LAPORAN

RESPONSI PBO

ACHMAD YOGA ALFANDI

5230411291

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS TEKNOLOGI YOGYAKARTA

2024

Soal Teori

1. **Jelaskan perbedaan use case diagram dengan class diagram?**

   Use Case Diagram adalah diagram yang menggambarkan interaksi antara sistem dan aktor (pengguna/sistem eksternal), yang menunjukkan fungsionalitas yang diharapkan dari sebuah sistem. Diagram ini mendeskripsikan sebuah sistem dari sudut pandang pengguna sistem tersebut (user's view). Use case diagram merupakan salah satu diagram behavioral UML yang menunjukkan "APA" yang sistem dapat lakukan.

   Class Diagram adalah diagram struktur yang menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki atribut (variabel-variabel) dan metode (fungsi-fungsi). Class diagram menunjukkan hubungan antar kelas dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.

   Perbedaan utama:

UML (Unified Modeling Language)	Use Case Diagram	Class Diagram
Level abstraksi	High-level, fokus pada fungsionalitas	Low-level, fokus pada implementasi
Audience	Stakeholder non-teknis bisa memahami	Lebih ditujukan untuk developer/programmer
Timing	Dibuat di awal fase analisis	Dibuat saat fase desain
Detail	Sederhana, menunjukkan fungsi utama	Detail, menunjukkan struktur internal lengkap

2. **Jelaskan jenis-jenis dependensi?**

   Dependensi adalah hubungan antara tugas-tugas atau komponen yang saling bergantung satu sama lain untuk menyelesaikan suatu pekerjaan.

   Berikut ini adalah beberapa jenis dependensi yang umum:

   1. Dependensi Logis
      - Finish to Start (FS): Tugas A harus selesai sebelum Tugas B dapat dimulai.
        Contoh: Pengecatan dinding harus selesai sebelum pemasangan dekorasi.
      - Start to Start (SS): Tugas A harus dimulai sebelum Tugas B dapat dimulai.
        Contoh: Setelah menggali pondasi, Anda bisa mulai membangun fondasi.
      - Finish to Finish (FF): Tugas A harus selesai sebelum Tugas B bisa selesai.
        Contoh: Menulis dan mengedit buku harus selesai bersama-sama.
      - Start to Finish (SF): Tugas A harus dimulai sebelum Tugas B dapat selesai. (Ini jarang terjadi dalam praktik.)
   2. Dependensi Wajib (Mandatory Dependencies)
      Hubungan ini tidak bisa dihindari dan biasanya terkait dengan faktor fisik atau hukum. Contoh: Pemasangan atap tidak bisa dilakukan sebelum struktur dinding utama dibangun.
   3. Dependensi Pilihan (Discretionary Dependencies)
      Hubungan ini diputuskan berdasarkan praktik terbaik atau kebijakan tertentu, dan bisa diubah jika perlu. Contoh: Pengujian perangkat lunak mungkin dipilih untuk dilakukan setelah semua modul selesai, tetapi dalam beberapa kasus bisa dilakukan paralel.
   4. Dependensi Eksternal

Hubungan antara tugas-tugas proyek dan elemen eksternal di luar kendali tim proyek. Contoh: Pengiriman bahan baku tergantung pada pemasok luar.

5. Dependensi Internal
   Hubungan antar tugas-tugas yang dikelola secara internal dalam proyek. Contoh: Pengembangan modul perangkat lunak A mungkin harus diselesaikan sebelum modul B dikembangkan.

3. Apa perbedaan pemrograman terstruktur dengan berorientasi objek, jelaskan?

Pemrograman terstruktur adalah metode pemrograman yang menekankan pada proses atau fungsi. Program dibagi menjadi bagian-bagian kecil (fungsi/prosedur) yang lebih sederhana untuk menyelesaikan masalah. Data dan fungsi dipisahkan, dimana fungsi berperan untuk memproses data. Pendekatan ini menggunakan alur top-down, yaitu memecah masalah besar menjadi sub-masalah yang lebih kecil.

Pemrograman berorientasi objek (OOP) adalah metode pemrograman yang berfokus pada objek, dimana data dan fungsi (method) dikemas menjadi satu kesatuan dalam class. Program terdiri dari kumpulan objek yang saling berinteraksi. OOP menggunakan konsep encapsulation (pembungkusan), inheritance (pewarisan), polymorphism (banyak bentuk), dan abstraction (penyederhanaan).

Perbedaan utamanya :

1. Pendekatan
- Terstruktur: Fokus pada fungsi dan alur program
- OOP: Fokus pada objek dan interaksinya

2. Organisasi Data
- Terstruktur: Data dan fungsi terpisah
- OOP: Data dan method tergabung dalam class

3. Keamanan Data
- Terstruktur: Keamanan data kurang terjamin karena data bisa diakses global
- OOP: Keamanan data lebih terjamin karena adanya enkapsulasi

4. Pengembangan:
- Terstruktur: Cocok untuk program kecil dan sederhana
- OOP: Lebih cocok untuk program besar dan kompleks

5. Reusability:
- Terstruktur: Reusability terbatas pada fungsi
- OOP: Reusability tinggi melalui inheritance dan polymorphism

6. Maintenance:
- Terstruktur: Lebih sulit untuk maintenance program besar
- OOP: Lebih mudah untuk maintenance karena modular

7. Kompleksitas:
- Terstruktur: Lebih sederhana dan mudah dipahami
- OOP: Lebih kompleks tetapi lebih fleksibel

8. Kinerja:
- Terstruktur: Eksekusi program lebih cepat, memori lebih kecil
- OOP: Eksekusi bisa lebih lambat, memori lebih besar

Masing-masing paradigma memiliki keunggulan dalam konteks tertentu:

Pemrograman Terstruktur ideal untuk:
- Program sederhana
- Sistem dengan sumber daya terbatas
- Aplikasi yang membutuhkan kinerja tinggi
- Alur program yang linear

Pemrograman Berorientasi Objek ideal untuk:
- Aplikasi besar dan kompleks
- Pengembangan tim
- Sistem yang membutuhkan maintenance jangka panjang
- Program yang membutuhkan reusability tinggi.

4. Jelaskan konsep objek dan beri contohnya?

Dalam pemrograman berorientasi objek (OOP), objek adalah entitas yang merepresentasikan sesuatu di dunia nyata atau abstraksi yang didefinisikan dalam kode.

Objek memiliki dua elemen utama:

Atribut: Merupakan data atau properti yang dimiliki oleh objek. Atribut ini menggambarkan karakteristik objek.
Metode: Adalah fungsi atau tindakan yang dapat dilakukan oleh objek. Metode menentukan perilaku atau aksi yang bisa dilakukan oleh objek.

Konsep Objek
Objek adalah instansiasi dari kelas. Kelas adalah blueprint atau cetak biru yang mendefinisikan atribut dan metode. Ketika sebuah objek dibuat, ia memiliki data spesifik berdasarkan cetak biru tersebut.
Dalam OOP, segala sesuatu dapat diperlakukan sebagai objek, seperti orang, kendaraan, hewan, atau bahkan struktur data.

Contoh Objek
Misalkan kita memiliki kelas Mobil yang merepresentasikan karakteristik dan perilaku dari sebuah mobil.
class Mobil:

```
        def __init__(self, merk, warna):
            self.merk = merk  # Atribut
            self.warna = warna  # Atribut

        def nyalakan_mesin(self):  # Metode
            print(f"Mesin {self.merk} dinyalakan!")

        def matikan_mesin(self):  # Metode
            print(f"Mesin {self.merk} dimatikan.")
```

Membuat Objek dari Kelas Mobil

```
mobil1 = Mobil("Toyota", "Merah")
mobil2 = Mobil("Honda", "Hitam")

# Mengakses atribut
print(mobil1.merk)  # Output: Toyota
print(mobil2.warna)  # Output: Hitam

# Memanggil metode
mobil1.nyalakan_mesin()  # Output: Mesin Toyota dinyalakan!
mobil2.matikan_mesin()  # Output: Mesin Honda dimatikan.
```

Penjelasan
mobil1 dan mobil2 adalah contoh objek yang dibuat dari kelas Mobil. Mereka memiliki atribut merk dan warna yang spesifik, serta dapat melakukan aksi (memanggil metode) seperti nyalakan_mesin() dan matikan_mesin().
Objek mobil1 memiliki data "Toyota" dan "Merah", sedangkan mobil2 memiliki data "Honda" dan "Hitam".

5. Jelaskan jenis-jenis access modifier beri contohnya dalam bari pemrograman?

Access modifier adalah keyword dalam pemrograman berorientasi objek yang digunakan untuk mengatur tingkat aksesibilitas anggota (seperti variabel, method, atau constructor) dalam sebuah kelas.
Berikut adalah jenis-jenis access modifier :

1. Public
Anggota yang dideklarasikan sebagai public dapat diakses dari mana saja, baik dari dalam kelas yang sama, kelas lain, maupun package lain.

Contoh java:

```
public class MyClass {
    public int myNumber = 10;

    public void showNumber() {
        System.out.println(myNumber);
    }
}

// Kelas lain
public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.myNumber); // Bisa diakses
        obj.showNumber(); // Bisa diakses
    }
}
```

2. Private
Anggota yang dideklarasikan sebagai private hanya dapat diakses dari dalam kelas itu sendiri. Anggota private tidak dapat diakses dari luar kelas.

Contoh java:

```
public class MyClass {
    private int myNumber = 10;

    private void showNumber() {
        System.out.println(myNumber);
    }
}

// Kelas lain
public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        // System.out.println(obj.myNumber); // Tidak bisa diakses (Error)
        // obj.showNumber(); // Tidak bisa diakses (Error)
    }
}
```

3. Protected
Anggota yang dideklarasikan sebagai protected dapat diakses di dalam package yang sama dan juga di subclass (kelas turunan) meskipun berada di package yang berbeda.

Contoh java:

```
package package1;

public class MyClass {
    protected int myNumber = 10;

    protected void showNumber() {
        System.out.println(myNumber);
    }
}

// Kelas di package lain
package package2;

import package1.MyClass;

public class SubClass extends MyClass {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
        System.out.println(obj.myNumber); // Bisa diakses
        obj.showNumber(); // Bisa diakses
    }
}
```

4. Default (Package-Private)
Jika tidak ada access modifier yang dideklarasikan, maka default-nya adalah package-private, yang berarti anggota hanya dapat diakses oleh kelas-kelas dalam package yang sama.

Contoh java:

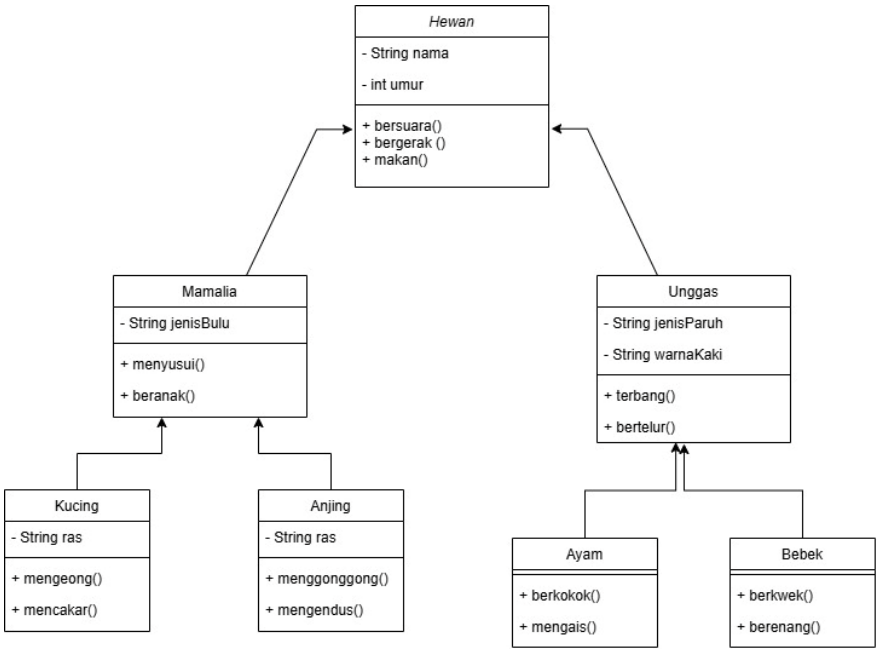
```
class MyClass {
    int myNumber = 10; // Default access modifier

    void showNumber() {
        System.out.println(myNumber);
    }
}

// Kelas lain dalam package yang sama
public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.myNumber); // Bisa diakses
        obj.showNumber(); // Bisa diakses
    }
}
```

Ringkasan
Public: Akses dari mana saja
Private: Akses hanya dalam kelas itu sendiri
Protected: Akses dalam package yang sama dan subclass di luar package
Default (Package-Private): Akses hanya dalam package yang sama.

6. Gambarkan contoh pewarisan dalam diagram class?



Saya telah membuat diagram class yang menunjukkan hierarki pewarisan untuk kelas Hewan. Berikut penjelasannya:

1. Hewan adalah kelas induk (superclass) yang memiliki properti dan method dasar:
    - Properti: nama, umur
    - Method: bersuara(), bergerak(), makan()
2. Mamalia dan Unggas mewarisi dari kelas Hewan:
    - Mamalia menambahkan properti jenisBulu dan method menyusui()
    - Unggas menambahkan properti jenisParuh, warnaKaki dan method terbang(), bertelur()
3. Kelas-kelas spesifik mewarisi dari Mamalia dan Unggas:
    - Kucing dan Anjing mewarisi dari Mamalia
    - Ayam dan Bebek mewarisi dari Unggas

Setiap kelas turunan memiliki method spesifiknya sendiri, seperti:

- Kucing: mengeong(), mencakar()
- Anjing: menggonggong(), mengendus()
- Ayam: berkokok(), mengais()
- Bebek: berkwek(), berenang()

Diagram ini menunjukkan hubungan "is-a" dalam pewarisan, dimana:

- Kucing adalah Mamalia
- Mamalia adalah Hewan
- Bebek adalah Unggas
- Unggas adalah Hewan Dan seterusnya.