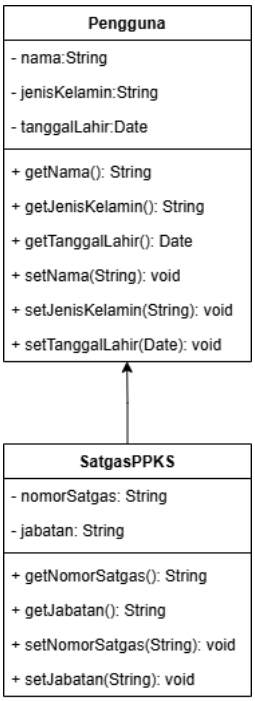
TUGAS 4
INHERITANCE

Diajukan untuk memenuhi salat satu tugas Mata kuliah Pemrograman Berorientasi Objek

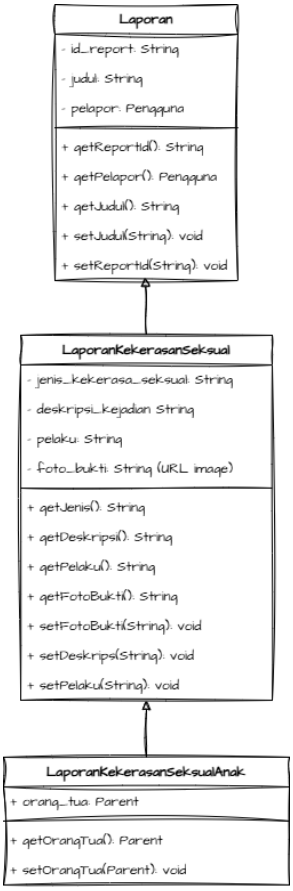
Disusun Oleh:
Daiva Raditya Pradipa (231511039)

Jurusan Teknik Komputer dan Informatika
Program Studi D-3 Teknik Informatika
Politeknik Negeri Bandung
2024

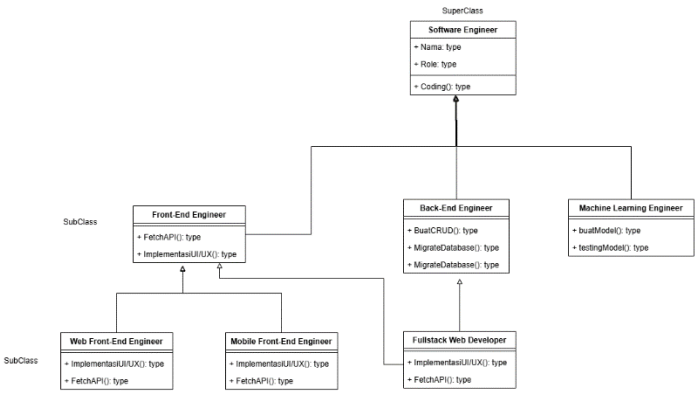
1. Buatlah contoh hirarki yang menerapkan **single inheritance** (sesuaikan dengan kasus anda di matkul PRPL) ?



2. Buatlah contoh hirarki yang menerapkan **multiple level single inheritance** (sesuaikan dengan kasus anda di matkul PRPL)?



3. Buatlah contoh hirarki yang menerapkan **multiple inheritance** (jangan menggunakan contoh yang sudah diberikan) ?



4. Jelaskan mengapa inheritance dapat meningkatkan aspek **"reuse"** ?

   Hal ini dikarenakan saat menggunakan konsep inheritance (pewarisan), subclass bisa mewarisi atribut dan metode dari superclass. Ini artinya, kita tidak perlu menulis ulang atribut atau metode yang sudah ada di superclass karena subclass bisa langsung menggunakan atribut dan metode yang diwariskan dari superclass (super()), tanpa perlu mendefinisikannya lagi pada masing-masing subclass (jika terdapat lebih dari 1 subclass yang mengextends atau menjadi subclass dari satu super class).

5. Bagaimana pendapat anda terhadap banyaknya level hirarki pada inheritance ?

   Menutut saya dengan adanya banyak level hirarki tentunya memiliki kelebihan dan kekurangannya diantaranya:

   Kelebihan:

   a. Memaksimalkan konsep Reuseability
      Tentunya dengan banyaknya level hirarki pada inheritance kita dapat memaksimalkan konsep reuseabaility karena kita dapat mewariskan atribut dan method yang ada pada class di level atas pada masing masing class pada level bawah. Hal ini akan mengurangi duplikasi kode, membuat kode lebih terorganisir dan ringkas pada masing-masing class pada level bawah.
   b. Struktur Sistem yang Terorganisir
      Dengan banyak level hierarki, program dapat diorganisir dengan lebih baik. Kelas-kelas di level yang lebih tinggi dapat mendefinisikan perilaku umum, sementara kelas-kelas di level bawah dapat menambah perilaku spesifik. Hal ini membantu menciptakan struktur yang modular, teratur, dan mengikuti prinsip desain abstraksi dan generalisasi.

   Kekurangan:

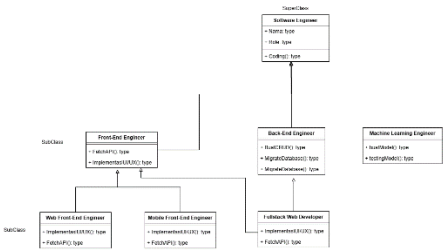
   a. Sturktur program yang menjadi kompleks

Semakin banyak level hierarki, semakin rumit struktur program. Ini dapat menyebabkan kebingungan saat mencoba memahami bagaimana komponen-komponen dalam sistem berinteraksi, terutama untuk pengembang baru yang tidak familiar dengan sistem tersebut.

b. Kesulitan untuk mengubah class secara independen
Class yang berada di hierarki dalam bisa memiliki keterikatan yang erat (tight coupling) dengan kelas-kelas di atasnya. Ini bisa menyulitkan saat kita ingin membuat perubahan class secara independen, karena banyak bagian dari kode saling bergantung satu sama lain.

6. Apakah permasalahan yang muncul pada kasus **multiple inheritance** pada Java ?

Misalkan terdapat class Frontend enginner dan class Backend engineer. Kemudian ada class Fullstack engineer yang menjadi subclass dari Frontend enginner dan class Backend engineer.



Saat kita ingin mencoba mengimplementasikan struktur tersebut dalam java mungkin kita akan menuliskan Class FullStack extends Frontend extends Backend atau Class FullStack extends, Frontend, Backend.

Masalah yang muncul pada syntax ini adalah java tidak memperbolehkan penulisan pewarisan 2 supperclass pada satu subclass dengan cara seperti ini, java akan memberikan pesan error pada saat menuliskan kedua syntax tersebut. Mengapa demikian ? Hal ini dikarenakan bila java memperbolehkan implementasi multiple inheritance dengan syntax tersebut dapat memicu masalah seperti

a. Ambiguity (Ketidakjelasan) atau Diamond Problem

Jika Java mendukung pewarisan dari dua superclass, kasus ambigu bisa terjadi ketika dua superclass memiliki metode yang sama, dan subclass tidak tahu metode mana yang harus dipanggil. Sebagai contoh:

```
class Frontend {
  public void develop() {
      System.out.println("Frontend Development");
   }
}

class Backend {
   public void develop() {
      System.out.println("Backend Development");
   }
```

```
}

class FullStack extends Frontend, Backend {
   // Error: Java tidak mengizinkan pewarisan ganda dari Frontend dan Backend
}
```

Dalam contoh di atas, jika FullStack memanggil metode develop(), menjadi tidak jelas apakah yang dimaksud adalah metode develop() dari kelas Frontend atau dari Backend. Masalah ini dikenal sebagai Diamond Problem.

7. Bagaimana solusi yang mungkin dilakukan untuk menangani **multiple inheritance** pada Java ?

   Untuk mengatasi masalah java yang tidak memperbolehkan adalah dengan menggunakan multiple interfaces. interface hanya mendefinisikan kontrak (deklarasi metode tanpa implementasi). Oleh karena itu, ketika suatu class mengimplementasikan dua atau lebih interface, class tersebut harus membuat implementasi untuk metode yang didefinisikan oleh interface tersebut, sehingga tidak ada ambiguitas.

   Contoh implementasi

```
interface Frontend {
   void developFrontend();
}

interface Backend {
   void developBackend();
}

class FullStack implements Frontend, Backend {
   @Override
   public void developBackend() {
      System.out.println("Development");
   }

   public void developFrontend() {
      System.out.println("Development");
   }
}
```

Namun bagaimana apabila interface Backend dan Frontend memiliki nama method yang sama. Dalam kasus ini, ketika kelas yang mengimplementasikan kedua interface tersebut harus menyediakan implementasi untuk metode tersebut, kita perlu memastikan bahwa implementasi tersebut sesuai dengan konteksnya.

```
interface Frontend {
    void develop();
}

interface Backend {
    void develop();
}
class FullStack implements Frontend, Backend {
    @Override
    public void develop() {
        System.out.println("Developing features...");
    }
}
```

Class Fullstack hanya akan membuat 1 implementasi meskipun terdapat dua method dengan nama yang sama dari interface Backend dan Frontend.