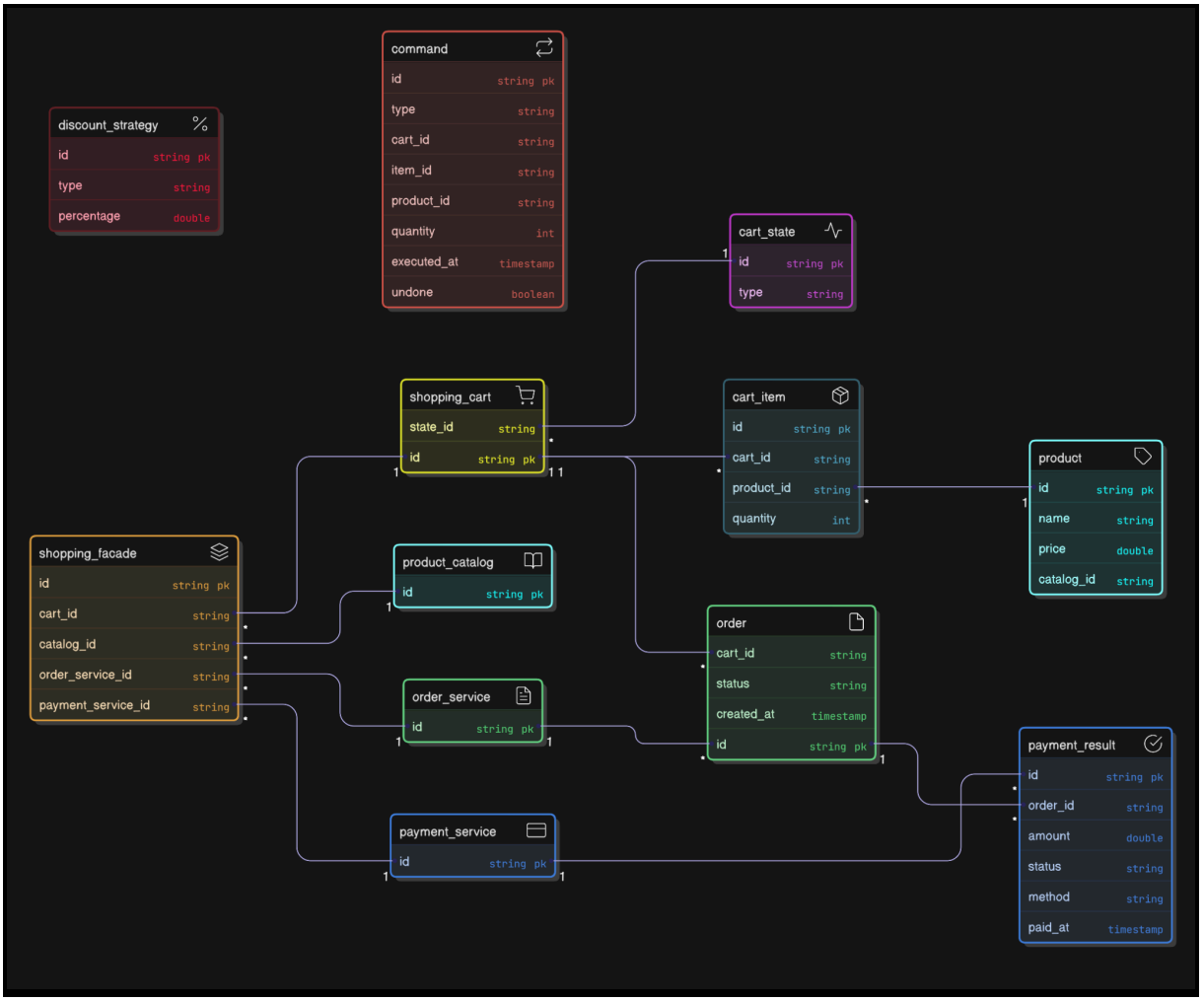
Diagram (UML)


Main (UI)
ShoppingFacade
+addToCart()
+removeFromCart()
+addProductToCatalogAndCart()
+getCart()
+getProductCatalog()
+placeOrder()
+selectPaymentMethod()
+pay()
Service
ServicePesan
ServicePembayaran


UML Design secara Keseluruhan
ShoppingFacade
- cart: ShoppingCart
- catalog: ProductCatalog
- orderService: ServicePesan
- paymentService: ServicePembayaran
+ addToCart(productId: String, qty: int): void
+ removeFromCart(productId: String): void
+ addProductToCatalogAndCart(product: Product): void
+ getCart(): ShoppingCart
+ getProductCatalog(): List<Product>
+ placeOrder(): Order


+ selectPaymentMethod(method: String): void
+ pay(amount: double): PaymentResult
uses
1
ShoppingCart
- instance: ShoppingCart
- items: List<CartItem>
- state: CartState
+ getInstance(): ShoppingCart
+ addItem(item: CartItem): void
+ removeItem(itemId: String): void
+ checkout(): void
CartState
+ addItem(): void
+ removeItem(): void
+ checkout(): void
EmptyState
+ addItem()
FilledState
+ checkout()
CheckedOutState
+ pay()
OrderService
+ createOrder(): Order
PaymentService
+ processPayment():


+ cancelOrder(): void
PaymentResult
ProductCatalog
- products: List<Product>
+ addProduct(): void
+ removeProduct(): void
+ getProducts(): List<Product>
DiscountStrategy
+ apply(total: double):
double
NoDiscount
+ apply()
PercentageDiscount
+ apply()
Command
+ execute(): void
+ undo(): void
Product
- id: String
- name: String
- price: double
AddItemCommand
- item: CartItem
- cart: ShoppingCart
+ execute()
+ undo()
RemoveItemCommand
- itemId: String
- cart: ShoppingCart
+ execute()
+ undo()



Implementation in Java

a. Technologies Used

- Java SE (console application)
- Struktur package: `Main`, `Model`, `facade`

b. Step-by-Step Implementation

1. Presentation Layer (Main.java)

- Menampilkan menu interaktif untuk user.
- Menerima input user untuk berbagai operasi (tambah/hapus barang, refill stok, lihat katalog, checkout, pembayaran).
- Semua operasi dilakukan melalui objek `ShoppingFacade`.

2. Business Logic Layer

- ShoppingFacade:
  Mengelola seluruh proses bisnis, menjadi satu-satunya interface ke subsistem.
- Service:
  Menangani logika keranjang dan katalog (tambah/hapus barang, cek stok).
- ServicePesan:
  Menangani proses pemesanan (validasi keranjang, hitung total).
- ServicePembayaran:
  Menangani proses pembayaran (pilih metode, proses pembayaran).

3. Data Access Layer

- Menggunakan `Map<Integer, Product>` untuk menyimpan data katalog dan keranjang secara lokal, sehingga mudah diakses dan dimodifikasi sesuai kebutuhan aplikasi.

c. Code Snippets

Contoh method pada ShoppingFacade:
`java`

```
public void addProductToCatalogAndCart(Product product) {
    try {
        int idKey = Integer.parseInt(product.getId());
        productCatalog.put(idKey, product);
        addToCart(idKey);
        System.out.println("Produk " + product.getName() + "
berhasil ditambahkan ke katalog dan keranjang.");
    } catch (NumberFormatException e) {
        System.out.println("ID produk harus berupa angka.");
    }
}
```

Contoh pengecekan stok di menu:

```
if (facade.getProductCatalog().containsKey(addId)) {
    facade.addToCart(addId);
} else {
```

```
    System.out.println("Stok kosong! Produk dengan ID " +
addId + " tidak tersedia.");
}
```

Contoh menampilkan isi keranjang:
`java`

```
facade.getCart().forEach(product -> product.displayProduct());
```

Conclusion

Aplikasi SocierMart berhasil menerapkan arsitektur berlapis dengan pemanfaatan Facade Pattern untuk menyederhanakan interaksi antara UI dan subsistem. Penggunaan struktur data `Map` untuk katalog dan keranjang membuat aplikasi mudah dikembangkan dan diuji. Dengan menu interaktif, aplikasi ini dapat menangani manajemen stok, transaksi, dan pembayaran secara efisien. Design pattern yang dipilih terbukti efektif dalam meningkatkan modularitas, keterbacaan, dan maintainability kode.

References

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Dokumentasi Java SE