LAPORAN PRAKTIKUM 7

PEMROGRAMAN BERORIENTASI OBJEK (PBO)

Nama	: Alya Gustiani Nur ‘Afifah
NIM	: 231511035
Kelas/Prodi	: 2B/D3 Teknik Informatika

Politeknik Negeri Bandung

2024

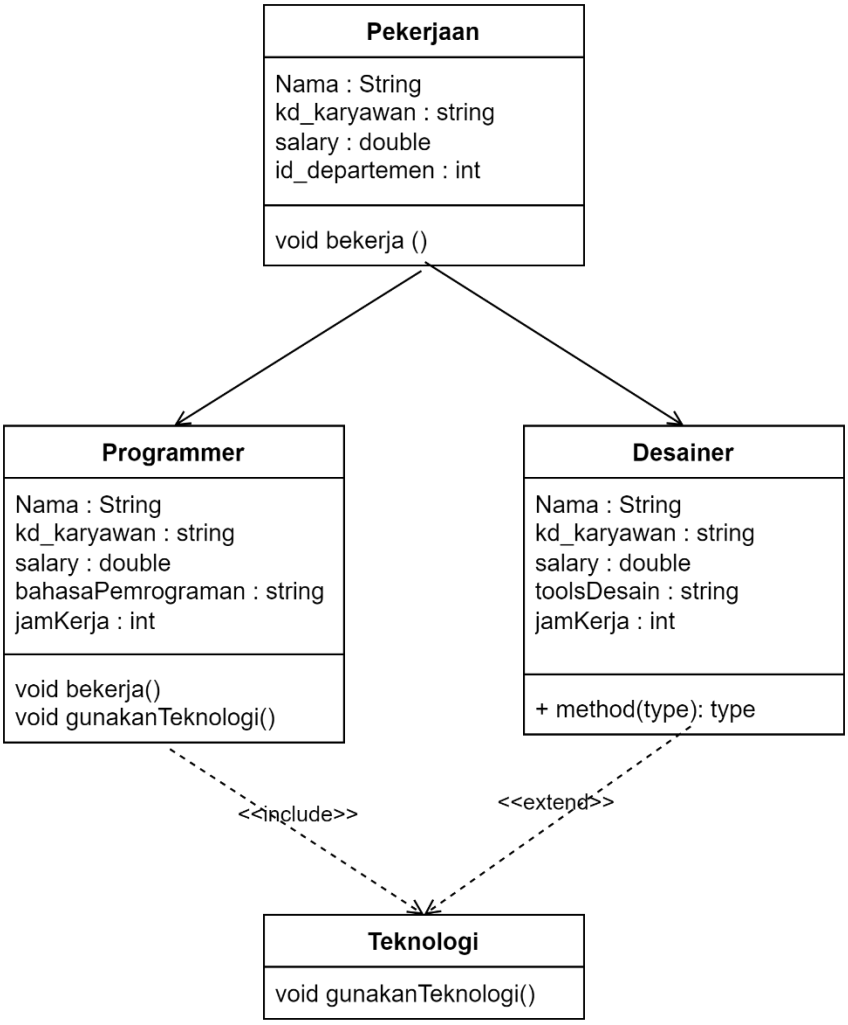
1. Link Repository GitHub : https://github.com/alyagustiani/PBOSem2/tree/master/2B_Pertemuan%207_035_Alya%20Gustiani
2. Soal 1 : mengimplementasikan konsep abstract class, interface dan implementation dalam satu kasus.
    a. Contoh kasus

        Sebuah perusahaan teknologi sedang mengembangkan aplikasi baru untuk pengguna smartphone. Untuk menyelesaikan proyek ini, mereka membentuk tim yang terdiri dari beberapa profesi, termasuk **programmer** dan **desainer**. Masing-masing profesi memiliki cara kerja yang berbeda tetapi juga memiliki beberapa kesamaan, seperti menggunakan teknologi terkini dalam pekerjaan mereka.

    b. Lesson learnt

        Dalam mengimplementasikan konsep abstract class, interface, dan implementasi dalam satu kasus, ada beberapa lesson learnt yang dapat diambil. Pertama, abstract class digunakan untuk generalisasi. Kelas abstrak berguna saat Anda memiliki beberapa objek dengan karakteristik yang sama, tetapi Anda tidak ingin mengizinkan pembuatan objek langsung dari kelas tersebut. Sebagai contoh, dalam kasus ini, Pekerjaan menjadi kerangka umum untuk berbagai profesi seperti programmer dan desainer, memungkinkan generalisasi perilaku seperti metode bekerja(). Kedua, interface digunakan untuk mendefinisikan perilaku yang dapat dimiliki oleh berbagai kelas tanpa mengikatnya pada hierarki tertentu. Interface memungkinkan berbagai kelas yang tidak berhubungan secara langsung untuk berbagi perilaku umum, seperti Teknologi, yang memberikan kemampuan bagi berbagai profesi untuk menggunakan teknologi. Hal ini memungkinkan implementasi yang fleksibel dan dapat diperluas tanpa harus terikat pada hirarki tunggal. Ketiga, kombinasi abstract class dan interface memberikan solusi desain yang lebih modular dan fleksibel, memungkinkan kode untuk lebih mudah dipelihara dan dikembangkan di masa mendatang. Penggunaan konsep ini membantu menghindari pengulangan kode dan mempermudah penambahan fitur baru melalui pola inheritance dan polymorphism, meningkatkan efisiensi pengembangan.

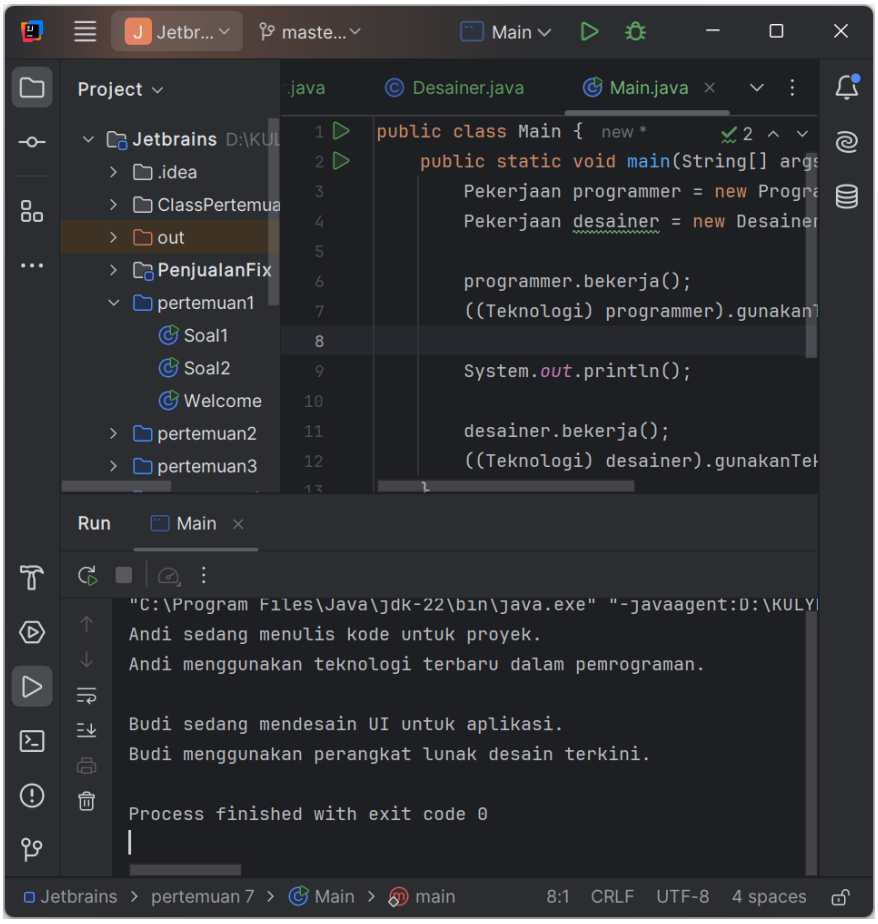
3. Soal 2 : Buat diagram kelasnya
    a. Screenshoot setiap jawaban soal yang dipertanyakan.



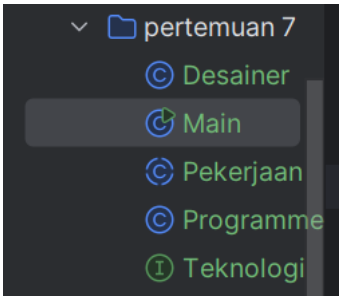
    b. Permasalahan yang dihadapi.
    Tidak semua variabel cocok untuk diturunkan dari *abstract class*. Contohnya, variabel jamKerja mungkin berbeda antara Programmer dan Desainer, sehingga tidak bisa didefinisikan secara umum di *abstract class* Pekerjaan.
    c. Solusi dari permasalahan yang dihadapi.
    Variabel yang tidak relevan untuk semua subclass, seperti jamKerja, didefinisikan secara spesifik di masing-masing subclass (Programmer dan Desainer). Ini menciptakan fleksibilitas dalam mendesain subclass, memungkinkan mereka memiliki atribut yang spesifik sesuai dengan kebutuhan pekerjaan tersebut.
    d. Lesson learnt

Dalam penggambaran diagram kelas, penting untuk menyeimbangkan generalisasi dan spesifikasi dengan menempatkan atribut yang umum di abstract class dan atribut yang lebih spesifik di subclass, sehingga sistem menjadi fleksibel dan mudah dipelihara. Penggunaan interface harus bijak, dengan fokus pada representasi karakteristik umum yang dapat dibagi tanpa mengikat kelas pada hierarki yang kaku. Fleksibilitas dalam menentukan apakah variabel diturunkan atau tidak membantu menjaga desain sistem relevan dan terstruktur, sehingga mempermudah pengembangan dan perluasan sistem di masa depan.

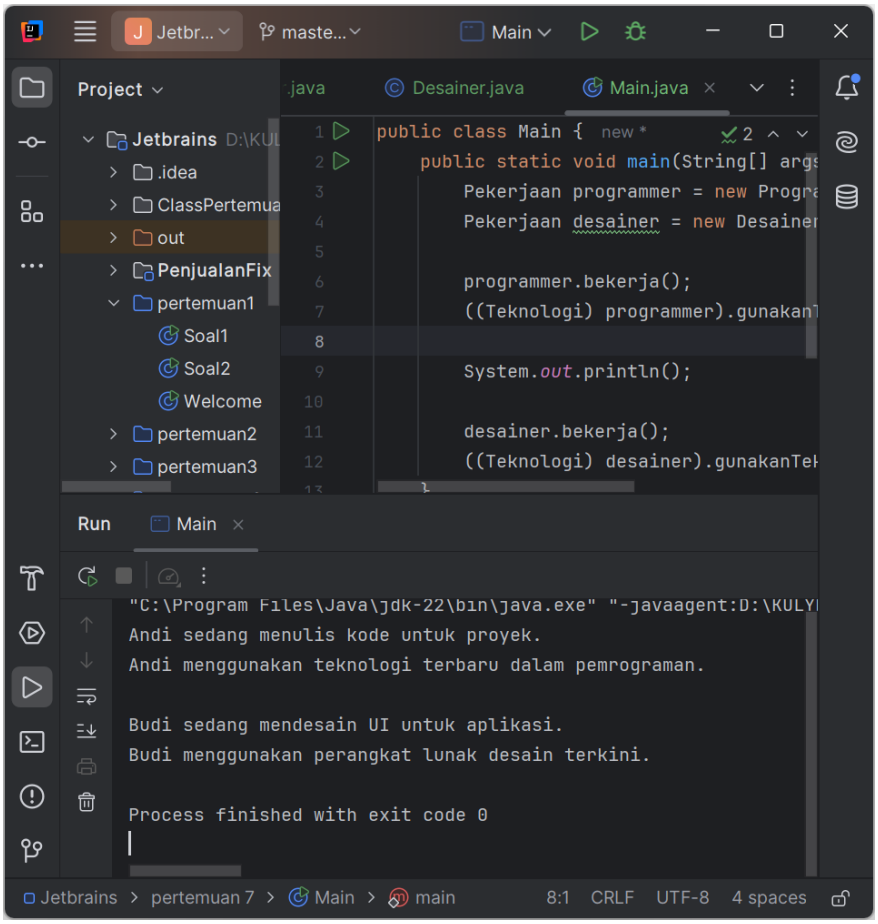
4. Soal 3 : buat programnya
   a. Screenshoot hasil akhir program



   b. Screenshoot setiap jawaban soal yang dipertanyakan.



struktur main, kelas, dan interface



Hasil akhir program

c. Penjelasan program

1.**Kelas Abstrak Pekerjaan**

```
public abstract class Pekerjaan {
    protected String nama;
    protected int id;
    protected double gaji;
    protected String departemen;

    public Pekerjaan(String nama, int id, double gaji, String
departemen) {
        this.nama = nama;
        this.id = id;
        this.gaji = gaji;
        this.departemen = departemen;
    }

    public abstract void bekerja();
}
```

Penjelasan:
- Pekerjaan adalah kelas abstrak yang mewakili konsep umum dari sebuah pekerjaan. Kelas ini memiliki variabel yang diturunkan ke subclass, seperti nama, id, gaji, dan departemen.
- Kelas ini juga memiliki metode abstrak bekerja(), yang harus diimplementasikan oleh setiap subclass, tetapi implementasinya bisa berbeda tergantung pekerjaan spesifiknya.

2. Interface Teknologi

```
public interface Teknologi {
    void gunakanTeknologi();
}
```

Penjelasan:
- Teknologi adalah sebuah interface yang mendefinisikan metode gunakanTeknologi().
- Setiap kelas yang mengimplementasikan interface ini harus menyediakan implementasi konkret untuk metode tersebut.
- Interface ini memungkinkan berbagai pekerjaan yang memanfaatkan teknologi (seperti programmer dan desainer) untuk memiliki kemampuan yang serupa, yaitu penggunaan teknologi, tanpa mengikat mereka ke dalam hierarki yang sama.

3. Kelas Programmer

```
public class Programmer extends Pekerjaan implements Teknologi {
    private String bahasaPemrograman;

    public Programmer(String nama, int id, double gaji, String
departemen, String bahasaPemrograman) {
        super(nama, id, gaji, departemen);
        this.bahasaPemrograman = bahasaPemrograman;
    }

    @Override
    public void bekerja() {
        System.out.println(nama + " sedang menulis kode dengan
bahasa " + bahasaPemrograman);
    }

    @Override
    public void gunakanTeknologi() {
        System.out.println(nama + " menggunakan IDE dan alat
pengembangan lainnya.");
    }
}
```

Penjelasan:
- Programmer adalah subclass dari Pekerjaan yang mewarisi atribut umum seperti nama, id, gaji, dan departemen.
- bahasaPemrograman adalah atribut khusus yang hanya dimiliki oleh Programmer dan tidak ada di kelas Pekerjaan.
- bekerja() diimplementasikan untuk menunjukkan bahwa programmer bekerja dengan menulis kode dalam bahasa pemrograman tertentu.
- gunakanTeknologi() diimplementasikan dari interface Teknologi untuk menunjukkan bahwa seorang programmer menggunakan teknologi terkait pengembangan perangkat lunak.

4. Kelas Desainer

```
public class Desainer extends Pekerjaan implements Teknologi {
    private String alatDesain;

    public Desainer(String nama, int id, double gaji, String
departemen, String alatDesain) {
        super(nama, id, gaji, departemen);
        this.alatDesain = alatDesain;
    }

    @Override
    public void bekerja() {
        System.out.println(nama + " sedang mendesain menggunakan
" + alatDesain);
    }

    @Override
    public void gunakanTeknologi() {
        System.out.println(nama + " menggunakan perangkat lunak
desain grafis.");
    }
}
```

Penjelasan:
- Desainer adalah subclass dari Pekerjaan yang juga mewarisi atribut umum seperti nama, id, gaji, dan departemen.
- alatDesain adalah atribut khusus yang hanya dimiliki oleh Desainer dan tidak ada di kelas Pekerjaan.
- bekerja() diimplementasikan untuk menunjukkan bahwa desainer bekerja dengan mendesain menggunakan alat desain tertentu.

- gunakanTeknologi() diimplementasikan dari interface Teknologi untuk menunjukkan bahwa seorang desainer menggunakan perangkat lunak desain grafis.

5. Main Class

```
public class Main {
    public static void main(String[] args) {
        Programmer programmer = new Programmer("Andi", 101,
8000000, "IT", "Java");
        Desainer desainer = new Desainer("Budi", 102, 7000000,
"Desain", "Photoshop");

        programmer.bekerja();
        programmer.gunakanTeknologi();

        System.out.println();

        desainer.bekerja();
        desainer.gunakanTeknologi();
    }
}
```

Penjelasan:
- Di dalam kelas Main, kita membuat objek Programmer dan Desainer, lalu memanggil metode bekerja() dan gunakanTeknologi() untuk menunjukkan bagaimana keduanya bekerja dengan cara berbeda dan menggunakan teknologi yang berbeda, meskipun sama-sama mewarisi dari kelas abstrak Pekerjaan dan mengimplementasikan interface Teknologi.