LAPORAN 10
Pemrograman Berorientasi Objek

“Interface”

Dosen Pengampu : **Paulina Heruningsih Prima Rosa.**

DIBUAT OLEH :
Nama : Yohanis Celvin D.P.U.Pati
NIM : 245314033

KELAS : BP

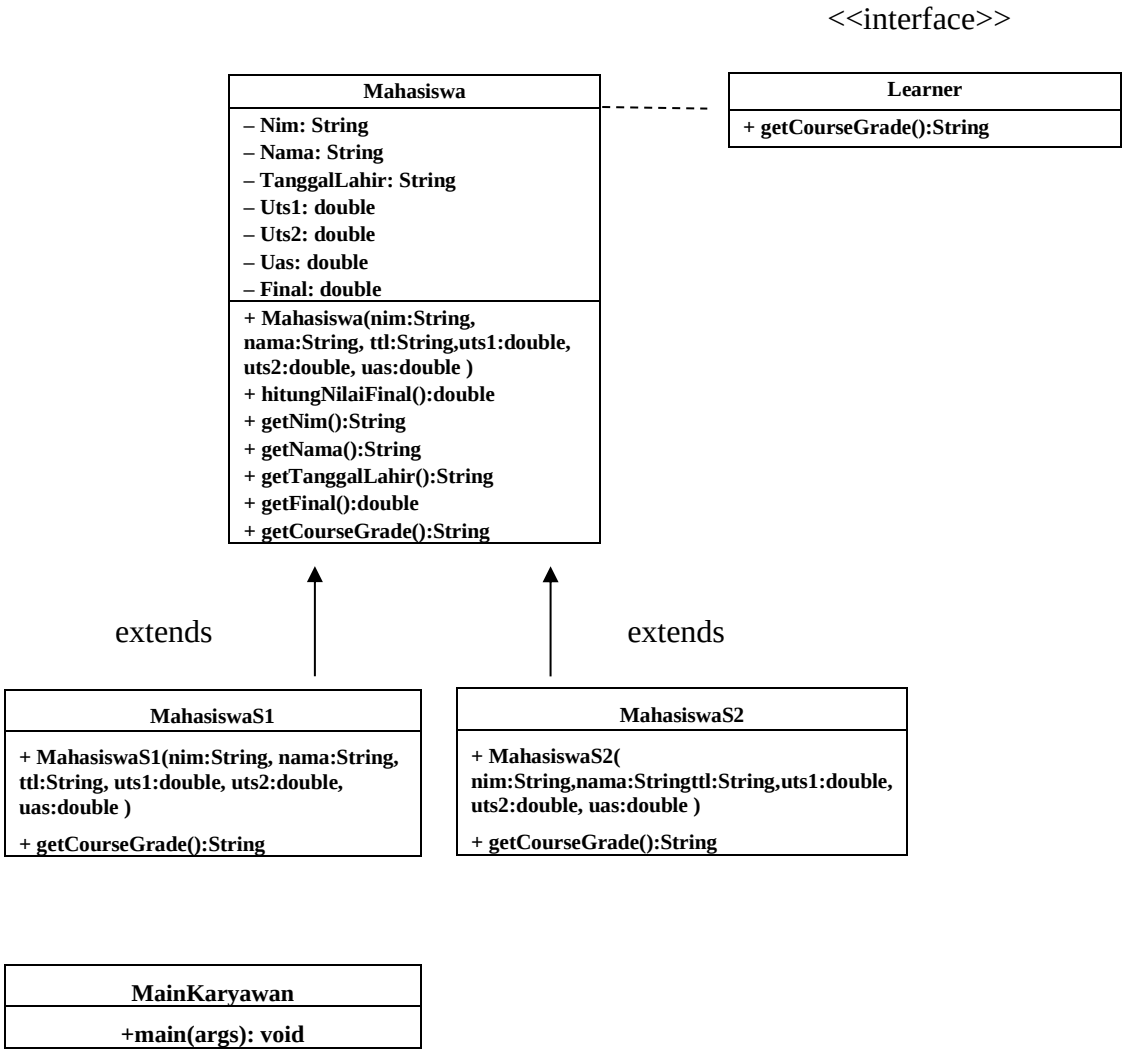
PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS SANATA DHARMA
YOGYAKARTA

2024

Kasus 1

A. DIAGRAM CLASS

Penggambaran diagram kelas



B. LISTING PROGRAM

```
Source  History
/**
 * @author Celvin Pati
 */
class Mahasiswa implements Learner {
    // Atribut dengan tipe protected agar bisa diakses oleh kelas turunan
    protected String Nim;           // Nomor Induk Mahasiswa
    protected String Nama;          // Nama lengkap mahasiswa
    protected String TanggalLahir; // Format: DD MM YYYY
    protected double Uts1;         // Nilai Ujian Tengah Semester 1 (0-100)
    protected double Uts2;         // Nilai Ujian Tengah Semester 2 (0-100)
    protected double Uas;          // Nilai Ujian Akhir Semester (0-100)
    protected double Final;        // Nilai akhir yang dihitung dari UTS1, UTS2, dan UAS
    // Konstruktor untuk inisialisasi data mahasiswa dan hitung nilai akhir
    public Mahasiswa(String Nim, String Nama, String TanggalLahir,
                     double Uts1, double Uts2, double Uas) {
        this.Nim = Nim;
        this.Nama = Nama;
        this.TanggalLahir = TanggalLahir;
        this.Uts1 = Uts1;
        this.Uts2 = Uts2;
        this.Uas = Uas;
        this.Final = HitungNilaiFinal(); // untuk hitung nilai final secara otomatis
    }
    //untuk hitung nilai akhir dengan bobot 30% UTS1, 30% UTS2, dan 40% UAS
    public double HitungNilaiFinal() {
        return (0.3 * Uts1) + (0.3 * Uts2) + (0.4 * Uas);
    }
    // Implementasi metode dari interface Learner
    @Override
    public String getCourseGrade() {
        if (Final >= 80) return "A";
        else if (Final >= 70) return "B";
        else if (Final >= 56) return "C";
        else if (Final >= 45) return "D";
        else return "E";
    }
    // Getter untuk mengakses data privat
    public String getNim() { return Nim; }
    public String getNama() { return Nama; }
    public String getTanggalLahir() { return TanggalLahir; }
    public double getFinal() { return Final; }
}
```

```
/**
 *
 * @author Celvin Pati
 */
interface Learner {
    String getCourseGrade(); // Metode untuk mengembalikan nilai huruf (A-E)
}
```

```
/**
 *
 * @author Celvin Pati
 */
class MahasiswaS1 extends Mahasiswa {
    // Konstruktor panggil konstruktor kelas induk
    public MahasiswaS1(String Nim, String Nama, String TanggalLahir,
                       double Uts1, double Uts2, double Uas) {
        super(Nim, Nama, TanggalLahir, Uts1, Uts2, Uas);
    }

    // Override metode getCourseGrade()
    @Override
    public String getCourseGrade() {
        return super.getCourseGrade();
    }
}
```

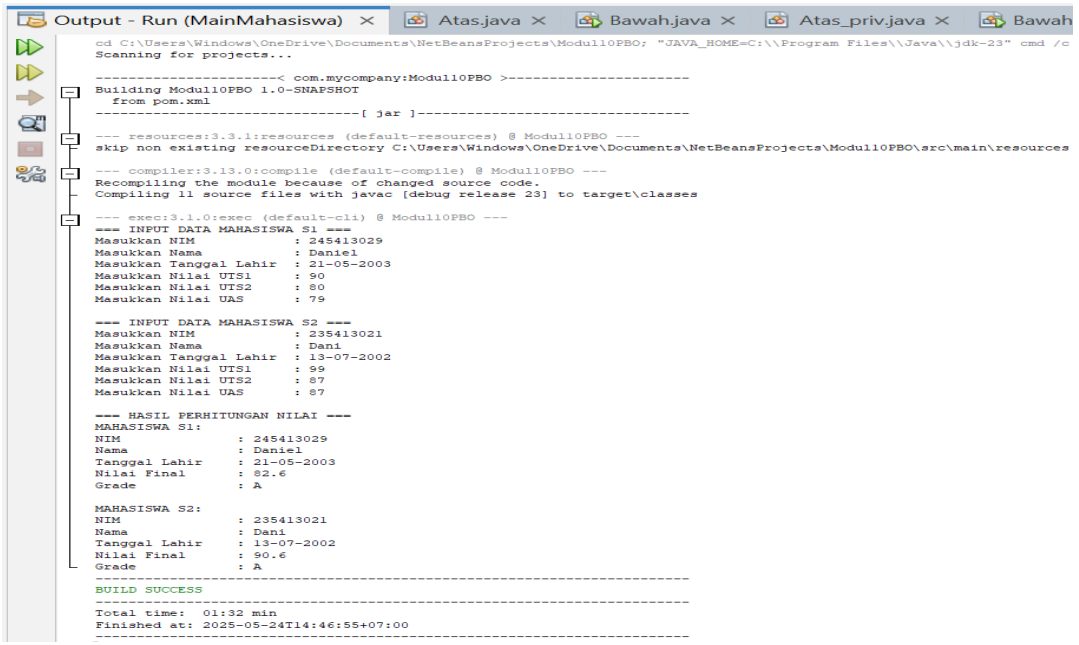
```
/**
 *
 * @author Celvin Pati
 */
class MahasiswaS2 extends Mahasiswa {
    // Konstruktor untuk panggil konstruktor kelas induk
    public MahasiswaS2(String Nim, String Nama, String TanggalLahir,
                       double Uts1, double Uts2, double Uas) {
        super(Nim, Nama, TanggalLahir, Uts1, Uts2, Uas);
    }

    // Override metode getCourseGrade()
    @Override
    public String getCourseGrade() {
        if (Final >= 85) return "A";
        else if (Final >= 70) return "B";
        else if (Final >= 56) return "C";
        else if (Final >= 45) return "D";
        else return "E";
    }
}
```

```
import java.util.Scanner;

/**
 *
 * @author Celvin Pati
 */
import java.util.Scanner;
public class MainMahasiswa {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        // Input Data Mahasiswa S1
        System.out.println("=== INPUT DATA MAHASISWA S1 ===");
        System.out.print("Masukkan NIM             : ");
        String nim1 = input.nextLine();
        System.out.print("Masukkan Nama            : ");
        String nama1 = input.nextLine();
        System.out.print("Masukkan Tanggal Lahir   : ");
        String tl1 = input.nextLine();
        System.out.print("Masukkan Nilai UTS1      : ");
        double uts1_1 = input.nextDouble();
        System.out.print("Masukkan Nilai UTS2      : ");
        double uts2_1 = input.nextDouble();
        System.out.print("Masukkan Nilai UAS       : ");
        double uas1 = input.nextDouble();
        input.nextLine(); // buang newline
        // objek MahasiswaS1
        MahasiswaS1 mhs1 = new MahasiswaS1(nim1, nama1, tl1, uts1_1, uts2_1, uas1);
        //Data Mahasiswa S2
        System.out.println("\n=== INPUT DATA MAHASISWA S2 ===");
        System.out.print("Masukkan NIM             : ");
        String nim2 = input.nextLine();
        System.out.print("Masukkan Nama            : ");
        String nama2 = input.nextLine();
        System.out.print("Masukkan Tanggal Lahir   : ");
        String tl2 = input.nextLine();
        System.out.print("Masukkan Nilai UTS1      : ");
        double uts1_2 = input.nextDouble();
        System.out.print("Masukkan Nilai UTS2      : ");
        double uts2_2 = input.nextDouble();
        System.out.print("Masukkan Nilai UAS       : ");
        double uas2 = input.nextDouble();
        //objek MahasiswaS2
        MahasiswaS2 mhs2 = new MahasiswaS2(nim2, nama2, tl2, uts1_2, uts2_2, uas2);
        //Hasil Perhitungan
        System.out.println("\n--- HASIL PERHITUNGAN NILAI ---");
        System.out.println("MAHASISWA S1:");
        System.out.println("NIM\t\t: " + mhs1.getNim());
        System.out.println("Nama\t\t: " + mhs1.getNama());
        System.out.println("Tanggal Lahir\t: " + mhs1.getTanggalLahir());
        System.out.println("Nilai Final\t: " + mhs1.getFinal());
        System.out.println("Grade\t\t: " + mhs1.getCourseGrade());
        System.out.println("\nMAHASISWA S2:");
        System.out.println("NIM\t\t: " + mhs2.getNim());
        System.out.println("Nama\t\t: " + mhs2.getNama());
        System.out.println("Tanggal Lahir\t: " + mhs2.getTanggalLahir());
        System.out.println("Nilai Final\t: " + mhs2.getFinal());
        System.out.println("Grade\t\t: " + mhs2.getCourseGrade());
        input.close();
    }
}
```

C. OUTPUT Unit Test nya



D. ANALISA

- **Interface Learner**

No	Syntax	penjelasan
1	import java.util.Scanner;	// impor class Scanner dari Java Utility Library untuk input dari pengguna.
2	interface Learner { String getCourseGrade(); }	//Interface Learner mendefinisikan kontrak berupa method getCourseGrade() yang harus diimplementasikan oleh setiap kelas yang mengimplementasikannya.

- **Class Mahasiswa**

No	Syntax	penjelasan
1	class Mahasiswa implements Learner	// Kelas induk Mahasiswa mengimplementasikan Learner, yang artinya wajib menyediakan implementasi getCourseGrade().
2	protected String Nim; ... protected double Final;	// Atribut yang menyimpan data mahasiswa (NIM, nama, tanggal lahir, nilai ujian). Dideklarasikan protected agar bisa diakses oleh subclass-nya (MahasiswaS1 dan MahasiswaS2).
3	public Mahasiswa(...){ ... }	//Konstruktor Mahasiswa yang menerima data input dan langsung menghitung nilai akhir (Final).
4	public double hitungNilaiFinal()	//Method untuk menghitung nilai akhir dari UTS1 (30%), UTS2 (30%), dan UAS (40%).
5	public String getCourseGrade()	//Implementasi dari method interface Learner. Memberikan konversi nilai akhir menjadi huruf untuk Mahasiswa S1.
6	public String getNim() { ... } dsb.	//Getter method untuk mengakses atribut Nim, Nama, TanggalLahir, dan Final dari luar class.

- **Class MahasiswaS1**

No	Syntax	penjelasan
1	class MahasiswaS1 extends Mahasiswa	// Subclass MahasiswaS1 mewarisi dari class Mahasiswa.
2	super(Nim, Nama, TanggalLahir, Uts1, Uts2, Uas);	//Konstruktor subclass yang memanggil konstruktor superclass untuk inisialisasi data.
3	@Override public String getCourseGrade()	//Method ini override method dari Mahasiswa, tapi hanya memanggil ulang (super.getCourseGrade()), karena logikanya sama.

- **Class MahasiswaS2**

No	Syntax	penjelasan
1	class MahasiswaS2 extends Mahasiswa	// Subclass MahasiswaS2, yang juga mewarisi Mahasiswa, tetapi mengubah cara konversi nilai.
2	public MahasiswaS2(String Nim, String Nama, String TanggalLahir, double Uts1, double Uts2, double Uas)	//Konstruktor MahasiswaS2 yang menggunakan parameter untuk mengisi data mahasiswa dan langsung memanggil konstruktor superclass dengan super(...).
3	super(Nim, Nama, TanggalLahir, Uts1, Uts2, Uas);	//Memanggil konstruktor dari Mahasiswa agar data mahasiswa S2 terinisialisasi dengan baik.
4	@Override public String getCourseGrade()	//Override method getCourseGrade() dengan standar penilaian lebih tinggi.
5	if (Final >= 85) return "A"; ...	//Logika konversi nilai Mahasiswa S2. Nilai minimal untuk A lebih tinggi dibandingkan Mahasiswa S1 (85 vs 80).

- **Class MainMahasiswa**

No	Syntax	penjelasan
1	Scanner input = new Scanner(System.in);	// Membuat objek Scanner untuk menerima input dari user melalui terminal.
2	System.out.println("=== INPUT DATA MAHASISWA S1 ===");	//Menampilkan judul sebagai petunjuk kepada user untuk memasukkan data Mahasiswa S1.
3	String nim1 = input.nextLine(); ... double uas1 = input.nextDouble();	//Membaca input data Mahasiswa S1 (NIM, nama, tanggal lahir, nilai UTS1, UTS2, dan UAS).
4	input.nextLine();	//Menghapus karakter newline (\n) yang tertinggal setelah membaca angka.
5	MahasiswaS1 mhs1 = new MahasiswaS1(...);	//Membuat objek Mahasiswa S1 dengan data input.
6	System.out.println("=== INPUT DATA MAHASISWA S2 ===");	//Menampilkan petunjuk untuk input Mahasiswa S2.
7	String nim2 = input.nextLine(); ... double uas2 = input.nextDouble();	//Membaca input untuk Mahasiswa S2.
8	MahasiswaS2 mhs2 = new MahasiswaS2(...);	//Membuat objek Mahasiswa S2 dengan data input yang telah diberikan.

9	System.out.println("\n=== HASIL PERHITUNGAN NILAI ===");	//Menampilkan header output hasil perhitungan.
10	System.out.println("MAHASISWA S1: ...	//Menampilkan detail data Mahasiswa S1, nilai akhirnya, dan grade-nya.
11	System.out.println("MAHASISWA S2: ...	//Menampilkan detail data Mahasiswa S2, nilai akhirnya, dan grade-nya.
12	input.close();	//Menutup objek Scanner untuk membersihkan resource.

➢ Penjelasan alur jalannya program, penggunaan, dan manfaat interface dalam program:

1) Kelas Learner (Interface)

   Kelas ini adalah kontrak kusus yang diimplementasi daro kelas-kelas apapun yang mau di anggap sebagai “pembelajaran”

   Fungsinya untuk memaksa kelas turunan (Mahasiswa) supaya menyediakan cara untuk mengubah nilai numerik menjadi grade huruf (A-E).

2) Kelas Mahasiswa (Supperclass)

   Kelas ini adalah kelas dasar untuk MahasiswaS1 dan MahasiswaS2, dan kelas ini mengimplementasi interface Learner, jadi wajib menyediakan implementasi getCourceGrade().

   Fungsionalnya:

   - Konstruktor untuk mengisi semua atribut saat objek dibuat.
   - hitungNilaiFinal() untuk menghitung nilai akhir dari 30% UTS1, 30% UTS2, 40% UAS.
   - getCourseGrade() untuk mengonversi nilai akhir menjadi huruf berdasarkan standar mahasiswa S1.

3) Kelas MahasiswaS1

   Kelas ini adalah subclass dari Mahasiswa yang mewakili jenjang S1, dan kelas ini menggunakan metode Override, metode getCourseGrede(), meskipun logikanya sama dengan kelas induk, tujuannya supaya bisa menyesuaikan jika nanti kriteria penilaian S1 berubah, dan kelas ini tetap memakai logikakonversi nilai default (A $\geq$ 80, B $\geq$ 70, dst).

4) Kelas MahasiswaS2

   Kelas ini juga subcass dari Mahasiswa dan kelas ini mewakili mahasiswa jenjang S2, kelas ini juga memakai metode override metode getCourseGrade() dengan logika yang lebih ketat, misalnya: A ≥ 85 (lebih tinggi dari S1) dan B ≥ 70, C ≥ 56, dst.

   Perubahan logika ini memungkinkan adanya penyesuaian standar akademik untuk tingkat pendidikan yang lebih tinggi.

5) Kelas MainMahasiswa (Fungsi main())

   Kelas ini berfungsi sebagai titik awal program (entry point), fungsinya untuk Meminta input dari user untuk dua mahasiswa: satu S1 dan satu S2, Membuat objek MahasiswaS1 dan MahasiswaS2 berdasarkan input, Menampilkan data mahasiswa beserta nilai final dan grade-nya, dan output yang ditampilkan memberi gambaran hasil konversi nilai tiap jenjang.

➢ Penggunaan dan Manfaat Interface

1) Interface Learner interface dipakai di baris ini:

```
interface Learner {
   String getCourseGrade();
    }
```

Yang kemudian diimplementasikan oleh:

```
class Mahasiswa implements Learner {
   ...
}
```

No	Manfaat	penjelasan
1	Polimorfisme	// Dengan interface Learner, kita bisa membuat koleksi objek Learner (misalnya array) yang berisi objek MahasiswaS1 dan MahasiswaS2, lalu memanggil getCourseGrade() tanpa peduli tipenya.

2	Reusabilitas dan Fleksibilitas	//Jika nanti dibuat jenis mahasiswa baru (misalnya MahasiswaDiploma), cukup implementasikan Learner, tanpa mengubah kode lama.
3	Pemisahan Kontrak dan Implementasi	//Interface mendefinisikan "apa yang harus dilakukan"**,** bukan "bagaimana". Implementasi diserahkan ke masing-masing kelas.
4	Kode Lebih Modular	//Kode menjadi terstruktur dan modular karena logika konversi nilai bisa diubah per kelas tanpa menyentuh interface.

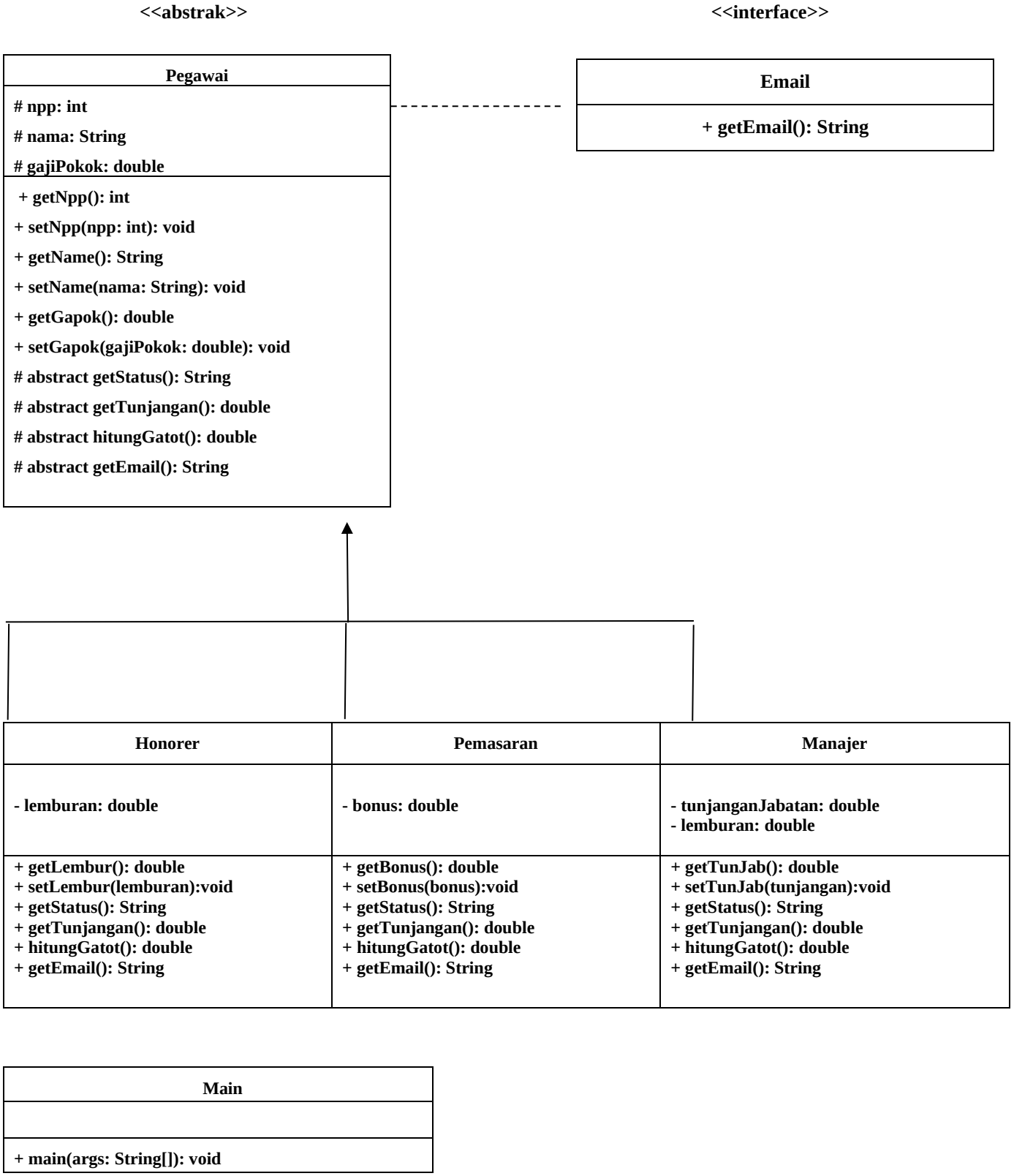
➢ Perbandingan

aspek	Versi Tanpa Interface	Versi Dengan Interface (Learner)
Struktur Kelas	Hanya ada inheritance (kelas induk Mahasiswa, dan subclass MahasiswaS1/MahasiswaS2).	Menggunakan inheritance + **interface Learner** yang mengharuskan method getCourseGrade() diimplementasikan.
Keterikatan antar kelas	Kelas MahasiswaS1 dan MahasiswaS2 bergantung langsung ke kelas Mahasiswa.	Kelas-kelas tetap tergantung ke Mahasiswa, tetapi kontrak perilaku (nilai → grade) ditetapkan oleh Learner.
Polimorfisme (berbasis interface)	Tidak mendukung penggunaan objek secara polymorphic berdasarkan perilaku (getCourseGrade()).	Mendukung polimorfisme berbasis interface, misalnya kita bisa buat: List<Learner> daftarMahasiswa = new ArrayList<>(); lalu

		isinya bisa MahasiswaS1, MahasiswaS2, dst.
Ekstensi program	Sulit dikembangkan jika ingin menambahkan jenis mahasiswa baru (misal: D3, mahasiswa exchange, dll), karena harus bergantung ke kelas Mahasiswa.	Lebih mudah dikembangkan: tinggal buat kelas baru yang mengimplementasikan Learner saja, tidak wajib mewarisi Mahasiswa.
Keterbukaan terhadap perubahan (Open/Closed Principle)	Jika logika getCourseGrade() diubah, perlu menyentuh langsung kelas Mahasiswa atau turunannya.	Logika perubahan dapat dilakukan per kelas (terpisah) tanpa mengubah interface Learner.
Konsistensi perilaku	Tidak ada jaminan semua subclass memiliki metode getCourseGrade().	Ada **kontrak wajib** yang menjamin semua kelas yang "belajar" pasti punya getCourseGrade().
Reusabilitas	Terbatas pada hierarki warisan (extends).	Tinggi – berbagai kelas bisa mengimplementasikan interface Learner, meskipun tidak berhubungan secara hierarki kelas.
Kode lebih fleksibel?	Tidak fleksibel.	Ya, fleksibel dan scalable.

Kasus 2

A. DIAGRAM CLASS



B. LISTING PROGRAM

```java
 * @author Celvin Pati
 */
abstract class Pegawai implements Email {
    protected int npp;           // Nomor pegawai
    protected String nama;       // Nama lengkap pegawai
    protected double gajiPokok;  // Gaji pokok pegawai
    // Getter untuk npp
    public int getNpp() {
        return npp;
    }
    // Setter untuk npp
    public void setNpp(int npp) {
        this.npp = npp;
    }
    // Getter untuk nama
    public String getName() {
        return nama;
    }
    // Setter untuk nama
    public void setName(String nama) {
        this.nama = nama;
    }
    // Getter untuk gaji pokok
    public double getGapok() {
        return gajiPokok;
    }
    // Setter untuk gaji pokok
    public void setGapok(double gapok) {
        this.gajiPokok = gapok;
    }

    // Method abstract
    abstract String getStatus();      // Status jabatan
    abstract double getTunjangan();   // Besar tunjangan
    abstract double hitungGatot();    // Total gaji (gapok + tunjangan)
    public abstract String getEmail(); // Alamat email sesuai format
}
```

```java
/**
 *
 * @author Celvin Pati
 */
interface Email {
    String getEmail();   // inisialisasi metode untuk  email
}
```

```java
 * @author Celvin Pati
 */
class Honorer extends Pegawai {
    protected double lemburan; // Jumlah lembur
    // Getter lemburan
    public double getLembur() {
        return lemburan;
    }
    // Setter lemburan
    public void setLembur(double lemburan) {
        this.lemburan = lemburan;
    }
    // implementasi status
    @Override
    String getStatus() {
        return "Honorer";
    }
    // Implementasi tunjangan
    @Override
    double getTunjangan() {
        return lemburan;
    }
    // Implementasi total gaji
    @Override
    double hitungGatot() {
        return getGapok() + lemburan;
    }
    // Implementasi email: nip@honorer.usd.ac.id
    @Override
    public String getEmail() {
    public String getEmail() {
        return getNpp() + "@honorer.usd.ac.id";
    }
}
```

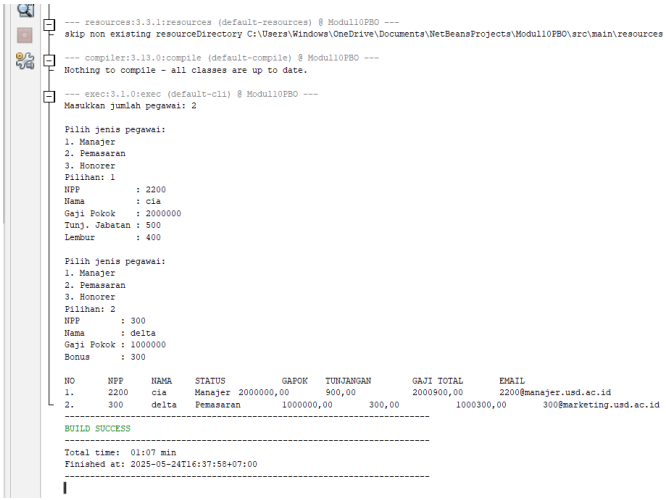
```java
 * @author Celvin Pati
 */
class Manajer extends Pegawai {
    protected double tunjanganJabatan; // Tunjangan jabatan
    protected double lemburan;         // Lembur
    // Getter tunjangan jabatan
    public double getTunJab() {
        return tunjanganJabatan;
    }
    // Setter tunjangan jabatan
    public void setTunJab(double tunjanganJabatan) {
        this.tunjanganJabatan = tunjanganJabatan;
    }
    // Getter lembur
    public double getLembur() {
        return lemburan;
    }
    // Setter lembur
    public void setLembur(double lemburan) {
        this.lemburan = lemburan;
    }
    // Implementasi status
    @Override
    String getStatus() {
        return "Manajer";
    }
    // Implementasi tunjangan = tunjanganJabatan + lembur
    @Override
    double getTunjangan() {
        return tunjanganJabatan + lemburan;
    }
    // Implementasi total gaji
    @Override
    double hitungGatot() {
        return getGapok() + tunjanganJabatan + lemburan;
    }
    // Implementasi email: nip@manajer.usd.ac.id
    @Override
    public String getEmail() {
        return getNpp() + "@manajer.usd.ac.id";
    }
}
```

```java
 * @author Celvin Pati
 */
class Pemasaran extends Pegawai {
    protected double bonus; // Besar bonus
    // Getter bonus
    public double getBonus() {
        return bonus;
    }
    // Setter bonus
    public void setBonus(double bonus) {
        this.bonus = bonus;
    }
    // Implementasi status
    @Override
    String getStatus() {
        return "Pemasaran";
    }
    // Implementasi tunjangan
    @Override
    double getTunjangan() {
        return bonus;
    }
    // Implementasi total gaji
    @Override
    double hitungGatot() {
        return getGapok() + bonus;
    }
    // Implementasi email: nip@marketing.usd.ac.id
    @Override
    public String getEmail() {
    public String getEmail() {
        return getNpp() + "@marketing.usd.ac.id";
    }
}
```

```java
 * @author Celvin Pati
 */
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in); // Scanner untuk baca input dari pengguna
        //untuk minta jumlah pegawai
        System.out.print("Masukkan jumlah pegawai: ");
        int jumPeg = input.nextInt();
        input.nextLine(); // buang sisa newline
        // array untuk menampung objek Pegawai
        Pegawai[] karyawan = new Pegawai[jumPeg];

        // Loop untuk mengisi data setiap pegawai
        for (int i = 0; i < jumPeg; i++) {
            System.out.println("\nPilih jenis pegawai:");
            System.out.println("1. Manajer");
            System.out.println("2. Pemasaran");
            System.out.println("3. Honorer");
            System.out.print("Pilihan: ");
            int pilihan = input.nextInt();
            input.nextLine(); // buang newline
            switch (pilihan) {
                case 1: {
                    // Input data Manajer
                    Manajer m = new Manajer();
                    System.out.print("NPP          : "); m.setNpp(input.nextInt()); input.nextLine();
                    System.out.print("Nama         : "); m.setName(input.nextLine());
                    System.out.print("Gaji Pokok   : "); m.setGapok(input.nextDouble());
                    System.out.print("Tunj. Jabatan : "); m.setTunJab(input.nextDouble());
                    System.out.print("Lembur       : "); m.setLembur(input.nextDouble());
                    karyawan[i] = m; // Simpan ke array
                    break;
                }
                case 2: {
                    // Input data Pemasaran
                    Pemasaran p = new Pemasaran();
                    System.out.print("NPP        : "); p.setNpp(input.nextInt()); input.nextLine();
                    System.out.print("Nama       : "); p.setName(input.nextLine());
                    System.out.print("Gaji Pokok : "); p.setGapok(input.nextDouble());
                    System.out.print("Bonus      : "); p.setBonus(input.nextDouble());
                    karyawan[i] = p; // Simpan ke array
                    break;
                }
                case 3: {
                    // Input data Honorer
                    Honorer h = new Honorer();
                    System.out.print("NPP        : "); h.setNpp(input.nextInt()); input.nextLine();
                    System.out.print("Nama       : "); h.setName(input.nextLine());
                    System.out.print("Gaji Pokok : "); h.setGapok(input.nextDouble());
                    System.out.print("Lembur     : "); h.setLembur(input.nextDouble());
                    karyawan[i] = h; // Simpan ke array
                    break;
                }
                default:
                    // Jika pilihan tidak valid, ulangi
                    System.out.println("Pilihan tidak valid, silakan ulang.");
                    i--; // Kembali ke i yang sama
            }
        }
        //untuk cetak header tabel hasil
        System.out.println("\nNO\tNPP\tNAMA\tSTATUS\t\tGAPOK\tTUNJANGAN\tGAJI TOTAL\tEMAIL");
        // Loop untuk menampilkan data tiap pegawai
        for (int i = 0; i < jumPeg; i++) {
            Pegawai p = karyawan[i];
            System.out.printf(
                "%d.\t%d\t%s\t%s\t%.2f\t%.2f\t\t%.2f\t%s%n",
                i+1,            // Nomor urut
                p.getNpp(),     // NPP
                p.getName(),    // Nama
                p.getStatus(),  // Status jabatan
                p.getGapok(),   // Gaji pokok
                p.getTunjangan(), // Tunjangan
                p.hitungGatot(),  // Gaji total
                p.getEmail()      // Email
            );
        }
        input.close(); // Tutup scanner
    }
}
```

C. OUTPUT



D. ANALISA

- **Interface Email**

No	Syntax	penjelasan
1	interface Email { String getEmail(); }	//Interface Email hanya mendefinisikan satu method getEmail() yang wajib diimplementasikan oleh setiap kelas yang mengimplementasikannya. Tujuannya adalah untuk memastikan bahwa setiap objek pegawai memiliki alamat email sesuai format masing-masing.

- **Class Abstrak Pegawai**

No	Syntax	penjelasan
1	abstract class Pegawai implements Email { ... }	// Mendefinisikan kelas abstrak Pegawai yang mewarisi interface Email. Tidak dapat diinstansiasi secara langsung.
2	protected int npp; protected String nama; protected double gajiPokok;	//Atribut dasar dari setiap pegawai: nomor pegawai, nama, dan gaji pokok.
3	public int getNpp() { return npp; }	//Getter untuk mengambil nilai atribut npp.
4	public void setNpp(int npp) { this.npp = npp; }	//Setter untuk menetapkan nilai npp.
5	abstract String getStatus();	//Method abstrak untuk mendapatkan status jabatan, wajib diimplementasikan oleh subclass.
6	abstract double getTunjangan();	//Method abstrak untuk mendapatkan tunjangan total, wajib diimplementasikan oleh subclass.
7	abstract double hitungGatot();	//Method abstrak untuk menghitung total gaji (gaji pokok + tunjangan).
8	public abstract String getEmail();	//Method abstrak dari interface Email, mengembalikan alamat email pegawai.

- **Class Honorer**

No	Syntax	penjelasan
1	class Honorer extends Pegawai { ... }	// Kelas Honorer adalah subclass dari Pegawai.
2	protected double lemburan;	//Menyimpan nilai lembur untuk pegawai honorer.

3	public double getLembur() { return lemburan; }	//Getter untuk mengambil jumlah lemburan.
4	public void setLembur(double lemburan) { this.lemburan = lemburan; }	//Setter untuk menetapkan jumlah lemburan.
5	String getStatus() { return "Honorer"; }	//Implementasi method abstrak, mengembalikan status pegawai "Honorer".
6	double getTunjangan() { return lemburan; }	//Implementasi method abstrak, tunjangan hanya dari lemburan.
7	double hitungGatot() { return getGapok() + lemburan; }	//Menghitung total gaji = gaji pokok + lembur.
8	public String getEmail() { return getNpp() + "@honorer.usd.ac.id"; }	//Format email khusus untuk pegawai honorer.

- **Kelas Pemasaran**

No	Syntax	penjelasan
1	class Pemasaran extends Pegawai { ... }	// Subclass dari Pegawai khusus untuk divisi pemasaran.
2	protected double bonus;	//Menyimpan nilai bonus pemasaran.
3	public double getBonus() { return bonus; }	//Getter untuk mengambil nilai bonus.
4	public void setBonus(double bonus) { this.bonus = bonus; }	//Setter untuk menetapkan nilai bonus.
5	String getStatus() { return "Pemasaran"; }	//Menyatakan status pegawai sebagai "Pemasaran".
6	double getTunjangan() { return bonus; }	//Tunjangan hanya berasal dari bonus.

7	double hitungGatot() { return getGapok() + bonus; }	//Total gaji = gaji pokok + bonus.

8	public String getEmail() {
	return getNpp() +
	"@marketing.usd.ac.id";
	}
		//Email khusus untuk pegawai pemasaran.

- Kelas Manajer

No	Syntax	penjelasan
1	class Manajer extends Pegawai { ... }	// Subclass dari Pegawai khusus untuk posisi manajerial.
2	protected double tunjanganJabatan; protected double lemburan;	//Menyimpan tunjangan jabatan dan lembur.
3	public double getTunJab() { return tunjanganJabatan; }	//Getter untuk tunjangan jabatan.
4	public void setTunJab(double tunjanganJabatan) { this.tunjanganJabatan = tunjanganJabatan; }	//Setter tunjangan jabatan.
5	public double getLembur() { return lemburan; }	//Getter lembur.
6	public void setLembur(double lemburan) { this.lemburan = lemburan; }	//Setter lembur.
7	String getStatus() { return "Manajer";	//Status sebagai "Manajer".
8	double getTunjangan() { return tunjanganJabatan + lemburan; }	//Tunjangan berasal dari jabatan + lembur.
9	double hitungGatot() { return getGapok() + tunjanganJabatan + lemburan; }	//Total gaji manajer.
10	public String getEmail() { return getNpp() + "@manajer.usd.ac.id"; }	//Email khusus untuk manajer.

➢ Penjelasan alur jalannya program, penggunaan, dan manfaat interface dalam program:

1) Inisialisasi Program

   Program dimulai dari method main di dalam kelas Main. Di sini pengguna diminta menginput jumlah pegawai yang akan didata.

2) Perulangan Input Data Pegawai

   Program melakukan loop sebanyak jumlah pegawai yang dimasukkan. Dalam setiap iterasi, pengguna akan memilih jenis pegawai:

   - Manajer
   - Pemasaran
   - Honorer

3) Input dan Penyimpanan Data Pegawai

   Sesuai pilihan:

   - Objek pegawai dari kelas yang sesuai (Manajer, Pemasaran, atau Honorer) dibuat.
   - Data seperti npp, nama, gaji pokok, dan tunjangan khusus diinputkan oleh pengguna.
   - Objek tersebut disimpan ke dalam array Pegawai[].

4) Output Hasil Data Pegawai

   Setelah semua data dimasukkan, program akan mencetak tabel yang berisi:

   - Nomor pegawai
   - NPP
   - Nama
   - Status (jabatan)
   - Gaji pokok
   - Tunjangan
   - Total gaji
   - Email

➢ Penggunaan interface dalam program

1) Interface Email

```
interface Email {
  String getEmail();
  }
```

- Digunakan sebagai kontrak bahwa semua kelas pegawai harus memiliki method getEmail().
- Diimplementasikan oleh kelas Pegawai, dan diteruskan ke semua turunannya (Manajer, Pemasaran, Honorer).
- Setiap kelas memberikan format email yang berbeda**.**

➢ Manfaat interface dalam program

Manfaat	Penjelasan
Konsistensi	Memastikan bahwa semua jenis pegawai memiliki method getEmail() meskipun formatnya berbeda.
Polimorfisme	Memungkinkan pemanggilan getEmail() tanpa peduli tipe objeknya (Manajer, Honorer, dll.) selama objek tersebut bertipe Email.
Fleksibilitas	Jika di masa depan ada jenis pegawai baru (misalnya Freelancer), cukup tambahkan kelas yang implement Email. Tidak perlu ubah kode lama.
Pemisahan Tanggung Jawab	Memisahkan logika utama data pegawai (NPP, nama, gaji, dll) dari tanggung jawab membuat email.
Keterbacaan & Maintainability	Lebih mudah dibaca dan dipelihara karena struktur program modular dan memiliki kontrak yang jelas.

➢ Perbandingan

Aspek	Program Tanpa Interface	Program Dengan Interface (Email)
Struktur Kode	Semua method didefinisikan langsung di kelas turunan atau abstrak.	Menggunakan interface Email untuk mendefinisikan kontrak getEmail().
Abstraksi Email	Method getEmail() tetap ada, tapi tidak dijamin dimiliki semua kelas.	Semua kelas pegawai **wajib** memiliki getEmail() karena terikat kontrak interface.
Konsistensi Method	Bisa saja kelas baru lupa menambahkan method getEmail().	Interface memastikan semua kelas pegawai punya method getEmail().
Polimorfisme	Tidak bisa menggunakan instanceof Email atau casting ke interface.	Bisa memanfaatkan polymorphism, misal: (Email) obj.getEmail()
Prinsip OOP (SOLID)	Kurang mendukung prinsip Interface Segregation & Dependency Inversion.	Lebih dekat dengan prinsip-prinsip OOP, terutama Interface Segregation**.**