JobSheet - Week 2 - Object, Class & Encapsulation

**Nama: Ivanka Bunga Aulia**

**NIM: 251524105**

**Kelas: 1D**

**Repo GitHub: [https://github.com/IvankaBunga/Teknik-Pemrograman](https://github.com/IvankaBunga/Teknik-Pemrograman)**

**Instruksi Pengerjaan:**

1. Kerjakan 2 soal di bawah ini dengan melengkapi setiap kolom jawaban yang disediakan pada jobsheet ini.
2. Jawaban setiap soal mencakup source code, screenshot hasil dari program yang ditampilkan full screen termasuk taskbar (tambahkan beberapa screenshot jika diperlukan), penjelasan permasalahan dan solusi yang dihadapi, nama teman yang membantu memecahkan masalah (opsional).
3. Dikumpulkan pada Assignment Classroom sesuai dengan deadline yang tertera pada assignment tersebut.
4. Format penamaan file jobsheet: W2_P_<Kelas 1X>_<3 Digit_NIM_Terakhir>.docx/pdf. Contoh: W2_P_1B_001.docx/pdf.
5. Submit semua jawaban dalam bentuk file java pada repository GitHub masing-masing.

# No. 1 Evaluasi & Refactoring Class Restaurant

| **Soal Praktikum** |
| --- |
| Diketahui terdapat dua file program yang digunakan untuk menyimpan daftar menu makanan dan stoknya, yaitu Restaurant.java & RestaurantMain.java.<br><br>**1. Restaurant.java**<br>`public class Restaurant {`<br>`  public String[] nama_makanan;`<br>`  public double[] harga_makanan;`<br>`  public int[] stok;`<br>`  public static byte id = 0;`<br><br>`  public Restaurant() {`<br>`  nama_makanan = new String[10];`<br>`  harga_makanan = new double[10];`<br>`  stok = new int[10];`<br>`  }`<br>`}`<br><br>`public void tambahMenuMakanan(String nama, double harga, int stok) {` |

```
    this.nama_makanan[id] = nama;
    this.harga_makanan[id] = harga;
    this.stok[id] = stok;
}

public void tampilMenuMakanan() {
    for (int i = 0; i <= id; i++) {
        if (!isOutOfStock(i)) {
            System.out.println(
                nama_makanan[i] + "[" + stok[i] + "]" + "\tRp. " + harga_makanan[i]
            );
        }
    }
}

public boolean isOutOfStock(int id) {
    if (stok[id] == 0) {
        return true;
    } else {
        return false;
    }
}

public static void nextId() {
    id++;
}
```

**2. RestaurantMain.java**

```
public class RestaurantMain {
    public static void main(String[] args) {
        Restaurant menu = new Restaurant();

        menu.tambahMenuMakanan("Pizza", 250000, 20);
        Restaurant.nextId();

        menu.tambahMenuMakanan("Spaghetti", 80000, 20);
        Restaurant.nextId();

        menu.tambahMenuMakanan("Tenderloin Steak", 60000, 30);
        Restaurant.nextId();

        menu.tambahMenuMakanan("Chicken Steak", 45000, 30);

        menu.tampilMenuMakanan();
    }
}
```

## A. Instruksi - Analisis Desain Class:

Amati kode program yang diberikan, kemudian jawab pertanyaan berikut:

1. Apakah class Restaurant sudah menerapkan Encapsulation dengan benar? Anda dapat mengacu pada Buku Ajar Tekpro (Teori) - Chapter 2.4 Enkapsulasi (Lihat di Buku Acuan pada Google Classroom) atau Buku Java Core Design Hint Chapter 4.10.
2. Apakah attribute yang diterapkan saat ini aman dari akses secara

langsung?

3. Apakah terdapat pelanggaran prinsip OOP (misalnya public attribute)?

## B. Instruksi - Perbaikan Kode Program:

Lakukan perbaikan desain class dengan ketentuan:

1. Semua attribute harus bersifat private
2. Akses data dilakukan melalui getter dan setter
3. Validasi stok (stok tidak boleh negatif)
4. Pengembangan Fitur (Mini Case) Tambahkan fitur berikut:
   a) Pemesanan menu
   b) Stok otomatis berkurang setelah pemesanan
   c) Pesan ditolak jika stok tidak mencukupi

**Source Code**

Perbaikan Kode Program

**1. MenuMakanan.java**

```
public class MenuMakanan {
    private String nama;
    private double harga;
    private int stok;

    public MenuMakanan(String nama, double harga, int stok) {
        this.nama = nama;
        this.harga = harga;
        this.setStok(stok);
    }

    // Getter dan Setter
    public String getNama() {return nama;}
    public double getHarga() {return harga;}
    public int getStok() {return stok;}

    public void setStok(int stok) {
        if (stok >= 0) {
            this.stok = stok;
        } else {
            System.out.println("Stok tidak boleh negatif!");
        }
    }

    // Method untuk mengurangi stok saat dipesan
    public boolean kurangiStok(int jumlah) {
        if (this.stok >= jumlah) {
            this.stok -= jumlah;
            return true; // Berhasil dikurangi
        }
        return false; // Gagal karena stok kurang
    }
}
```

**2. Restaurant.java**

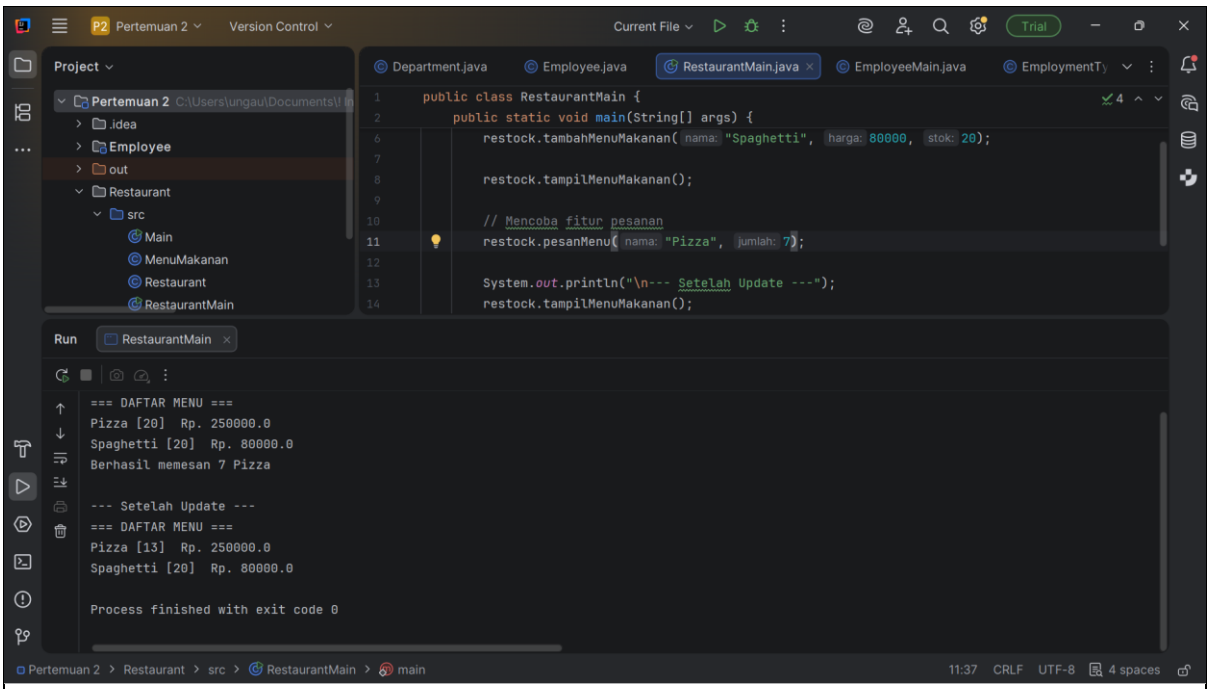
```
public class Restaurant {
    private MenuMakanan[] daftarMenu;
    private static byte id = 0;

    public Restaurant() {
```

```
        // Mengelola array yang isinya adalah OBJEK Makanan
        daftarMenu = new MenuMakanan[10];
    }

    public void tambahMenuMakanan(String nama, double harga, int stok) {
        if (id < daftarMenu.length) {
            daftarMenu[id] = new MenuMakanan(nama, harga, stok);
            id++;
        }
    }

    public void tampilMenuMakanan() {
        System.out.println("=== DAFTAR MENU ===");
        for (int i = 0; i < id; i++) {
            System.out.println(daftarMenu[i].getNama() + " [" + daftarMenu[i].getStok()
+ "]\tRp. " + daftarMenu[i].getHarga());
        }
    }

    // FITUR PEMESANAN
    public void pesanMenu(String nama, int jumlah) {
        for (int i = 0; i < id; i++) {
            if (daftarMenu[i].getNama().equalsIgnoreCase(nama)) {
                if (daftarMenu[i].kurangiStok(jumlah)) {
                    System.out.println("Berhasil memesan " + jumlah + " " + nama);
                } else {
                    System.out.println("Gagal! Stok " + nama + " tidak cukup.");
                }
                return;
            }
        }
        System.out.println("Menu tidak ditemukan!");
    }
}
```

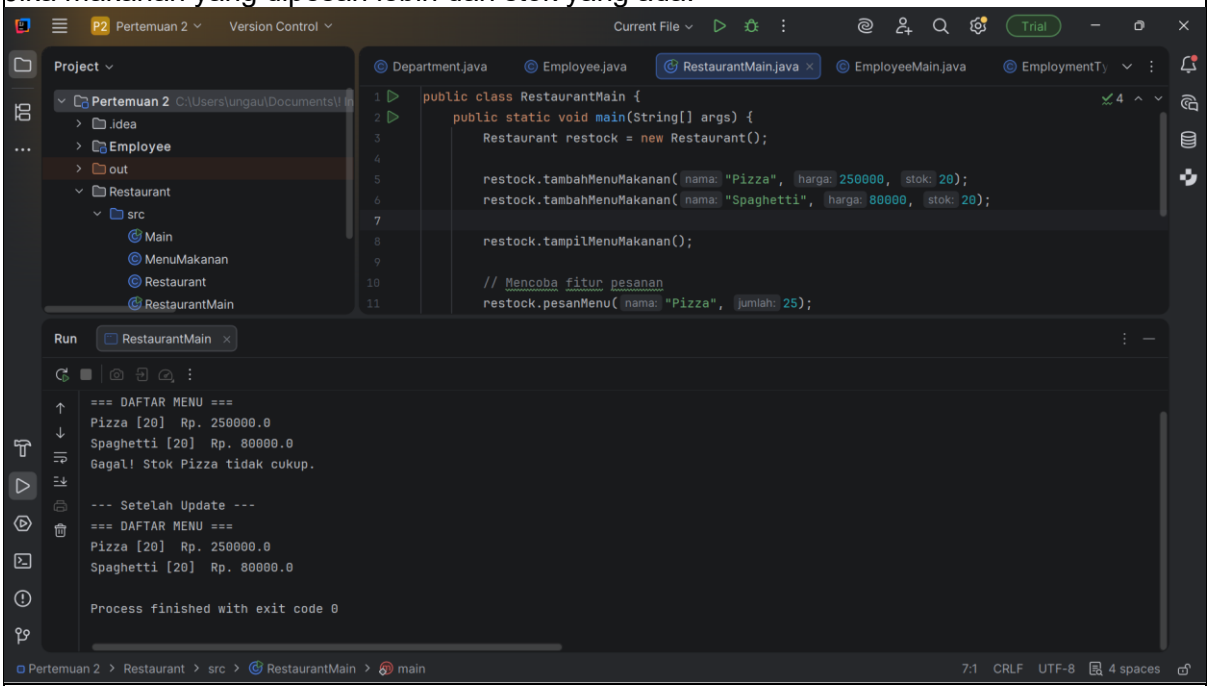
**3. RestaurantMain.java**

```
public class RestaurantMain {
    public static void main(String[] args) {
        Restaurant restock = new Restaurant();

        restock.tambahMenuMakanan("Pizza", 250000, 20);
        restock.tambahMenuMakanan("Spaghetti", 80000, 20);

        restock.tampilMenuMakanan();

        // Mencoba fitur pesanan
        restock.pesanMenu("Pizza", 5);

        System.out.println("\n--- Setelah Update ---");
        restock.tampilMenuMakanan();
    }
}
```

**Screenshot Hasil**




Jika makanan yang dipesan lebih dari stok yang ada:




## Penjelasan Permasalahan dan Solusi

1. Apakah class Restaurant sudah menerapkan Encapsulation dengan benar?
Belum, karena masih menggunakan access modifier public. Ini bisa membuat objek lain mengubah data secara langsung tanpa adanya validasi.

2. Tidak aman. Karena sifatnya public, orang lain bisa mengubah data dari class main. Ini menyebabkan rusaknya integritas sistem karena stok bisa saja diubah menjadi minus.

3. iya, ada dua pelanggaran OOP, yang pertama yaitu tidak ada *information hiding* karena memakai atribut public, harusnya memakai private untuk melindungi integritas data, mencegah manipulasi langsung tanpa adanya validasi, mengurangi ketergantungan antar class, dan memungkinkan perubahan internal tanpa memengaruhi sistem luar. Yang kedua adalah low cohesion, ini terjadi pada penambahan Restaurant.nextId() pada main, seharusnya ditambahkan pada class Restaurant dan fungsi main hanya memanggil method saja.

```
public class Restaurant {
    public String[] nama_makanan;
    public double[] harga_makanan;
    public int[] stok;
    public static byte id = 0;

    public Restaurant() {
        nama_makanan = new String[10];
        harga_makanan = new double[10];
        stok = new int[10];
    }
}
```

Di bagian ini seharusnya tidak ditutup setelah public Restaurant(), yang akan menyebabkan modul lain tidak dapat dipakai. Karena setiap method pada java harus berada dalam suatu class.

| **Nama Teman Hal yang Dibantu (Opsional)** |
| --- |
| M. Haafiz Nayyara |

# No. 2 Interaksi Object dan Struktur Class dalam Java

| **Soal Praktikum** |
| --- |
| Pada pertemuan sebelumnya, mahasiswa telah mempelajari konsep dasar dalam Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) yang menjadi fondasi utama dalam pengembangan aplikasi berbasis Java.<br>1. *Object*<br>Object merupakan representasi nyata dari suatu entitas yang memiliki data (attribute) dan perilaku (method). Object dibuat dari sebuah class dan digunakan untuk menjalankan proses dalam program.<br>2. *Class*<br>Class adalah cetak biru (blueprint) untuk membuat object. Class mendefinisikan struktur data dan perilaku yang dimiliki oleh object, sehingga memungkinkan pembuatan object dengan karakteristik yang sama.<br>3. *Message*<br>Message menggambarkan proses komunikasi antar object. Dalam Java, message diwujudkan dalam bentuk pemanggilan method pada suatu object untuk meminta object tersebut melakukan suatu aksi.<br>4. *Encapsulation*<br>Encapsulation adalah prinsip OOP yang digunakan untuk melindungi data dengan membatasi akses langsung terhadap attribute. Penerapan |

encapsulation dilakukan menggunakan access modifier dan accessor method (getter dan setter).

Pada pertemuan kali ini, mahasiswa akan mempelajari praktik beberapa konsep lanjutan dalam Pemrograman Berorientasi Objek menggunakan Java yang berfokus pada interaksi antar object serta struktur dan organisasi program.

1. *Relationship Between Class*
   Membahas hubungan antar class dalam sebuah sistem, seperti bagaimana satu class dapat menggunakan atau memiliki object dari class lain untuk membentuk sistem yang lebih kompleks.
2. *Static Field & Method*
   Mempelajari penggunaan field dan method yang bersifat static, yaitu milik class dan bukan milik object, serta memahami kapan dan mengapa konsep static digunakan.
3. *Method Parameters*
   Membahas cara pengiriman data ke dalam method melalui parameter, termasuk penggunaan object sebagai parameter untuk memungkinkan interaksi antar object.
4. *Object Construction*
   Mempelajari proses pembuatan object melalui constructor, termasuk inisialisasi attribute dan pengaturan kondisi awal object saat pertama kali dibuat.
5. *Packages*
   Membahas pengelompokan class ke dalam package untuk mengatur struktur program, meningkatkan keterbacaan kode, dan menghindari konflik nama class.
6. JAR
   Mempelajari konsep Java Archive (JAR) sebagai media untuk mengemas dan mendistribusikan aplikasi Java agar dapat dijalankan atau dibagikan dengan lebih mudah.

**Ketentuan Praktikum:**
Pada praktikum ini, mahasiswa diminta untuk memahami contoh program employee dan mengembangkan aplikasi tersebut dengan menerapkan beberapa konsep lanjutan dalam Pemrograman Berorientasi Objek menggunakan Java.

## A. Instruksi - Tulis Ulang Kode Program Employee

### 1. Kelas Department

```
package id.ac.polban.employee.model;

public class Department {
 private String name;

 public Department(String name) {
    this.name = name;
 }

 public String getName() {
    return name;
 }
```

```
 public void setName(String name) {
    this.name = name;
 }
}
```

## 2. Kelas Employee

```
package id.ac.polban.employee.model;

public class Employee {
    private int id;
    private String name;
    private Department department;
    private EmploymentType type;
    private double salary;

    public Employee(int id, String name, Department department,
EmploymentType type, double salary) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.type = type;
        this.salary = salary;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Department getDepartment() {
        return department;
    }
    public void setDepartment(Department department) {
        this.department = department;
    }
    public EmploymentType getType() {
        return type;
    }
    public void setType(EmploymentType type) {
        this.type = type;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

## 3. Kelas EmployeeType

```
package id.ac.polban.employee.model;

public class EmploymentType {
```

```
    private String type;

    public EmploymentType(String type) {
        this.type = type;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
}
```

### 4. Kelas EmployeeService

```
package id.ac.polban.employee.service;

import java.util.HashMap;
import java.util.Map;

import id.ac.polban.employee.model.*;

// mengelola operasi yang berkaitan dengan data dan aturan bisnis
public class EmployeeService {
    private Map<Integer, Employee> employees = new HashMap<>();

    public void addEmployee(Employee emp) {
        employees.put(emp.getId(), emp);
    }

    public Employee getEmployee(int id) {
        return employees.get(id);
    }

    public void raiseSalary(int id, double percent) {
        Employee emp = employees.get(id);
        if (emp != null) {
            emp.setSalary(emp.getSalary() * (1 + percent/100));
        }
    }
}
```

## B. Instruksi - Lengkapi Studi Kasus, Rancangan Class Diagram & Penjelasan:

Adapun ketentuan dan tugas praktikum adalah sebagai berikut:

1. Lengkapi studi kasus yang telah dibuat dengan menerapkan penggunaan *static field* dan *static method* secara tepat!
2. Terapkan konsep package dengan membuat minimal dua package, yaitu:
    - `id.ac.polban.employee.model`
    - `id.ac.polban.employee.service`
2. Buat diagram kelas (**class diagram**) yang menggambarkan struktur class, attribute, method, serta hubungan antar class dalam sistem.
3. Implementasikan relasi antar class pada program, yang mencakup:
    - *Dependency*

- *Aggregation*

2. Jelaskan perbedaan dan fungsi masing-masing jenis relasi tersebut berdasarkan kasus yang dibuat!
3. Lakukan proses generate aplikasi ke dalam file JAR!
4. Buat sebuah project Java baru yang hanya memuat file JAR hasil generate, kemudian jalankan aplikasi dari project tersebut untuk memastikan file JAR dapat digunakan dengan benar!
5. Seluruh source code, diagram kelas, serta penjelasan implementasi didokumentasikan pada jobsheet ini! Pastikan source code di submit pada repository GitHub masing-masing!

**Source Code**

**Employee.java**

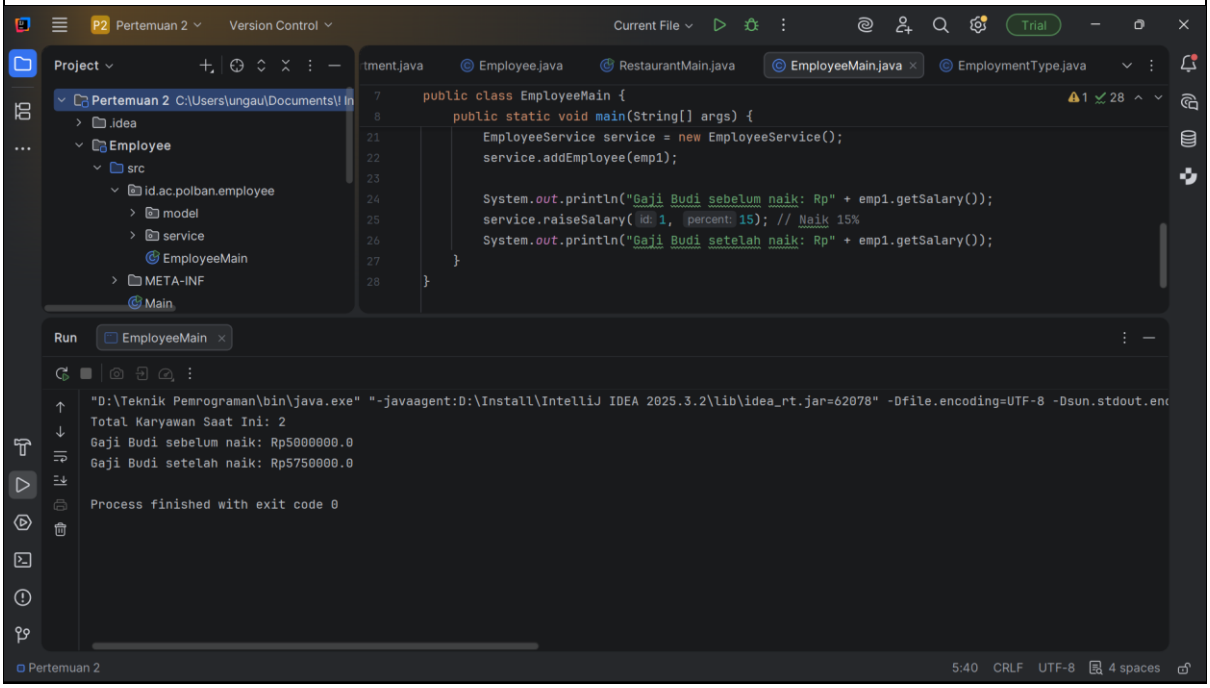
```
package id.ac.polban.employee.model;

public class Employee {
    // 1. Data Karyawan (Private biar aman)
    private int id;
    private String name;
    private Department department; // Relasi Aggregation
    private EmploymentType type;
    private double salary;

    // 2. Penghitung Karyawan (Static)
    private static int totalEmployee = 0;

    // 3. Constructor (Dijalankan saat 'new Employee')
    public Employee(int id, String name, Department dept,
EmploymentType type, double salary) {
        this.id = id;
        this.name = name;
        this.department = dept;
        this.type = type;
        this.salary = salary;

        // Tiap ada objek baru, angka ini naik
        totalEmployee++;
    }

    // 4. Method Static untuk ambil total
    public static int getTotalEmployee() {
        return totalEmployee;
    }

    // 5. Getter & Setter
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public Department getDepartment() { return department; }
    public void setDepartment(Department dept) { this.department =
dept; }

    public EmploymentType getType() { return type; }
    public void setType(EmploymentType type) { this.type = type; }
```

```
    public double getSalary() { return salary; }
  public void setSalary(double salary) { this.salary = salary; }
}
```

**EmployeeMain.java**

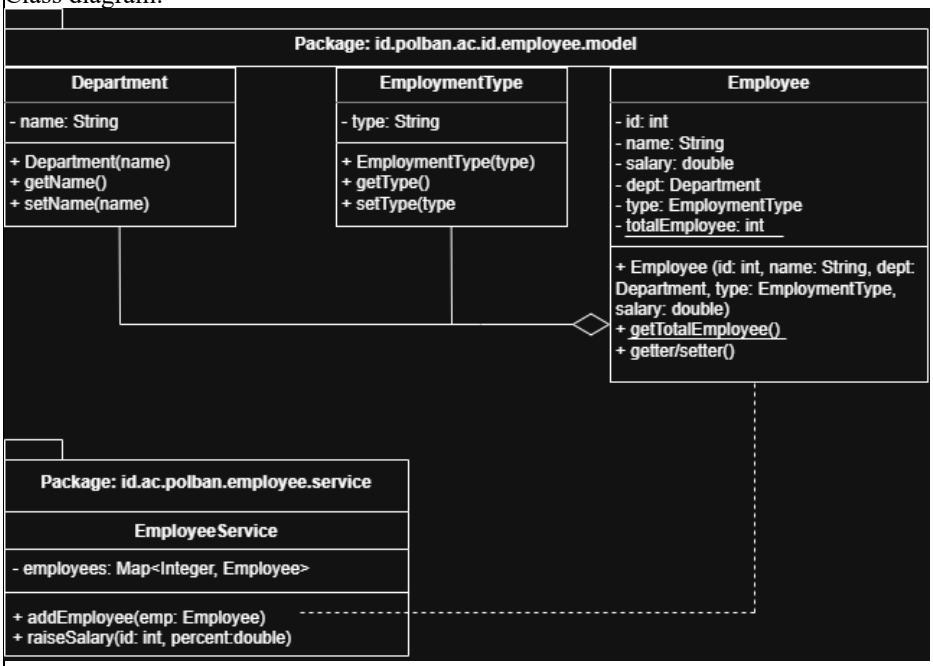
```
package id.ac.polban.employee;

// Mengambil semua isi dari package model dan service
import id.ac.polban.employee.model.*;
import id.ac.polban.employee.service.*;

public class EmployeeMain {
    public static void main(String[] args) {
        // 1. Membuat Departemen dan Tipe (Aggregation)
        Department deptIT = new Department("IT Support");
        EmploymentType permanent = new EmploymentType("Tetap");

        // 2. Menambahkan Karyawan
        Employee emp1 = new Employee(1, "Budi", deptIT, permanent,
5000000);
        Employee emp2 = new Employee(2, "Susi", deptIT, permanent,
6000000);

        // 3. Menampilkan jumlah karyawan
        System.out.println("Total Karyawan Saat Ini: " +
Employee.getTotalEmployee());

        // 4. Menggunakan Service untuk menaikkan gaji (Depedency)
        EmployeeService service = new EmployeeService();
        service.addEmployee(emp1);

        System.out.println("Gaji Budi sebelum naik: Rp" +
emp1.getSalary());
        service.raiseSalary(1, 15); // Naik 15%
        System.out.println("Gaji Budi setelah naik: Rp" +
emp1.getSalary());
    }
}
```

**Screenshot Hasil**



## Penjelasan Permasalahan dan Solusi

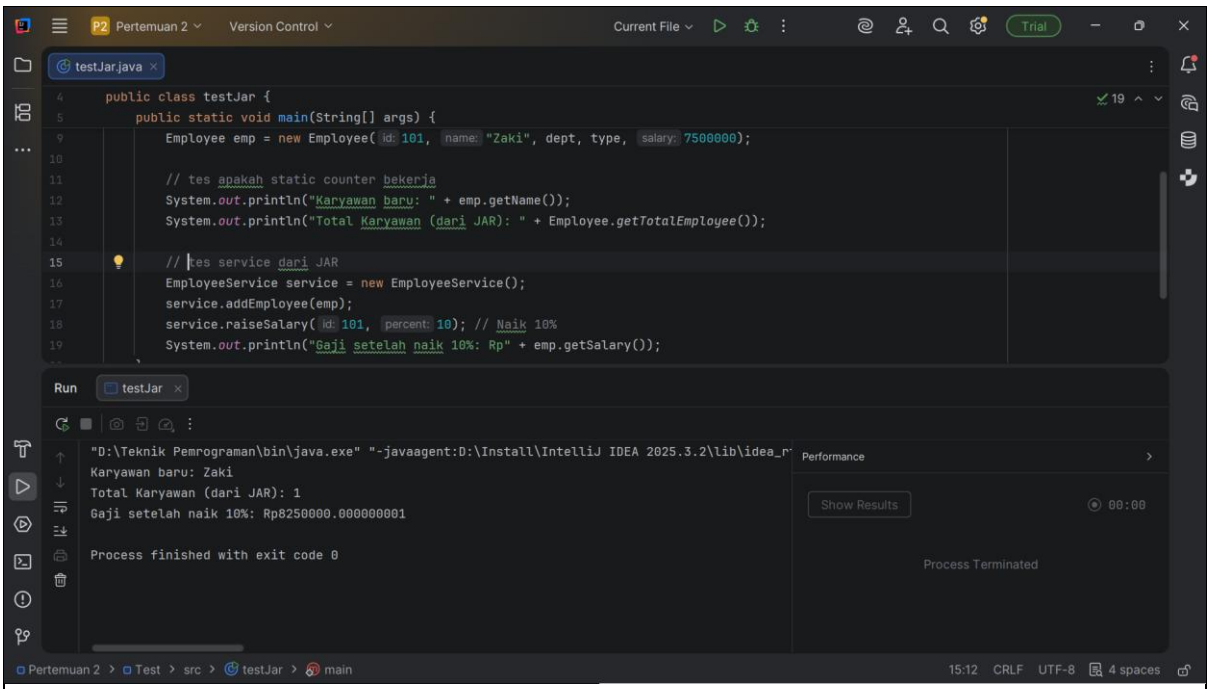
Class diagram:




Perbedaan dan fungsi relasi:

Program ini menerapkan dua jenis relasi utama, yaitu Aggregation dan Dependency untuk mengatur interaksi antar objek secara efisien. Relasi Agregasi direpresentasikan oleh class Employee yang memiliki atribut tipe Department dan EmploymentType, di mana objek-objek tersebut bersifat independen dan tetap dapat eksis di memori meskipun objek Employee dihapus. Sedangkan, relasi Dependensi terjadi pada class EmployeeService yang tidak memiliki objek Employee sebagai atribut permanen, melainkan hanya menggunakannya sebagai parameter sementara dalam *method* addEmployee dan raiseSalary untuk menjalankan logika bisnis tertentu. Oleh karena itu, Agregasi berfungsi sebagai struktur data "memiliki" yang fleksibel, sedangkan Dependensi berfungsi sebagai hubungan "menggunakan" yang hanya muncul saat sebuah fungsi dijalankan.

Membuat project baru untuk mencoba file jar:




| Nama Teman dan Hal yang Dibantu (Opsional) |
| --- |
| Syadida Tsaqifa Nada, Zahra Azkiya |