LAPORAN PRATIKUM V

PEMOGRAMAN BERBASI OBJEK

"Abstract Class, Interface, Final Class, Inner Class"

Disusun oleh :

Naufal Dira Agustian - 22115330011

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

2025

1. Pendahuluan

Pemrograman Berorientasi Objek (PBO) merupakan paradigma pemrograman yang fundamental dalam pengembangan perangkat lunak modern. Untuk membangun

aplikasi yang kompleks, PBO menyediakan mekanisme untuk mengelola kompleksitas tersebut secara efektif.

Praktikum kali ini berfokus pada konsep-konsep lanjutan dalam PBO, yaitu **Abstract Class**, **Interface**, **Final Class**, dan **Inner Class**. Pemahaman akan konsep-konsep ini sangat penting karena memungkinkan perancangan sistem yang lebih modular, fleksibel, dan mudah dikelola. Laporan ini akan menjabarkan implementasi dari konsep-konsep tersebut dalam sebuah studi kasus sistem manajemen kendaraan perusahaan transportasi, mulai dari perancangan kelas hingga analisis hasil program.

2. Tujuan Praktikum

Mahasiswa mampu memahami dan mengimplementasikan konsep *abstract class*, *abstract method*, *interface*, *final class*, dan *inner class* ke dalam kode program sehingga sistem dapat dirancang dengan lebih modular.

3. Dasar Teori

Praktikum ini mencakup beberapa konsep penting dalam Pemrograman Berorientasi Objek (PBO) di Java:

a. **Abstract Class:** Adalah kelas yang masih abstrak dan tidak dapat diinstansiasi secara langsung. Kelas ini berfungsi sebagai kerangka (*blueprint*) untuk kelas turunannya dan dideklarasikan dengan kata kunci abstract.

b. **Abstract Method:** Adalah metode yang dideklarasikan tanpa implementasi (tanpa *body*) di dalam *abstract class*. Metode ini memaksa kelas turunan untuk mengimplementasikannya.

c. **Interface:** Adalah *blueprint* yang mendefinisikan sekumpulan *abstract method*. Sebuah kelas dapat mengimplementasikan lebih dari satu *interface* (mendukung *multiple inheritance*) menggunakan kata kunci implements.

d. **Final Class:** Adalah kelas yang tidak bisa diwariskan atau diturunkan lagi. Ini berguna ketika kita ingin implementasi sebuah kelas tidak boleh dimodifikasi.

e. **Final Method:** Adalah metode yang tidak bisa di-*override* lagi di kelas turunan nya.

f. **Inner Class:** Adalah kelas yang dibuat di dalam kelas lain. Ini biasa digunakan ketika dua kelas saling berhubungan erat dan satu kelas tidak bisa ada tanpa kelas lainnya.

4. Implementasi

Berikut adalah kode program yang diimplementasikan berdasarkan studi kasus pada modul praktikum.

a. Kendaraan.java

Kelas ini adalah **abstract class** yang berfungsi sebagai *blueprint* untuk semua kendaraan. Anda tidak dapat membuat objek new Kendaraan() secara langsung. Kelas ini mendefinisikan properti umum (merk, model, tahun) dan sebuah **abstract method** nyalakanMesin(). Metode abstrak ini tidak memiliki isi dan *memaksa* semua kelas turunan (seperti Mobil) untuk membuat versinya sendiri. Kelas ini juga berisi **final method** tampilkanInfo() , yang artinya metode ini tidak dapat di-override atau diubah oleh kelas turunan mana pun.

```
package pratikum5;

public abstract class Kendaraan {

private String merk;

private String model;

private int tahunProduksi;

public Kendaraan(String merk, String model, int tahunProduksi) {

this.merk = merk;

this.model = model;

this.tahunProduksi = tahunProduksi;

}

public abstract void nyalakanMesin();

public final void tampilkanInfo() {

System.out.println("Merk: " + merk);

System.out.println("Model: " + model);

System.out.println("Tahun Produksi: " + tahunProduksi);

}

public String getMerk() {

return merk;

}
```

```
public String getModel() {

return model;

}

public int getTahunProduksi() {

return tahunProduksi;

}

}
```

b. BahanBakar.java

Ini adalah sebuah **interface** yang bertindak sebagai "kontrak". Kelas mana pun yang mengimplementasikannya (implements BahanBakar) *wajib* menyediakan kode untuk metode jenisBahanBakar(). *Interface* ini juga berisi **default method** infoKonsumsi(). Ini adalah metode yang sudah memiliki isi di dalam *interface*, sehingga kelas yang mengimplementasikannya dapat langsung menggunakannya tanpa harus menuliskannya ulang.

```
package pratikum5;

//interface BahanBakar [cite: 169, 172]

public interface BahanBakar {

// Method [cite: 170]

String jenisBahanBakar();

// Default method [cite: 171, 174]

default void infoKonsumsi() {

System.out.println("Info Konsumsi: Konsumsi bahan bakar tergantung kapasitas mesin");

}

}
```

c. TransportasiUmum.java

Ini adalah sebuah **interface** yang mendemonstrasikan **pewarisan antar-interface**. Dengan extends BahanBakar , ia mewarisi semua kontrak dari BahanBakar. Kelas apa pun yang mengimplementasikan TransportasiUmum (seperti Bus) harus menyediakan implementasi untuk metode kapasitasPenumpang() *dan* semua metode yang diwarisi dari BahanBakar (yaitu jenisBahanBakar()).

```
package pratikum5;

//interface TransportasiUmum extends BahanBakar

public interface TransportasiUmum extends BahanBakar {

// Method [cite: 179, 181]

int kapasitasPenumpang();

}
```

d. Mobil.java

Ini adalah **final class** , yang berarti tidak ada kelas lain yang bisa extends Mobil. Pewarisan berhenti di kelas ini. Kelas ini extends Kendaraan (mewarisi tampilkanInfo()) dan implements BahanBakar (mematuhi kontrak BahanBakar). Oleh karena itu, kelas ini *wajib* menyediakan implementasi untuk nyalakanMesin() (dari Kendaraan) dan jenisBahanBakar() (dari BahanBakar).

```
package pratikum5;

public final class Mobil extends Kendaraan implements BahanBakar {

public String jenisTransmisi;

public Mobil(String merk, String model, int tahunProduksi, String
jenisTransmisi) {

super(merk, model, tahunProduksi);

this.jenisTransmisi = jenisTransmisi;

}

@Override

public void nyalakanMesin() {
```

```
System.out.println("Nyalakan Mesin: Tekan tombol start");

}

@Override

public String jenisBahanBakar() {

return "Bensin";

}

public void fiturMobil() {

System.out.println("Fitur Mobil: Memiliki AC dan audio premium");

}

}
```

e. Bus.java

Ini adalah kelas turunan dari Kendaraan yang mengimplementasikan TransportasiUmum. Karena TransportasiUmum mewarisi BahanBakar, kelas Bus harus menyediakan implementasi untuk nyalakanMesin() (dari Kendaraan), jenisBahanBakar() (dari BahanBakar), dan kapasitasPenumpang() (dari TransportasiUmum). Di dalam kelas Bus juga terdapat **inner class** bernama JadwalPerjalanan. Ini adalah kelas di dalam kelas, digunakan karena jadwal (JadwalPerjalanan) secara logis terikat erat dengan objek bus-nya .

```
package pratikum5;

public class Bus extends Kendaraan implements TransportasiUmum {

String kelasBus;

public Bus(String merk, String model, int tahunProduksi, String kelasBus)
{

super(merk, model, tahunProduksi);

this.kelasBus = kelasBus;

}

@Override
```

```
public void nyalakanMesin() {

System.out.println("Nyalakan Mesin: Putar kunci untuk menyalakan");

}

@Override

public String jenisBahanBakar() {

return "Solar";

}

@Override

public int kapasitasPenumpang() {

return 45;

}

public void fiturBus() {

System.out.println("Fitur Bus: Dilengkapi kursi nyaman dan fasilitas hiburan");

}

public class JadwalPerjalanan {

String rute;

String waktuBerangkat;

public JadwalPerjalanan(String rute, String waktuBerangkat) {

this.rute = rute;

this.waktuBerangkat = waktuBerangkat;

}

// Method inner class [cite: 232, 242]

public void tampilkanJadwal() {
```

```
System.out.println("Jadwal Perjalanan: Rute " + rute + ", Berangkat: " +
waktuBerangkat);

}

}

}
```

f. TransportasiUdara.java

Ini adalah interface lain yang juga extends BahanBakar . Ia menambahkan kontrak baru yang spesifik untuk transportasi udara, yaitu metode jenisPenerbangan().

```
package pratikum5;

//interface TransportasiUdara extends BahanBakar [cite: 262-263]

public interface TransportasiUdara extends BahanBakar {

// Method [cite: 264]

String jenisPenerbangan();

}
```

g. Maskapai.java

Ini adalah interface sederhana yang hanya mendefinisikan satu kontrak, yaitu metode namaMaskapai().

```
package pratikum5;

public interface Maskapai {

// Method [cite: 266]

String namaMaskapai();

}
```

h. Pesawat.java

Kelas ini mendemonstrasikan kemampuan Java untuk **mengimplementasikan banyak *interface*** sekaligus. Ia extends Kendaraan dan implements TransportasiUdara, Maskapai . Karena TransportasiUdara juga mewarisi BahanBakar, kelas Pesawat pada akhirnya harus menyediakan implementasi untuk semua metode abstrak dari Kendaraan

(nyalakanMesin()) dan semua metode dari *interface* TransportasiUdara, BahanBakar, dan Maskapai.

```
package pratikum5;

public class Pesawat extends Kendaraan implements TransportasiUdara,
Maskapai {

public Pesawat(String merk, String model, int tahunProduksi) {

super(merk, model, tahunProduksi);

}

@Override

public void nyalakanMesin() {

System.out.println("Nyalakan Mesin: Bersiap lepas landas");

}

@Override

public String jenisBahanBakar() {

return "Avtur";

}

@Override

public String jenisPenerbangan() {

return "Komersial";

}

@Override

public String namaMaskapai() {

return this.getMerk();

}
```

```
}
```

i. MainApp.java

Ini adalah kelas utama yang berisi public static void main(String[] args), yang merupakan titik awal eksekusi program. Tujuannya adalah untuk membuat objek (instansi) dari Mobil, Bus, dan Pesawat, lalu memanggil metode-metode mereka untuk menguji fungsionalitas dan menghasilkan output yang diminta oleh studi kasus . Di sinilah kita juga melihat cara membuat objek *inner class*, yang memerlukan instansi dari kelas luarnya (bus.new JadwalPerjalanan(...)).

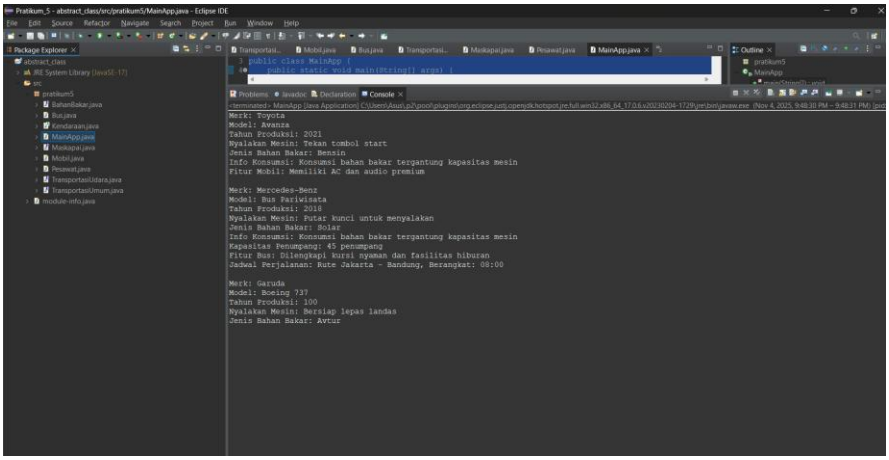
```
package pratikum5;

public class MainApp {

public static void main(String[] args) {

Mobil mobil = new Mobil("Toyota", "Avanza", 2021, "Manual");

mobil.tampilkanInfo();

mobil.nyalakanMesin();

System.out.println("Jenis Bahan Bakar: " + mobil.jenisBahanBakar());

mobil.infoKonsumsi();

mobil.fiturMobil();

System.out.println();

Bus bus = new Bus("Mercedes-Benz", "Bus Pariwisata", 2018,
"Eksekutif");

bus.tampilkanInfo();

bus.nyalakanMesin();

System.out.println("Jenis Bahan Bakar: " + bus.jenisBahanBakar());

bus.infoKonsumsi();

System.out.println("Kapasitas Penumpang: " + bus.kapasitasPenumpang()
+ " penumpang");

bus.fiturBus();
```

```
Bus.JadwalPerjalanan jadwal = bus.new JadwalPerjalanan("Jakarta - Bandung", "08:00");

jadwal.tampilkanJadwal();

System.out.println();

Pesawat pesawat = new Pesawat("Garuda", "Boeing 737", 100);

pesawat.tampilkanInfo();

pesawat.nyalakanMesin();

System.out.println("Jenis Bahan Bakar: " + pesawat.jenisBahanBakar());

}

}
```

5. Hasil dan Pembahasan

I. Hasil (Output Program)

Setelah MainApp.java dijalankan, output yang dihasilkan pada konsol adalah sebagai berikut, sesuai dengan yang diharapkan oleh modul praktikum .



II. Pembahasan

Program di atas berhasil mengimplementasikan semua konsep yang diminta dalam studi kasus:

1. **Abstract Class (Kendaraan)**: Kelas Kendaraan tidak bisa dibuat objeknya secara langsung. Kelas ini menjadi *blueprint* bagi Mobil, Bus, dan Pesawat. Ketiga kelas turunan tersebut *harus* mengimplementasikan *abstract method* nyalakanMesin() dengan cara mereka masing-masing.

2. **Interface (BahanBakar, TransportasiUmum, Maskapai)**:
    - BahanBakar mendefinisikan "kontrak" bahwa setiap kelas yang mengimplementasikannya harus memiliki metode jenisBahanBakar().
    - TransportasiUmum adalah contoh *interface inheritance*, di mana ia mewarisi (extends) BahanBakar dan menambahkan metodenya sendiri (kapasitasPenumpang()).
    - Pesawat menunjukkan kemampuan *multiple inheritance* dengan mengimplementasikan dua interface sekaligus: TransportasiUdara dan Maskapai.
3. **Final Class (Mobil)**: Kelas Mobil dideklarasikan sebagai final. Ini berarti tidak ada kelas lain yang dapat mewarisi (extends) dari kelas Mobil.
4. **Final Method (tampilkanInfo)**: Metode tampilkanInfo di kelas Kendaraan adalah final. Ini mengunci implementasi metode tersebut sehingga kelas turunan seperti Mobil dan Bus tidak dapat meng-override-nya.
5. **Inner Class (JadwalPerjalanan)**: Kelas JadwalPerjalanan dibuat di dalam kelas Bus. Ini logis, karena jadwal perjalanan terikat erat dengan objek bus-nya. Untuk membuatnya, kita memerlukan instansi dari Bus terlebih dahulu (bus.new JadwalPerjalanan(...)).