Desain Perangkat Lunak

1. Pendahuluan
   Desain Perangkat Lunak adalah fase transformasi kebutuhan (SRS Document) menjadi arsitektur sistem yang mencakup:
   - Struktur Data (Bagaimana data disimpan dan diakses).
   - Arsitektur Komponen (Hubungan antar modul).
   - Algoritma (Logika bisnis inti).
   - Antarmuka Pengguna (UI/UX) dan API (Application Programming Interface).

   Mengapa Desain Penting?
   - Biaya: Kesalahan desain yang tidak terdeteksi bisa meningkatkan biaya perbaikan hingga 100x di fase implementasi (Sumber: IBM Systems Sciences Institute).
   - Contoh Nyata:
     - Kasus Buruk: Aplikasi healthcare.gov gagal di awal peluncuran karena desain database tidak ternormalisasi.
     - Kasus Baik: Arsitektur microservices Netflix memungkinkan skalabilitas hingga 1 juta request/detik.

Gambar 1. SDLC

2. Prinsip Desain Perangkat Lunak (Detail Teknis)
   Modularitas
   - Cara Implementasi:
     - Bahasa Pemrograman: Gunakan package (Java/Python) atau namespace (C#).
     - Contoh Kode:

```
# Modul Pembayaran (payment.py)
class PaymentProcessor:
    def process_credit_card(self, card_details):
        # Logika validasi dan charge
        pass

# Modul Utama (app.py)
from payment import PaymentProcessor
processor = PaymentProcessor()
```

   Enkapsulasi
   - Level Akses:
     - private (__variable di Python, private di Java).
     - protected (_variable di Python).

- Contoh :

```
public class BankAccount {
    private double balance;  // Enkapsulasi: tidak bisa
diakses langsung dari luar class

    public void deposit(double amount) {
        if (amount > 0) this.balance += amount;
    }
}
```

SOLID Principles (Lengkap dengan Contoh)

- Single Responsibility Principle (SRP):
  - Contoh:

```
class UserManager:
    def add_user(self, user): ...  # Tanggung jawab:
manajemen user
    def send_email(self, user): ...  # Pelanggaran:
tidak terkait manajemen user
```

- Open/Closed Principle (OCP):
  - Contoh Pelanggaran:

```
class Shape(ABC):
    @abstractmethod
    def area(self): pass

class Circle(Shape):  # Ekstensi tanpa modifikasi
class Shape
    def __init__(self, radius): ...
```

- Liskov Substitution Principle (LSP):
  - Contoh Pelanggaran:

```
class Rectangle:
    def set_width(self, w): ...
    def set_height(self, h): ...

class Square(Rectangle):  # Square tidak bisa ganti
width/height independen
    def set_width(self, w):
        super().set_width(w)
        super().set_height(w)  # Memaksa height =
width
```

- Interface Segregation Principle (ISP):
  - Contoh :

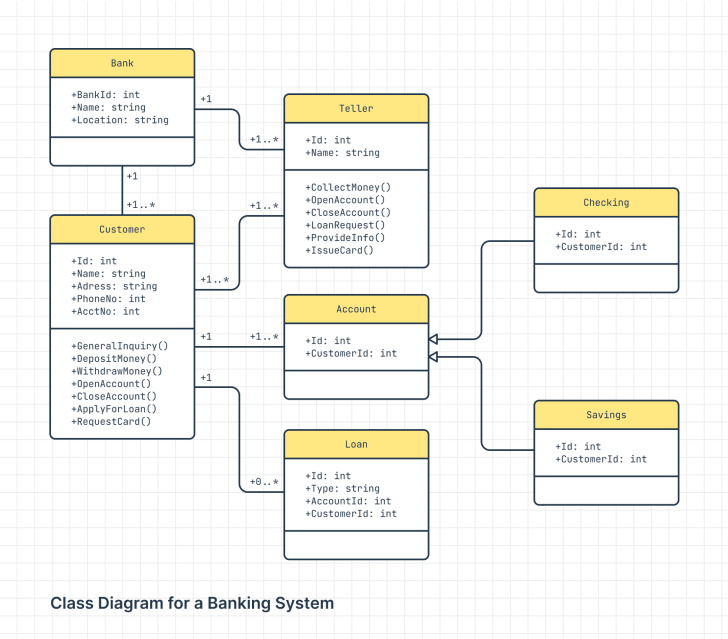
```
interface Printer {
    void print(Document d);
}
interface Scanner {
    void scan(Document d);
}
// Printer sederhana tidak perlu implementasi scan()
```

- Dependency Inversion Principle (DIP):
    - Contoh :

```
class Database(ABC):  # Abstraksi
    @abstractmethod
    def save(self, data): pass

class MySQLDatabase(Database):  # Low-level module
    def save(self, data): ...

class App:  # High-level module
    def __init__(self, db: Database):  # Bergantung
pada abstraksi
        self.db = db
```



Gambar 2. Diagram UML

3. Metode Desain (Deep Dive)
    a. Desain Berorientasi Objek (OOD)
        i. Design Patterns:
            1. Factory Pattern:

```
class RoomFactory:
    @staticmethod
    def create_room(type):
        if type == "standard": return
StandardRoom()
        elif type == "deluxe": return DeluxeRoom()
```

2. Observer Pattern:

```
public interface Observer {
    void update(String message);
}
public class Customer implements Observer {
    public void update(String message) {
        System.out.println("Notifikasi: " +
message);
    }
}
```

b. Arsitektur Sistem

- Layered vs Microservices:

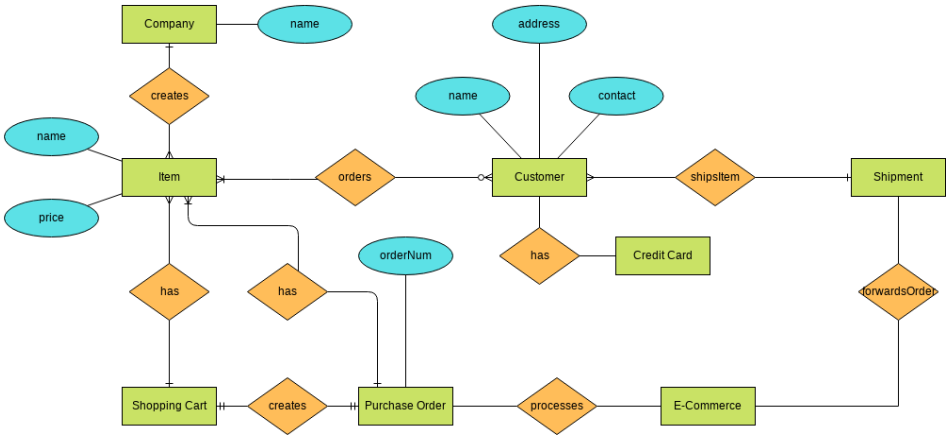
Aspek	Layered	Microservices
Skalabilitas	Vertikal (scale-up server)	Horisontal (scale-out service)
Kompleksitas	Rendah	Tinggi (butuh orchestration)
Contoh	Aplikasi Bank Tradisional	Netflix, Uber

- Diagram: Arsitektur 3-tier (Presentation → Business Logic → Data Access).

c. Desain Database

Normalisasi :

- 1NF: Tidak ada repeating groups (contoh: kolom hobbies berisi "reading,swimming" → pecah ke tabel terpisah).
- 2NF: Memenuhi 1NF + tidak ada partial dependency (contoh: OrderID + ProductID sebagai composite key).
- 3NF: Memenuhi 2NF + tidak ada transitive dependency (contoh: CustomerID → CustomerName → CustomerAddress).



Gambar 3. ERD Sistem Penjualan Online

4. Implementasi (Step-by-Step dengan Tools)
   Langkah 1: Analisis Kebutuhan
   - Tools :
     - Balsamiq untuk wireframe UI.
     - Swagger untuk dokumentasi API.
   - Contoh Dokumen:

```
## Sistem Reservasi Hotel
### Functional Requirements:
1. Pengguna dapat melihat daftar kamar yang tersedia.
2. Admin dapat menambah/mengupdate kamar.
### Non-Functional Requirements:
- Response time < 2 detik.
```

   Langkah 2: Pemodelan dengan UML

   - Class Diagram:

```
classDiagram
  class Customer {
    -id: int
    -name: String
    +makeReservation()
  }
  class Reservation {
    -id: int
    -checkInDate: Date
    +confirm()
  }
  Customer "1" -- "*" Reservation
```

   - Tools:
     - Visual Paradigm (Desktop).
     - PlantUML (Text-based).

Langkah 3: Implementasi Kode

Contoh Project Structure:

```
/hotel-reservation
  ├── src/
  │    ├── models/       # Entity classes
  │    ├── services/     # Business logic
  │    ├── repositories/ # Database access
  │    └── main.py
  ├── tests/             # Unit tests
  └── requirements.txt
```

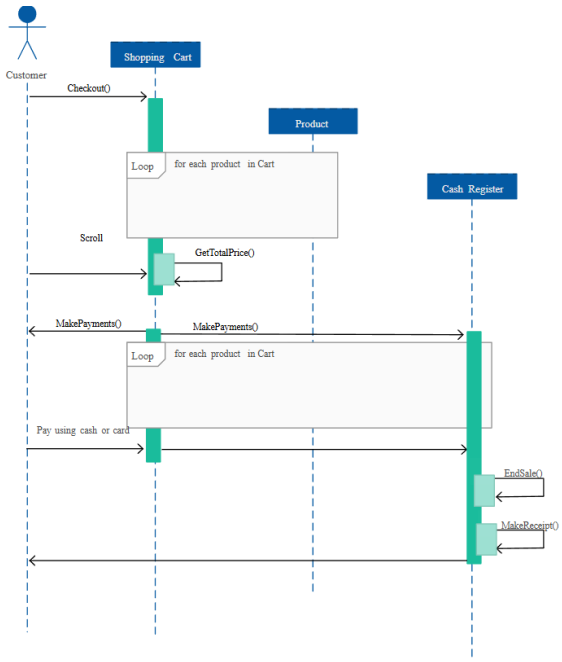
Contoh Unit Test (Python pytest):

```
def test_reservation_confirmation():
    room = Room(type="Deluxe", price=200)
    customer = Customer(name="Alice")
    reservation = Reservation(customer, room)
    assert reservation.confirm() == True
```

5. Studi Kasus Komprehensif: Aplikasi E-Commerce
   Desain Arsitektur
   - Pendekatan: Microservices + CQRS (Command Query Responsibility Segregation).
   - Komponen:
     - Product Service: Menangani katalog produk.
     - Order Service: Proses checkout.
     - Payment Service: Integrasi dengan gateway pembayaran.

   Alur Kerja:

   - User request produk → API Gateway → Product Service (GET /products).
   - User checkout → Order Service → Payment Service (POST /pay).



Gambar 4. Sequence diagram

6. Evaluasi (Latihan & Solusi)
   Latihan 1: Refactoring Kode
   Soal :

```
class ReportGenerator {
    public void generatePDF() { ... }
    public void generateExcel() { ... }
    public void calculateStatistics() { ... }  // Pelanggaran SRP!
}
```

   Solusi :
   Pisahkan ke class terpisah: PDFGenerator, ExcelGenerator, StatisticsCalculator.

   Latihan 2: Desain Database
   - Soal:
     - Tabel Orders dengan kolom: order_id, customer_name, customer_address, product_name, product_price.
     - Masalah: Transitive dependency (order_id → customer_name → customer_address).
   - Solusi:
     - Normalisasi ke 3NF:
       - Tabel Orders: order_id, customer_id.
       - Tabel Customers: customer_id, name, address
7. Referensi & Tools
   a. Referensi
      i. "Domain-Driven Design" oleh Eric Evans (untuk desain kompleks).
      ii. "Refactoring: Improving the Design of Existing Code" oleh Martin Fowler.

b. Alat
    i. Postman: Testing API.
    ii. DBeaver: Desain database.
    iii. SonarQube: Analisis kualitas kode.