LAPORAN PRAKTIKUM MINGGU KE 6

UML & SOLID Design Agri-POS

NAMA : ALVIRA LIBRA RAMADHANI

NIM : 240202851

KELAS : 3IKRA

1. **TUJUAN**

   Praktikum minggu ini bertujuan untuk:

   a. Memodelkan sistem menggunakan **UML** (Use Case, Activity, Sequence, Class).
   b. Menerapkan dan menjelaskan prinsip **SOLID** pada desain arsitektur Agri-POS.
   c. Menyusun dokumentasi desain yang konsisten dan mudah ditelusuri.

2. **DASAR TEORI**

   a. **UML** untuk memvisualisasikan arsitektur dan interaksi sistem.
   b. **Use Case** menggambarkan aktor & fungsionalitas sistem.
   c. **Activity Diagram** menunjukan alur bisnis (flow + alternatif).
   d. **Sequence Diagram** menampilkan pertukaran pesan antar objek dari sudut waktu.
   e. **Class Diagram** memodelkan struktur data dan relasi.
   f. **SOLID**: prinsip desain OOP untuk maintainability & extensibility.

3. **LANGKAH PRAKTIKUM**

   a. Analisis kebutuhan sistem (FR & NFR).
   b. Buat Use Case Diagram (aktor: Admin, Kasir, Payment Gateway).
   c. Buat Activity Diagram untuk proses **Checkout** (swimlane Kasir / Sistem / Payment Gateway).
   d. Buat Sequence Diagram untuk skenario pembayaran (Tunai & E-Wallet; skenario gagal).
   e. Buat Class Diagram (atribut, method/signature, visibility, multiplicity).
   f. Dokumentasikan penerapan SOLID dan traceability.
   g. Commit incremental: week6-uml-solid: iterasi-N <deskripsi>

4. DIAGRAM UML (Lampiran)
   a. USE CASE DIAGRAM

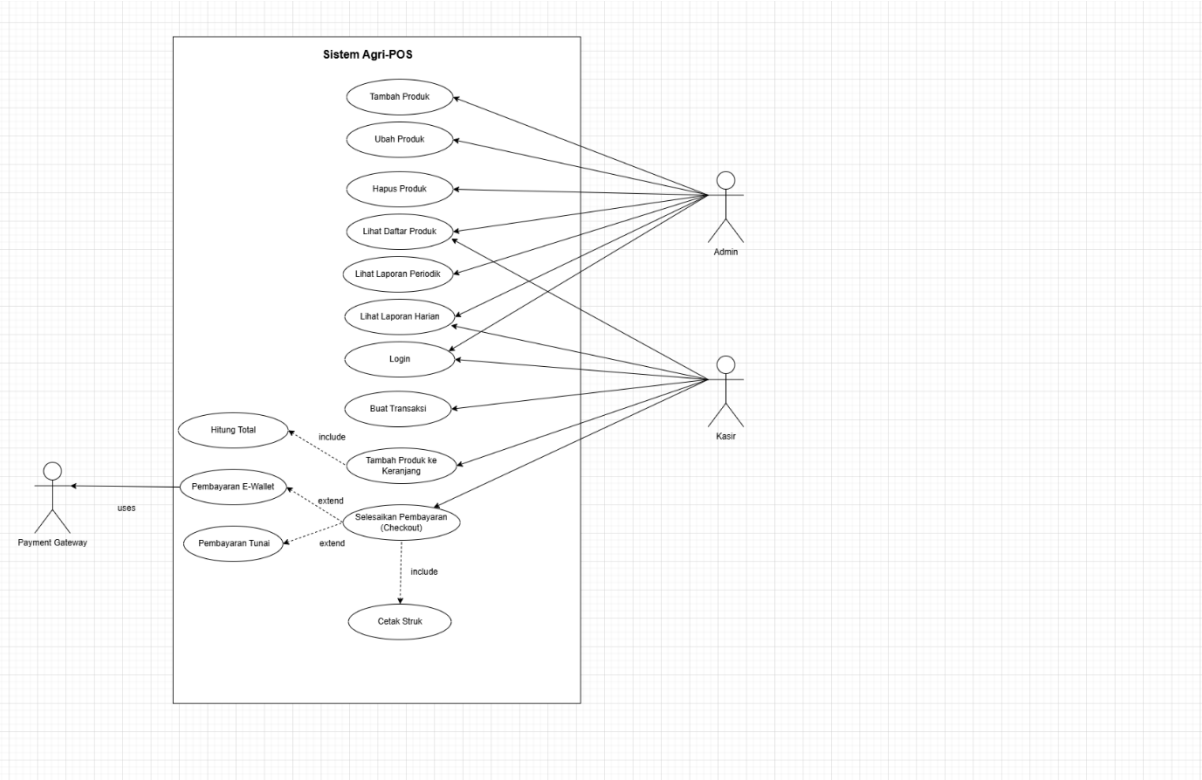


Diagram ini menunjukkan **aktor** yang berinteraksi dengan sistem (Admin, Kasir, dan Payment Gateway) serta **fungsi utama** yang tersedia, yaitu:

- Kelola Produk
- Melakukan Transaksi
- Checkout
- Cetak Struk
- Melihat Laporan
- Login
  Tujuan diagram ini adalah memberikan gambaran umum tentang fitur yang bisa digunakan masing-masing aktor.

b. ACTIVITY DIAGRAM – CHECKOUT

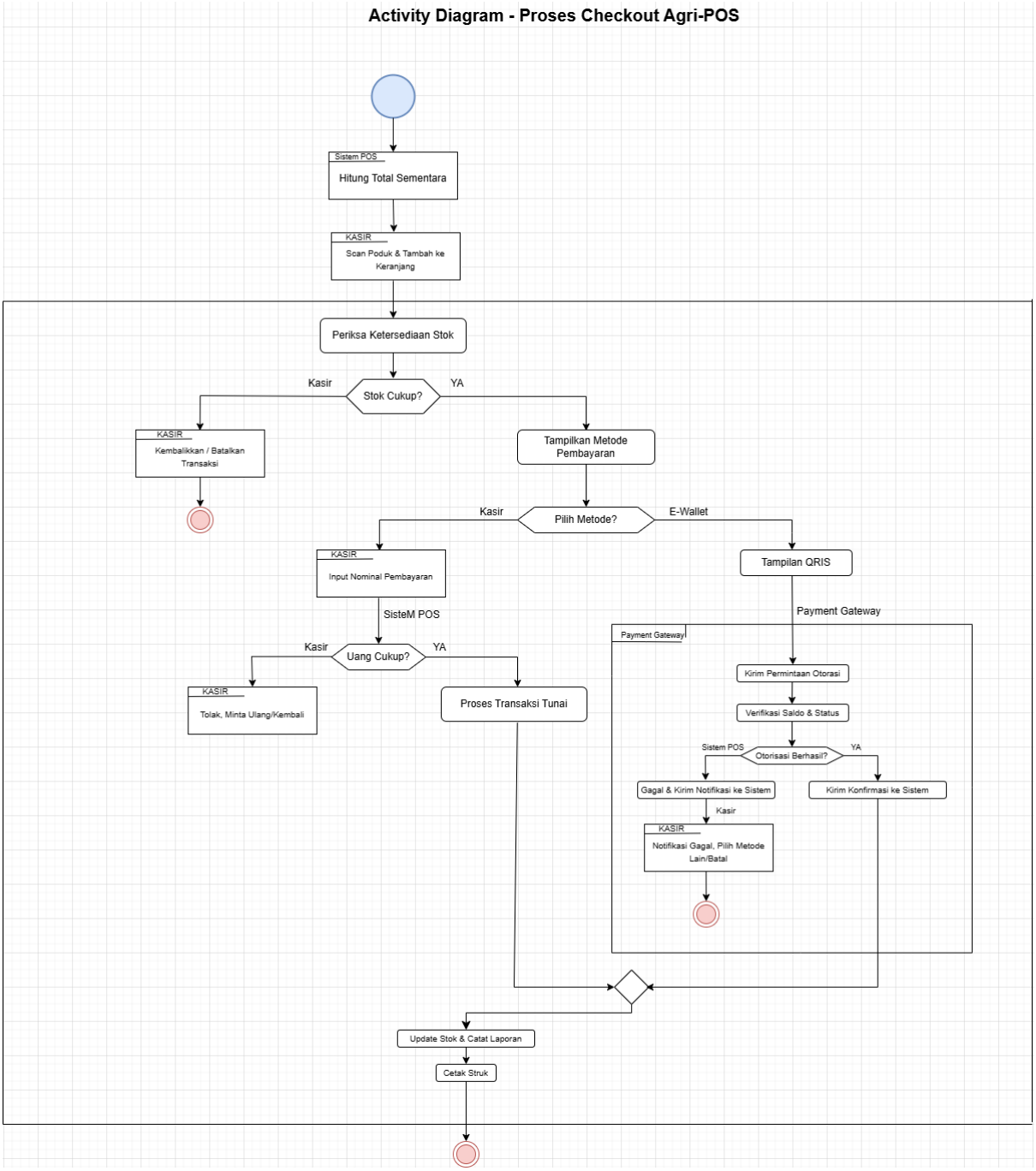
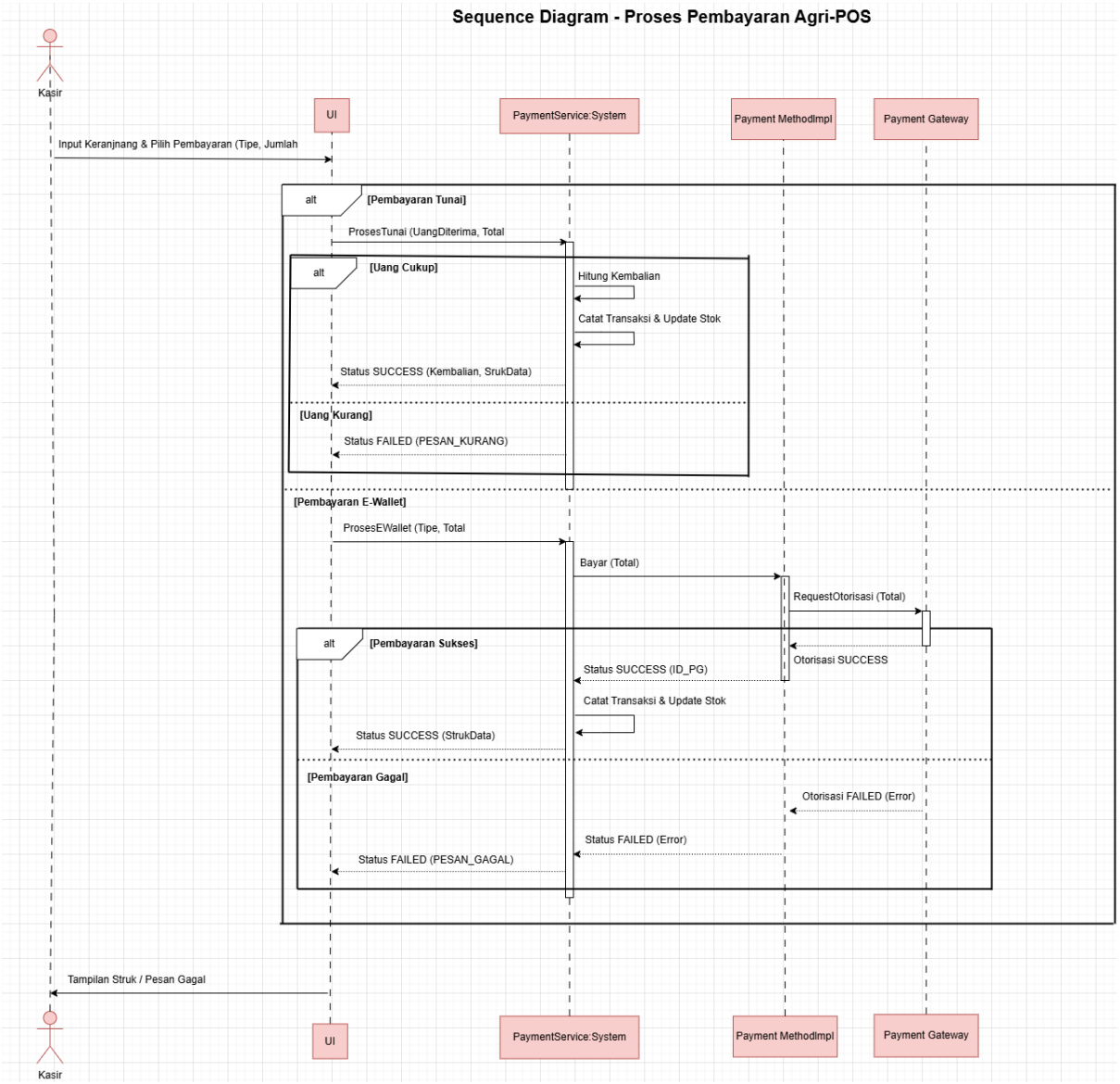


Diagram ini menjelaskan **alur proses checkout**, mulai dari kasir memilih metode pembayaran hingga transaksi selesai.
Terdapat percabangan untuk menangani kondisi:

- Stok produk tidak cukup
- Uang tunai tidak mencukupi
- Saldo e-wallet gagal/berhasil
  Proses berakhir pada **update stok** dan **pencetakan struk**.

c. SEQUENCE DIAGRAM – PEMBAYARAN



Sequence diagram menampilkan **urutan pesan** antar objek selama proses pembayaran berlangsung.
Terdapat dua alur utama:

- **Tunai:** sistem memeriksa kecukupan uang → menghitung kembalian → menyimpan transaksi.
- **E-Wallet:** sistem mengirim permintaan ke Payment Gateway → jika berhasil transaksi disimpan, jika gagal muncul notifikasi.

Diagram ini memperlihatkan interaksi dan waktu proses secara detail.

d. CLASS DIAGRAM

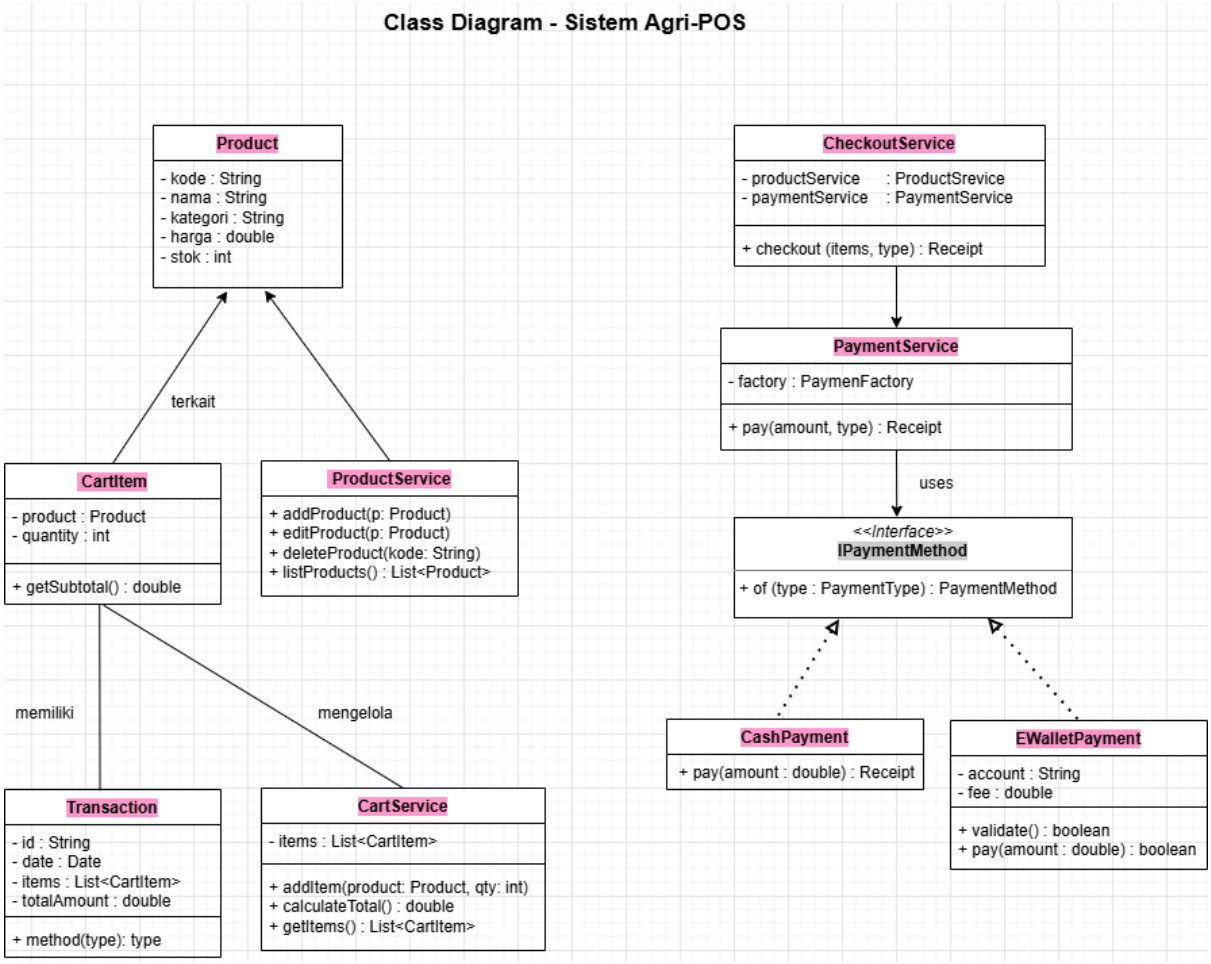


Diagram ini memperlihatkan **struktur kelas** yang digunakan sistem, seperti:

- Product, CartItem, CartService
- Transaction, PaymentService
- Interface IPaymentMethod dan implementasinya (CashPayment, EWalletPayment)
- CheckoutService untuk mengatur proses transaksi
  Diagram juga menunjukkan **atribut, method**, serta **relasi antar kelas** (multiplicity dan visibility), sehingga mudah memahami bagaimana data dan logika dikelola.

5. ANALISIS DAN KETERKAITAN ANTAR DIAGRAM

a. Use Case → memetakan kebutuhan fungsional (sumber awal desain).
b. Activity → menjabarkan alur yang diharapkan oleh Use Case (checkout).
c. Sequence → merealisasikan Activity ke interaksi objek/komponen (timing + error handling).
d. Class → menunjukkan struktur yang diperlukan untuk mengimplementasikan pesan dan aktivitas pada Sequence/Activity.

Konsistensi penamaan dijaga: nama use case, method, dan kelas di semua diagram sama untuk mempermudah traceability.

6. PENERAPAN PRINSIP SOLID

a. S — Single Responsibility Principle
   Setiap komponen memiliki satu tanggung jawab: ProductService untuk CRUD produk; CartService untuk operasi keranjang; PaymentService untuk mengatur proses pembayaran; ReportService untuk laporan.
b. O — Open/Closed Principle
   Sistem terbuka untuk penambahan metode pembayaran baru (mis. QRISPayment, BankTransferPayment) tanpa memodifikasi PaymentService yang sudah ada — cukup menambah class implementasi baru yang mematuhi interface pembayaran.
c. L — Liskov Substitution Principle
   Semua implementasi metode pembayaran (CashPayment, EWalletPayment, dsb.) dapat dipertukarkan di tempat yang menggunakan abstraksi IPaymentMethod tanpa mengubah perilaku yang diharapkan.
d. I — Interface Segregation Principle
   Interface dipisah sesuai kebutuhan. Misal IPaymentMethod hanya berisi pay() dan validate() jika perlu; fitur laporan tidak memaksa service transaksi untuk mengimplementasikan method yang tidak relevan.
e. D — Dependency Inversion Principle
   Komponen tingkat tinggi (CheckoutService, PaymentService) bergantung pada abstraksi (IPaymentMethod, ProductRepository), bukan implementasi konkret, sehingga memudahkan penggantian/simulasi saat testing.

7. TRACEABILITY MATRIX

FR	Use Case	Activity/Sequence	Class/Interface
Manajemen Produk	UC-Kelola Produk		Product, ProductService, ProductRepository
Transaksi & Keranjang	UC-Buat Transaksi	Activity Checkout	CartService, CartItem, CheckoutService
Pembayaran Tunai	UC-Checkout (Tunai)	Seq Tunai	CashPayment, PaymentService
Pembayaran E-Wallet	UC-Checkout (E-Wallet)	Seq E-Wallet	EWalletPayment, PaymentService, PaymentGateway
Cetak Struk	UC-Cetak Struk	Activity (akhir)	Transaction, ReceiptService
Laporan	UC-Lihat Laporan		ReportService

8. Checklist Pemeriksaan Mandiri

☑Semua FR tercover di Use Case Diagram

☑Activity & Sequence memuat alur sukses dan gagal (alt/opt)

☑Class Diagram memiliki visibility, tipe, multiplicity

☑Mapping SOLID (min 3) terlihat di desain dan dijelaskan

☑Konsistensi penamaan antar diagram terjaga

☑File sumber diagram (src/uml/*.drawio) + gambar docs/ disertakan
Commit mengikuti format week6-uml-solid: iterasi-N <deskripsi>

9. KESIMPULAN

Berdasarkan perancangan yang telah dilakukan, seluruh diagram UML—use case, activity, sequence, dan class—berhasil menggambarkan alur kerja dan struktur sistem Agri-POS secara jelas dan konsisten. Setiap diagram saling mendukung dalam menjelaskan interaksi aktor, proses bisnis, urutan pesan, serta hubungan antar kelas di dalam sistem. Selain itu, penerapan prinsip SOLID, terutama SRP, OCP, dan DIP, membuat desain lebih fleksibel, mudah dikembangkan, dan lebih terstruktur. Dengan demikian, desain sistem Agri-POS yang dihasilkan sudah sesuai kebutuhan dan layak dijadikan acuan implementasi**.**

10. QUIZ

1. **Jelaskan perbedaan aggregation dan composition serta berikan contoh penerapannya pada desain Anda.**
   Jawaban:
   Aggregation adalah hubungan "bagian-dari" yang longgar, di mana objek bagian *masih bisa berdiri sendiri* meskipun objek induknya hilang. Composition adalah hubungan "bagian-dari" yang kuat, di mana objek bagian *tidak bisa hidup sendiri* tanpa induknya.
   Contoh pada desain:
   a. **Aggregation: Transaction memiliki banyak CartItem. Jika transaksi dibuat, item pada keranjang sebenarnya sudah ada sebelumnya, jadi hubungan ini bersifat aggregation.**
   b. **Composition: Product memiliki PriceHistory (misalnya). Riwayat harga hanya ada jika produk tersebut ada, sehingga hubungan ini merupakan composition.**

2. **Bagaimana Prinsip Open/Closed Memastikan Sistem Mudah Dikembangkan?**
   Jawaban:
   Prinsip Open/Closed menyatakan bahwa kode harus terbuka untuk dikembangkan tetapi tertutup untuk diubah.
   Artinya, ketika ingin menambah fitur baru, developer cukup menambah class atau method baru tanpa harus mengedit kode inti yang sudah berjalan. Hal ini membuat sistem lebih stabil, mengurangi risiko bug, dan memudahkan pengembangan jangka panjang.
   Contohnya dalam desain: untuk menambah metode pembayaran baru, cukup membuat class baru yang mengimplementasi IPaymentMethod tanpa mengubah CheckoutService.

3. **Mengapa DIP Meningkatkan Testability? + Contoh**
   Jawaban:
   Dependency Inversion Principle (DIP) membuat sebuah class bergantung pada interface, bukan pada class konkret.
   Dengan begitu, saat pengujian, kita bisa mengganti implementasi asli dengan

mock atau dummy object sehingga fitur dapat diuji tanpa harus terhubung ke database, payment gateway, atau layanan eksternal lainnya.

Contoh dalam desain: CheckoutService menggunakan interface IPaymentMethod. Pada saat testing, kita bisa mengganti implementasi pembayaran asli dengan MockPaymentMethod yang mensimulasikan pembayaran tanpa benar-benar memanggil gateway. Ini membuat pengujian lebih mudah, cepat, dan aman.