Nama : Riezky Arif Fadhilah

NPM : 51421307

Kelas : 3IA227

LAPORAN AKHIR 1

1. Jelaskan apa yang dimaksud dengan desain perangkat lunak berbasis komponen dan bagaimana pendekatan ini membantu dalam mengurangi kompleksitas pengembangan perangkat lunak? Sertakan contoh bagaimana komponen dapat digunakan kembali dalam berbagai aplikasi.

   Jawab :
   Desain perangkat lunak berbasis komponen (Component-Based Software Engineering - CBSE) adalah pendekatan pengembangan perangkat lunak yang memecah sistem menjadi komponen-komponen yang dapat digunakan kembali dan dikombinasikan untuk membentuk aplikasi yang lebih besar.

   Pendekatan ini membantu mengurangi kompleksitas pengembangan perangkat lunak dengan beberapa cara:

- **Pemisahan Tanggung Jawab**: Komponen memiliki fungsi spesifik yang terpisah, sehingga pengembang dapat fokus pada bagian tertentu tanpa harus memahami keseluruhan sistem.
- **Penggunaan Kembali**: Komponen dapat digunakan kembali dalam berbagai proyek atau aplikasi. Misalnya, komponen autentikasi pengguna dapat digunakan dalam berbagai aplikasi yang memerlukan fitur login.
- **Pengujian dan Pemeliharaan yang Lebih Mudah**: Karena komponen independen, pengujian bisa dilakukan secara modular, dan pembaruan komponen tertentu tidak memengaruhi seluruh aplikasi.

   Contoh Penggunaan Kembali Komponen: Komponen seperti "modul pengolahan pembayaran" bisa digunakan di berbagai aplikasi, seperti e-commerce, aplikasi layanan streaming, atau sistem reservasi hotel. Dalam setiap aplikasi, fungsinya tetap sama, yakni memproses pembayaran, tetapi konteksnya berbeda.

2. Jelaskan konsep aplikasi terdistribusi dan bandingkan arsitektur 2-tier dan 3-tier. Berikan contoh aplikasi yang dapat diimplementasikan dengan arsitektur 3-tier dan jelaskan keuntungannya.

   Jawab :
   Aplikasi terdistribusi adalah aplikasi yang komponen-komponennya tersebar di berbagai lokasi dalam suatu jaringan, yang memungkinkan mereka berjalan di beberapa

komputer secara paralel. Dalam arsitektur ini, aplikasi berinteraksi dengan server melalui jaringan, dengan data dan pemrosesan yang dibagi di berbagai lokasi.

Arsitektur 2-tier: Dalam arsitektur ini, aplikasi terbagi menjadi dua lapisan:

- **Lapisan Presentasi (Client)**: Ini adalah lapisan front-end yang menangani interaksi pengguna.
- **Lapisan Data (Server/Database)**: Menyimpan dan mengelola data. Pemrosesan logika aplikasi sering dilakukan di sisi client atau server secara terpusat.

Arsitektur 3-tier: Memperkenalkan lapisan tambahan yang membagi tanggung jawab:

- **Lapisan Presentasi**: Client-side untuk interaksi pengguna.
- **Lapisan Logika Aplikasi**: Business logic diletakkan di server terpisah yang menangani pemrosesan.
- **Lapisan Data**: Berisi database yang menyimpan data secara terpusat.

Perbandingan arsitektur **2-tier** dan **3-tier** terletak pada cara kedua arsitektur ini memisahkan fungsi dalam aplikasi, yang mempengaruhi skalabilitas, fleksibilitas, dan kinerja aplikasi terdistribusi.

Keuntungan Arsitektur 3-tier:

- **Skalabilitas Lebih Baik**: Lapisan business logic dapat diskalakan secara independen dari client atau database.
- **Keamanan yang Lebih Baik**: Lapisan logika aplikasi menyediakan lapisan pengamanan tambahan antara pengguna dan database.
- **Pemeliharaan Mudah**: Pengubahan di satu lapisan tidak memengaruhi lapisan lain.

3. Jelaskan apa itu ERP (Enterprise Resource Planning) dan bagaimana sistem ini dapat membantu perusahaan mengelola operasional mereka. Berikan penjelasan tentang tantangan yang mungkin dihadapi perusahaan saat mengimplementasikan ERP dan bagaimana tantangan tersebut dapat diatasi.

   Jawab :
   ERP (Enterprise Resource Planning) adalah sistem terintegrasi yang digunakan oleh perusahaan untuk mengelola berbagai fungsi bisnis dalam satu platform yang terpadu, termasuk akuntansi, manajemen sumber daya manusia, produksi, dan pengelolaan inventaris. ERP memungkinkan perusahaan untuk memiliki pandangan holistik terhadap operasionalnya dengan menyatukan data dari berbagai departemen.

   Tantangan dalam Implementasi ERP:

- **Biaya**: Implementasi ERP membutuhkan investasi besar dalam perangkat lunak, hardware, dan pelatihan.
- **Penyesuaian dengan Proses Bisnis**: ERP sering memerlukan perubahan dalam proses bisnis yang ada agar sesuai dengan sistem, yang bisa memicu resistensi dari karyawan.
- **Pelatihan dan Perubahan Budaya**: Transisi ke sistem ERP memerlukan pelatihan intensif dan perubahan budaya di seluruh organisasi.

Cara Mengatasi Tantangan:

- **Perencanaan yang Matang**: Melakukan analisis kebutuhan yang mendalam sebelum implementasi untuk memastikan kesesuaian antara ERP dan proses bisnis perusahaan.
- **Pelatihan dan Manajemen Perubahan**: Memberikan pelatihan yang memadai kepada karyawan dan memperkenalkan sistem secara bertahap untuk mengurangi resistensi.
- **Konsultasi Ahli**: Menggunakan konsultan ERP yang berpengalaman untuk memastikan implementasi berjalan lancar.

4. Apa peran middleware dalam aplikasi terdistribusi berbasis komponen? Jelaskan bagaimana middleware berfungsi sebagai penghubung antara front-end dan back-end serta memberikan contoh nyata dari implementasi middleware dalam suatu sistem.

   Jawab :
   Middleware adalah perangkat lunak yang berfungsi sebagai lapisan penghubung antara front-end (lapisan presentasi) dan back-end (server atau database) dalam sistem terdistribusi. Middleware menyediakan layanan komunikasi, keamanan, transaksi, dan interoperabilitas antar aplikasi yang berjalan di lingkungan terdistribusi.

   Fungsi Middleware:

- **Penghubung antara Client dan Server**: Middleware memediasi permintaan dari front-end ke back-end dan memastikan komunikasi yang efisien.
- **Abstraksi Jaringan**: Middleware menyembunyikan kompleksitas komunikasi jaringan dari pengembang aplikasi, sehingga mereka tidak perlu khawatir tentang protokol komunikasi yang digunakan.
- **Pengelolaan Transaksi**: Menyediakan manajemen transaksi yang konsisten, terutama dalam sistem yang membutuhkan integritas data, seperti sistem perbankan.

   Contoh Middleware:

- **Apache Kafka**: Middleware yang berfungsi sebagai platform streaming untuk menghubungkan berbagai layanan dalam aplikasi besar yang terdistribusi. Kafka memungkinkan pertukaran data antar sistem dengan kecepatan tinggi, dan digunakan dalam berbagai aplikasi seperti sistem e-commerce, manajemen logistik, dan analitik real-time.

LAPORAN AKHIR 2

1. Jelaskan konsep-konsep utama dalam Object-Oriented Programming (OOP) seperti Encapsulation, Inheritance, dan Polymorphism. Berikan contoh penerapan setiap konsep ini dalam program Java.

   Jawab:
   Konsep-konsep Utama dalam Object-Oriented Programming (OOP)
   OOP adalah paradigma pemrograman yang berfokus pada penggunaan objek dan kelas untuk mengorganisasi kode. Berikut adalah beberapa konsep utama dalam OOP:

- **Encapsulation (Enkapsulasi):**
  Encapsulation adalah konsep di mana data (variabel) dan metode (fungsi) dari suatu objek disembunyikan atau dilindungi dari akses luar, dan hanya dapat diakses atau dimodifikasi melalui metode yang tersedia di dalam kelas tersebut. Tujuannya adalah menjaga keamanan data dan mengurangi kompleksitas dengan menyembunyikan detail implementasi.

  Contoh Encapsulation dalam Java:

```
class BankAccount {
    private double balance;  // Variabel ini dienkapsulasi

    // Constructor
    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    // Getter (akses balance)
    public double getBalance() {
        return balance;
    }

    // Setter (modifikasi balance)
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
}
```

- Inheritance (Pewarisan):
  Inheritance adalah mekanisme di mana sebuah kelas dapat mewarisi sifat dan perilaku (variabel dan metode) dari kelas lain. Kelas yang diwarisi disebut superclass (induk),

dan kelas yang mewarisi disebut subclass (anak). Ini membantu dalam pemanfaatan kembali kode dan meningkatkan keterbacaan.

Contoh Inheritance dalam Java:

```
// Superclass
class Animal {
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

// Subclass
class Dog extends Animal {
    public void makeSound() {
        System.out.println("Dog barks");
    }
}
```

- Polymorphism (Polimorfisme):
  Polymorphism memungkinkan satu metode dapat memiliki banyak bentuk. Dalam OOP, ini biasanya terjadi dalam bentuk method overriding (satu metode dengan perilaku berbeda di kelas turunan) dan method overloading (metode dengan nama sama tetapi parameter berbeda dalam satu kelas).\

  Contoh Polymorphism dalam Java:

```
// Method Overriding
class Animal {
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat meows");
    }
}

class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myCat = new Cat();  // Polimorfisme
        myAnimal.makeSound();  // Output: Animal makes a sound
        myCat.makeSound();     // Output: Cat meows
```

```
    }
}
```

2. Buatlah sebuah program yang memanfaatkan konsep-konsep OOP dan berikan penjelasannya.

   Jawab :
   Berikut adalah contoh program sederhana yang memanfaatkan **encapsulation**, **inheritance**, dan **polymorphism** dalam Java:

```
// Encapsulation Example
class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // Setter methods
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }

    // Method to display information
    public void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

```
// Inheritance Example
class Employee extends Person {
    private double salary;

    public Employee(String name, int age, double salary) {
        super(name, age);  // Call constructor of superclass (Person)
        this.salary = salary;
    }

    // Getter method for salary
    public double getSalary() {
        return salary;
    }

    // Method Overriding (Polymorphism)
    @Override
    public void displayInfo() {
        super.displayInfo();  // Call the superclass method
        System.out.println("Salary: $" + salary);
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an instance of Employee
        Employee emp = new Employee("John Doe", 30, 50000.0);

        // Accessing encapsulated data using getter and setter
        emp.setAge(35);  // Modifying the age
        System.out.println(emp.getName() + " is now " + emp.getAge() + " years old.");

        // Display employee information (Polymorphism in action)
        emp.displayInfo();
    }
}
```

Penjelasan Program:

- **Encapsulation**: Kelas Person memiliki atribut name dan age yang dienkapsulasi. Atribut ini hanya dapat diakses dan dimodifikasi melalui metode getter dan setter.
- **Inheritance**: Kelas Employee adalah subclass dari Person, yang berarti Employee mewarisi atribut dan metode dari Person.
- **Polymorphism**: Metode displayInfo di Employee menimpa (override) metode yang sama dari kelas Person, memberikan keluaran yang berbeda untuk objek Employee.