Week 2 - Object, Class & Encapsulation

Nama : **Alia Ardani**
NIM : **251524035**
Kelas : **1B**
Repo GitHub : **https://github.com/vssixla/Teknik_Pemrograman_2026**

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.

1. Restaurant.java

```
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];
    }
}
```

```
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

Program yang sudah diperbaiki dan ditambahkan fitur *(Mini Case)*.

1. Bagian 1 : Restaurant.java

```
public class Restaurant {

  // 1. Semua attribute harus bersifat private  ==============
  private String[] namaMakanan;
  private double[] hargaMakanan;
  private int[] stok;
  private int id = 0; // tidak perlu static lagi

  public Restaurant() {
    namaMakanan = new String[10];
    hargaMakanan = new double[10];
    stok = new int[10];
  }

  //  2. Akses data menggunakan Getter (Data Access Control)
  public String getNamaMakanan(int index) {
    return namaMakanan[index];
  }

  public double getHargaMakanan(int index) {
```

```
    return hargaMakanan[index];
}

public int getStok(int index) {
    return stok[index];
}

// 2. Setter dengan Validasi
public void setStok(int index, int stokBaru) {
    if (stokBaru >= 0) { // Mencegah stok nilai negatif
        stok[index] = stokBaru;
    } else {
        System.out.println("Stok tidak boleh negatif!");
    }
}

// 3. Validasi Stok Saat Menambahkan Menu
public void tambahMenuMakanan(String nama, double harga, int stokAwal) {
    if (stokAwal < 0) { //Kalau stok < 0 tidak bisa
        System.out.println("Stok tidak boleh negatif!");
        return;
    }

    namaMakanan[id] = nama;
    hargaMakanan[id] = harga;
    stok[id] = stokAwal;
    id++; // otomatis bertambah
}

// Untuk menampilkan daftar menu
public void tampilMenuMakanan() {
    System.out.println("==================== DAFTAR MENU ====================");

    for (int i = 0; i < id; i++) {
        if (!isOutOfStock(i)) {
            System.out.printf(
                "%-3d %-20s [%3d] %15s %,.0f\n",
                i,
                namaMakanan[i],
                stok[i],
                "Rp.",
                hargaMakanan[i]
            );
```

```
        }
      }
    }

    // Untuk Cek stok habis
    private boolean isOutOfStock(int index) {
      return stok[index] == 0;
    }

    // 4. Pengembangan Fitur (Mini Case)
    public void pesanMenu(int index, int jumlah) {
      if (index < 0 || index >= id) { // Validasi index menu
        System.out.println("Menu tidak ditemukan!");
        return;
      }
      if (jumlah <= 0) { // Validasi jumlah pesanan
        System.out.println("Jumlah pesanan tidak valid!");
        return;
      }

      // Validasi ketersediaan stok
      if (stok[index] >= jumlah) {
        stok[index] -= jumlah;  // b) Stok otomatis berkurang
        System.out.println("Pesanan berhasil! Total bayar: Rp. "
            + (hargaMakanan[index] * jumlah));  // Menampilkan total pembayaran
      } else { // c) Pesanan ditolak jika stok tidak cukup
        System.out.println("Pesanan ditolak! Stok tidak mencukupi.");
      }
    }
}
```

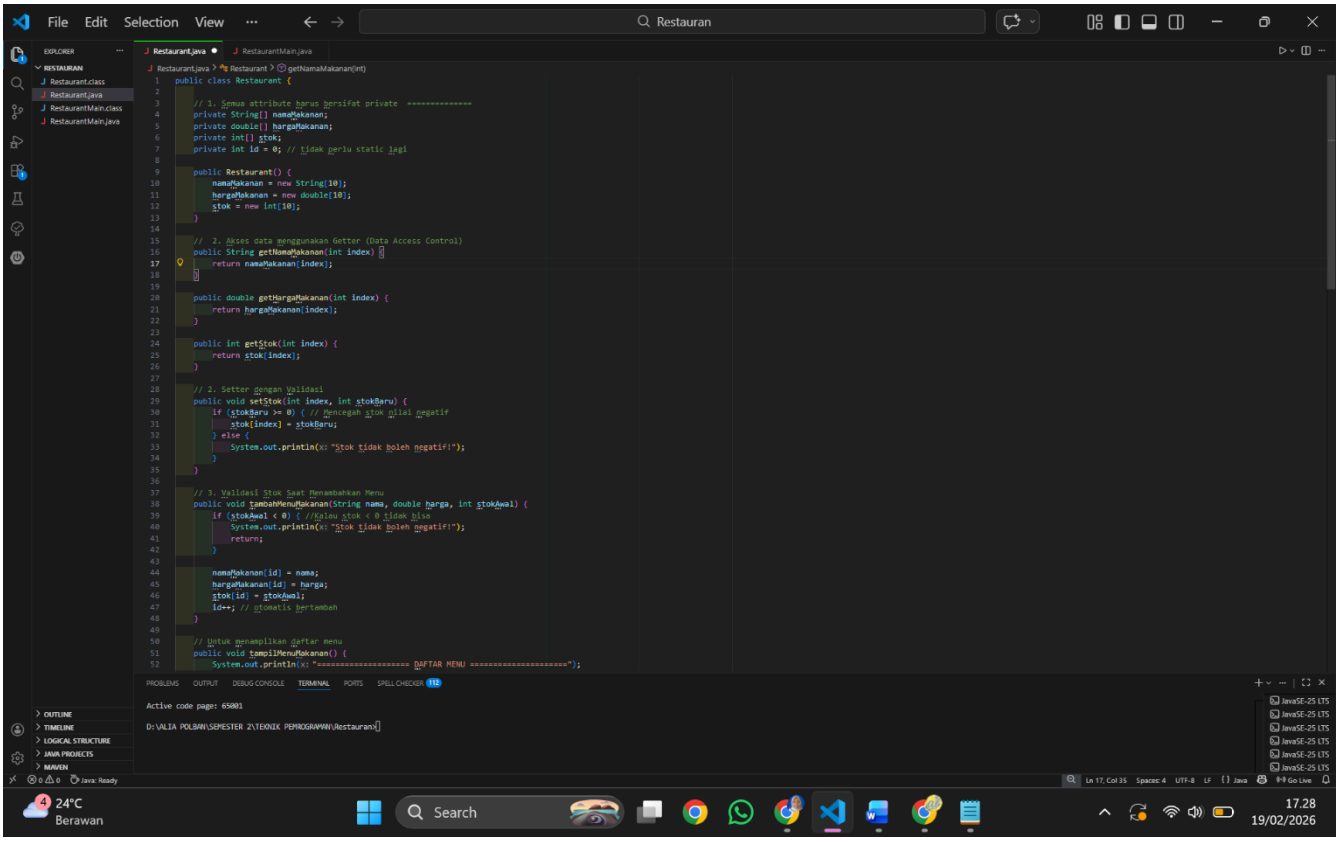
2. Bagian 2 : RestaurantMain.java

```
import java.util.Scanner;

public class RestaurantMain {

  public static void main(String[] args) {

    Scanner input = new Scanner(System.in);
    Restaurant menu = new Restaurant();

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

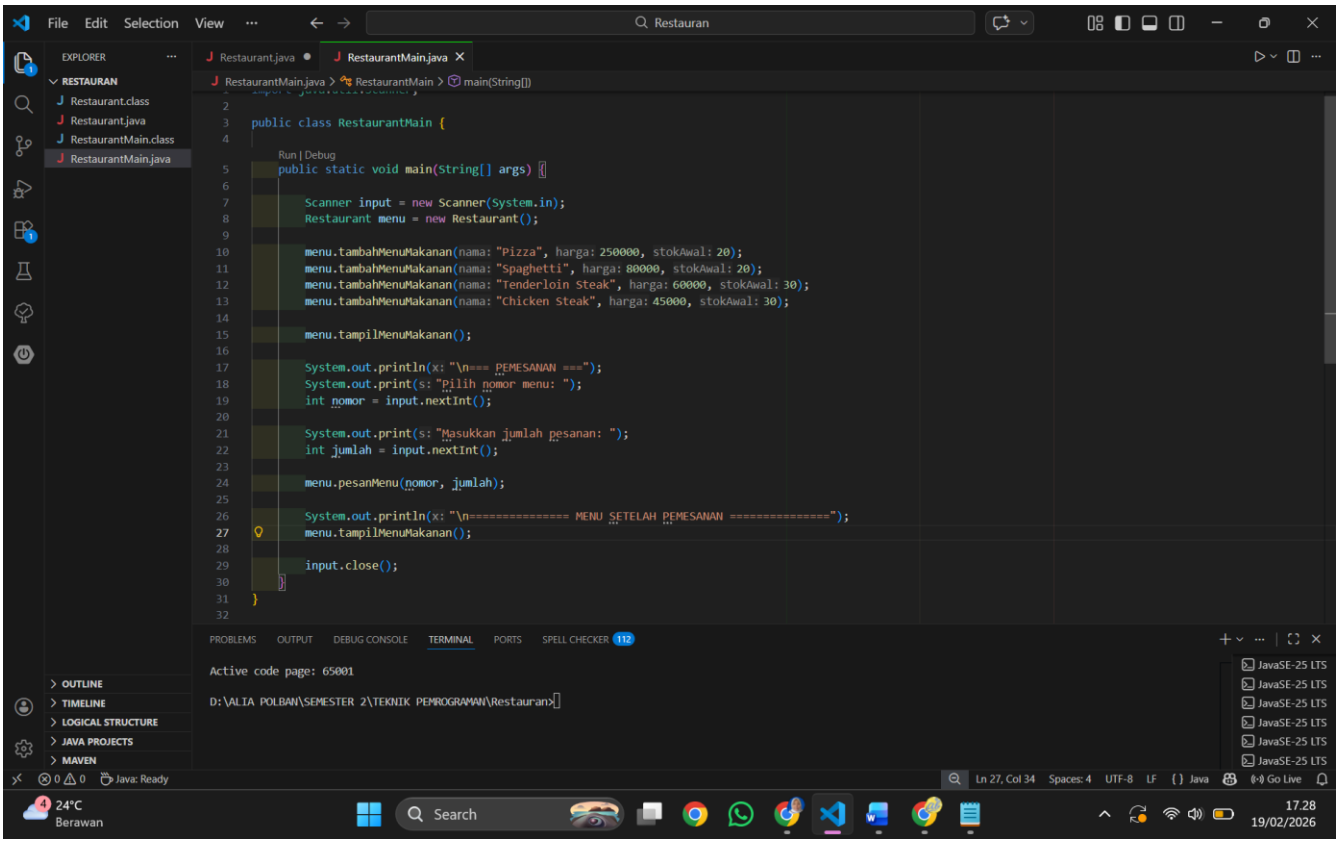
```
        menu.tambahMenuMakanan("Tenderloin Steak", 60000, 30);
        menu.tambahMenuMakanan("Chicken Steak", 45000, 30);

        menu.tampilMenuMakanan();

        System.out.println("\n=== PEMESANAN ===");
        System.out.print("Pilih nomor menu: ");
        int nomor = input.nextInt();

        System.out.print("Masukkan jumlah pesanan: ");
        int jumlah = input.nextInt();

        menu.pesanMenu(nomor, jumlah);

        System.out.println("\n================ MENU SETELAH PEMESANAN ================");
        menu.tampilMenuMakanan();

        input.close();
    }
}
```

Screenshot Hasil

Bagian 1 : Restaurant.java



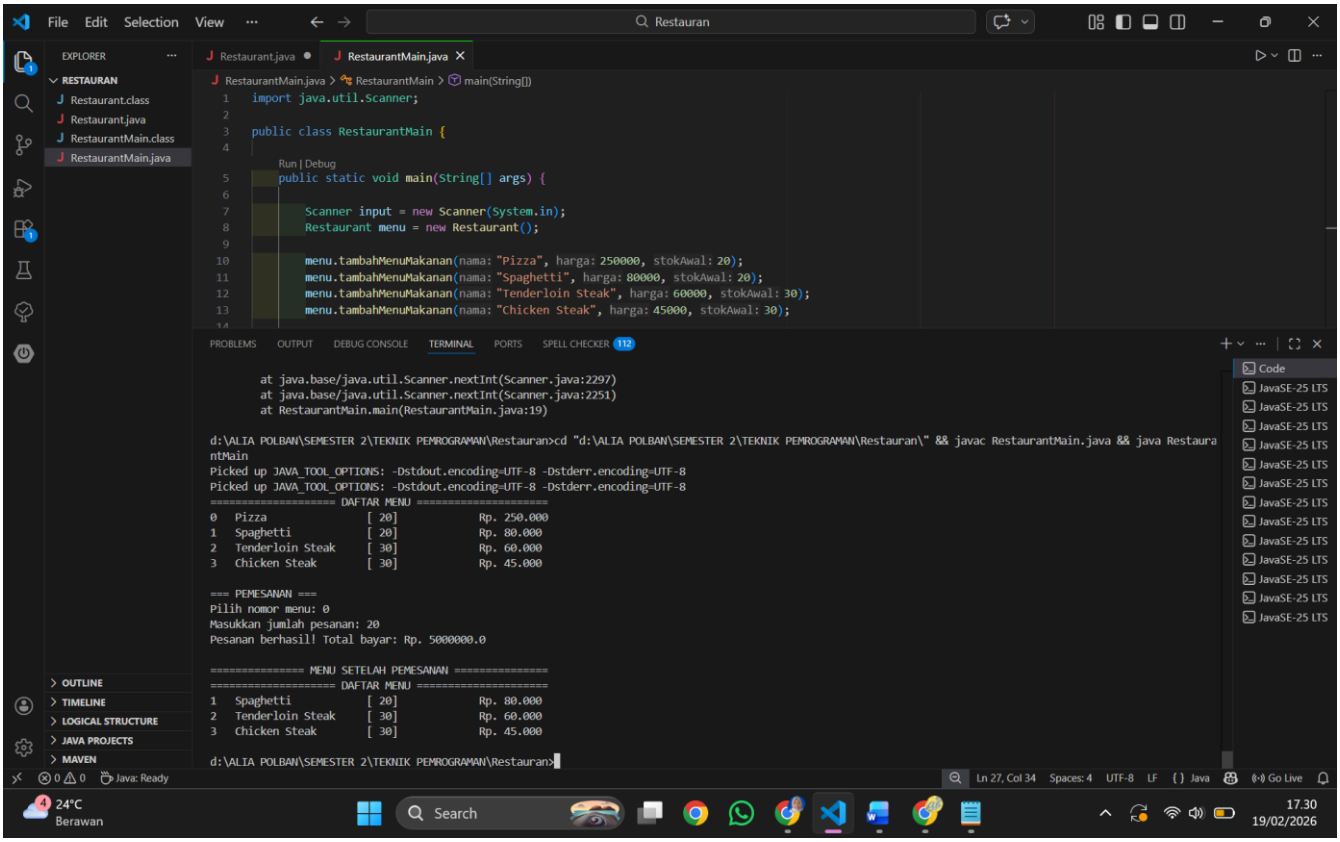
Bagian 2 : RestaurantMain.java



```java
public class RestaurantMain {

    Run | Debug
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        Restaurant menu = new Restaurant();

        menu.tambahMenuMakanan(nama: "Pizza", harga: 250000, stokAwal: 20);
        menu.tambahMenuMakanan(nama: "Spaghetti", harga: 80000, stokAwal: 20);
        menu.tambahMenuMakanan(nama: "Tenderloin Steak", harga: 60000, stokAwal: 30);
        menu.tambahMenuMakanan(nama: "Chicken Steak", harga: 45000, stokAwal: 30);

        menu.tampilMenuMakanan();

        System.out.println(x: "\n=== PEMESANAN ===");
        System.out.print(s: "Pilih nomor menu: ");
        int nomor = input.nextInt();

        System.out.print(s: "Masukkan jumlah pesanan: ");
        int jumlah = input.nextInt();

        menu.pesanMenu(nomor, jumlah);

        System.out.println(x: "\n============== MENU SETELAH PEMESANAN ==============");
        menu.tampilMenuMakanan();

        input.close();
    }
}
```

```
Active code page: 65001

D:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Restauran>
```

Output Program



```
        at java.base/java.util.Scanner.nextInt(Scanner.java:2297)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2251)
        at RestaurantMain.main(RestaurantMain.java:19)

d:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Restauran>cd "d:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Restauran\" && javac RestaurantMain.java && java Restaura
ntMain
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
================== DAFTAR MENU ==================
0   Pizza               [ 20]           Rp. 250.000
1   Spaghetti           [ 20]           Rp. 80.000
2   Tenderloin Steak    [ 30]           Rp. 60.000
3   Chicken Steak       [ 30]           Rp. 45.000

=== PEMESANAN ===
Pilih nomor menu: 0
Masukkan jumlah pesanan: 20
Pesanan berhasil! Total bayar: Rp. 5000000.0

============== MENU SETELAH PEMESANAN ==============
================== DAFTAR MENU ==================
1   Spaghetti           [ 20]           Rp. 80.000
2   Tenderloin Steak    [ 30]           Rp. 60.000
3   Chicken Steak       [ 30]           Rp. 45.000

d:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Restauran>
```

Penjelasan Permasalahan dan Solusi

A. Instruksi – Analisis Desain Class:

1. Apakah class Restaurant sudah menerapkan Encapsulation dengan benar?

   Jawab :

   Class Restaurant belum menerapkan Encapsulation dengan benar.

   Restaurant.java > ...

```
1  public class Restaurant {
2      public String[] nama_makanan;
3      public double[] harga_makanan;
4      public int[] stok;
5      public static byte id = 0;
```

   Hal ini dapat dilihat dari deklarasi atribut pada class Restaurant yang masih menggunakan akses modifier public.

   Sedangkan dalam konsep encapsulation, **atribut atau data dalam sebuah class seharusnya disembunyikan (information hiding) dengan menggunakan akses modifier seperti private.** Tujuannya supaya data **tidak bisa diakses atau diubah secara langsung dari luar class,** melainkan harus melalui method tertentu (getter dan setter) yang dapat mengontrol perubahan data.

   Namun, pada program ini karena atributnya public, maka class lain seperti RestaurantMain dapat mengakses dan mengubah isi array stok, nama_makanan, dan harga_makanan secara langsung.

2. Apakah attribute yang diterapkan saat ini aman dari akses secara langsung?

   Jawab:

   Tidak, attribute yang diterapkan saat ini tidak aman dari akses langsung.

   Karena pada class Restaurant, attribute dibuat dengan modifier public. Akibatnya, class lain seperti RestaurantMain bisa mengakses dan mengubah data secara langsung.

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

   Jawab:

   Iya, terdapat pelanggaran prinsip OOP. Pelanggaran prinsip OOP terdapat pada prinsip Encapsulation.

   Dalam OOP, attribute seharusnya dibuat private agar tidak bisa diakses atau diubah langsung dari luar class. Jika attribute public, maka class lain bisa memodifikasi data secara bebas.

   Pembuktian kesalahan dalam Encapsulation, dengan melakukan perubahan pada RestaurantMain dengan menambahkan beberapa baris kode, maka didapatkan output yang sama sesuai dengan baris di RestaurantMain.

   Penambahan pada baris RestaurantMain :

```
// test akses atribut (untuk cek encapsulation)
menu.stok[0] = -10; // ubah stok secara langsung
menu.nama_makanan[1] = "Bakso"; // ubah nama makanan langsung
menu.harga_makanan[2] = 999999; // ubah harga langsung
```

Output yang dihasilkan :

```
D:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Restauran>java RestaurantMain
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
Pizza[-10]      Rp. 250000.0
Bakso[20]       Rp. 80000.0
Tenderloin Steak[30]    Rp. 999999.0
Chicken Steak[30]       Rp. 45000.0
```

Dari output ini membuktikan bahwa data di dalam class Restaurant dapat dimodifikasi secara langsung dari luar class. Hal ini menunjukkan bahwa class Restaurant tidak memiliki mekanisme perlindungan data.

B. Instruksi - Perbaikan Kode Program

Instruksi Perbaikan Kode Program, telah diterapkan dalam source code yang telah dilampirkan.

Nama Teman Hal yang Dibantu (Opsional)
-

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.

1. *Object*
   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.
2. *Class*
   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.
3. *Message*
   Message menggambarkan proses komunikasi antar object. Dalam Java, message diwujudkan dalam bentuk pemanggilan method pada suatu object untuk meminta object tersebut melakukan suatu aksi.
4. *Encapsulation*
   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

1. Departmen.java

```
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. Employee.java

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

public class Employee {
  private static int totalEmployee = 0; //static field (penggunaan no 1)
  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;
     totalEmployee++; // setiap buat employee, total bertambah (penggunaan no 1)
  }

  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;
  }

  public static int getTotalEmployee() { //(penggunaan no 1)
    return totalEmployee;
  }

}
```

3. EmploymentType.java

```
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. EmployeeService.java

```
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));
    }
  }
}
```

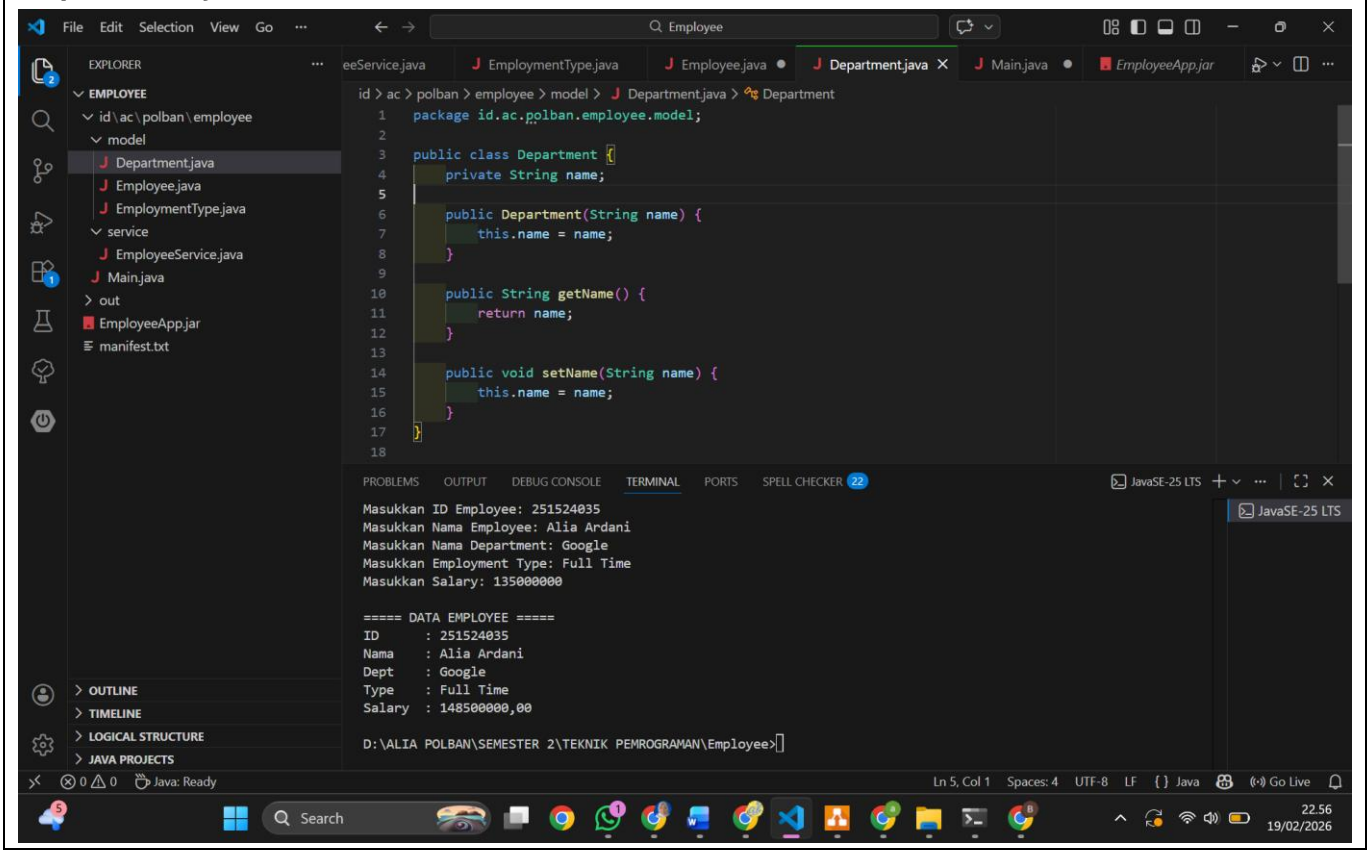
5. Main.java

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

import java.util.Scanner;
import id.ac.polban.employee.model.*;
import id.ac.polban.employee.service.*;

public class Main {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.print("Masukkan ID Employee: ");
    int id = input.nextInt();
    input.nextLine();

    System.out.print("Masukkan Nama Employee: ");
    String nama = input.nextLine();

    System.out.print("Masukkan Nama Department: ");
    String deptName = input.nextLine();

    System.out.print("Masukkan Employment Type: ");
    String typeName = input.nextLine();
```

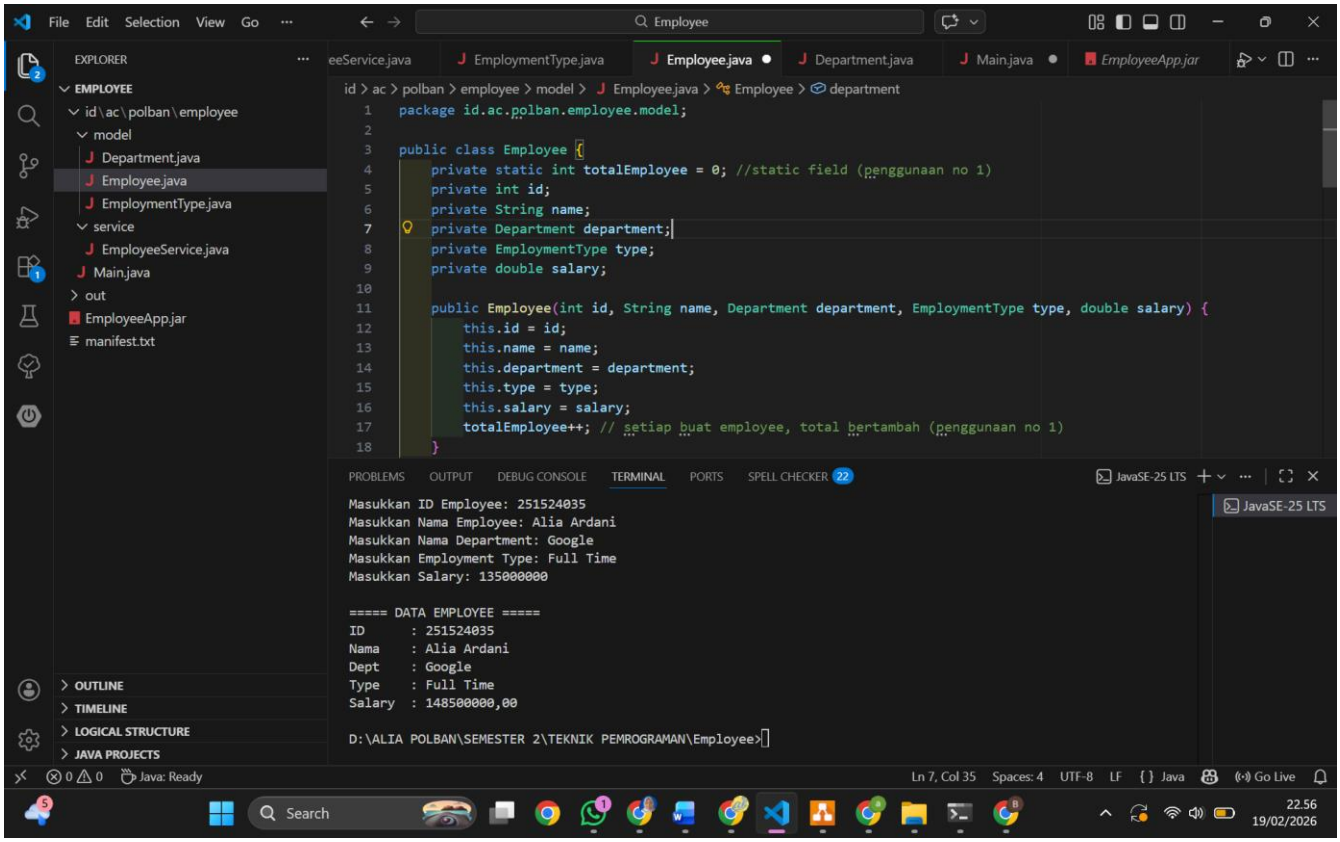
```
        System.out.print("Masukkan Salary: ");
        double salary = input.nextDouble();

        Department dept = new Department(deptName);
        EmploymentType type = new EmploymentType(typeName);
        Employee emp = new Employee(id, nama, dept, type, salary);

        EmployeeService service = new EmployeeService();
        service.addEmployee(emp);
        service.raiseSalary(id, 10);

        System.out.println("\n===== DATA EMPLOYEE =====");
        System.out.println("ID     : " + emp.getId());
        System.out.println("Nama   : " + emp.getName());
        System.out.println("Dept   : " + dept.getName());
        System.out.println("Type   : " + type.getType());
        System.out.printf("Salary  : %.2f\n", emp.getSalary());

        input.close();
    }
}
```

Screenshot Hasil

Department.java



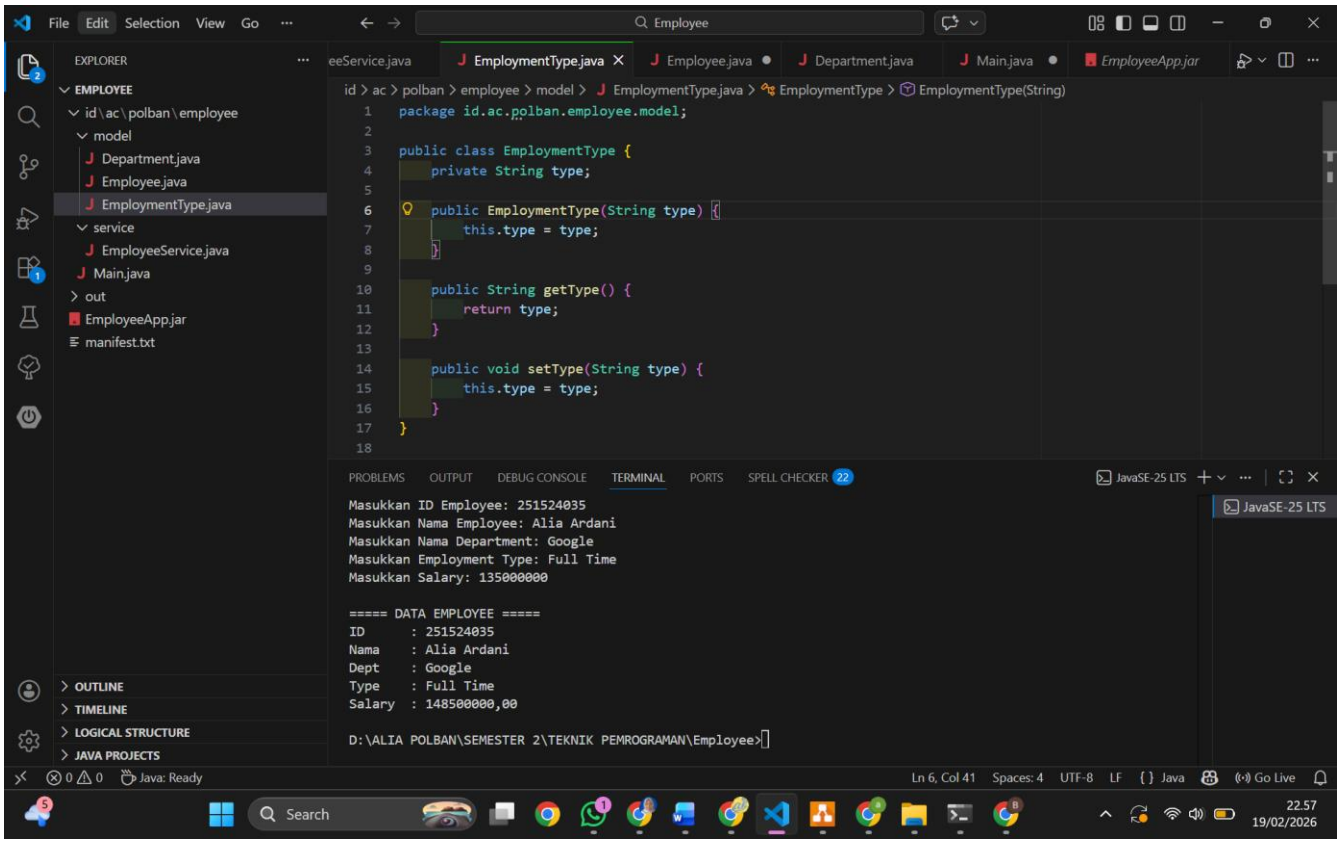
Employee.java



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

public class Employee {
    private static int totalEmployee = 0; //static field (penggunaan no 1)
    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;
        totalEmployee++; // setiap buat employee, total bertambah (penggunaan no 1)
    }
```

```
Masukkan ID Employee: 251524035
Masukkan Nama Employee: Alia Ardani
Masukkan Nama Department: Google
Masukkan Employment Type: Full Time
Masukkan Salary: 135000000

===== DATA EMPLOYEE =====
ID      : 251524035
Nama    : Alia Ardani
Dept    : Google
Type    : Full Time
Salary  : 148500000,00

D:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Employee>
```

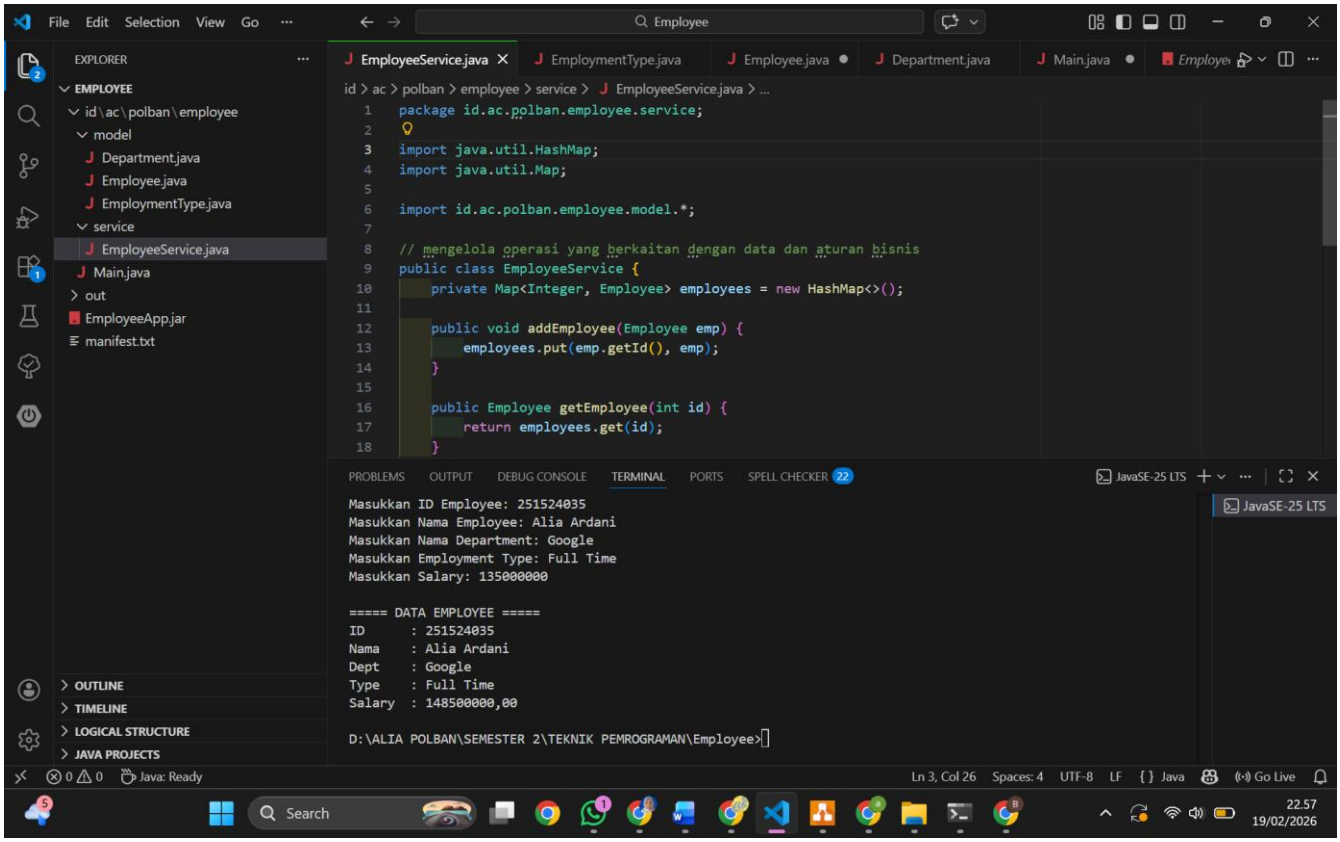
EmploymentType.java



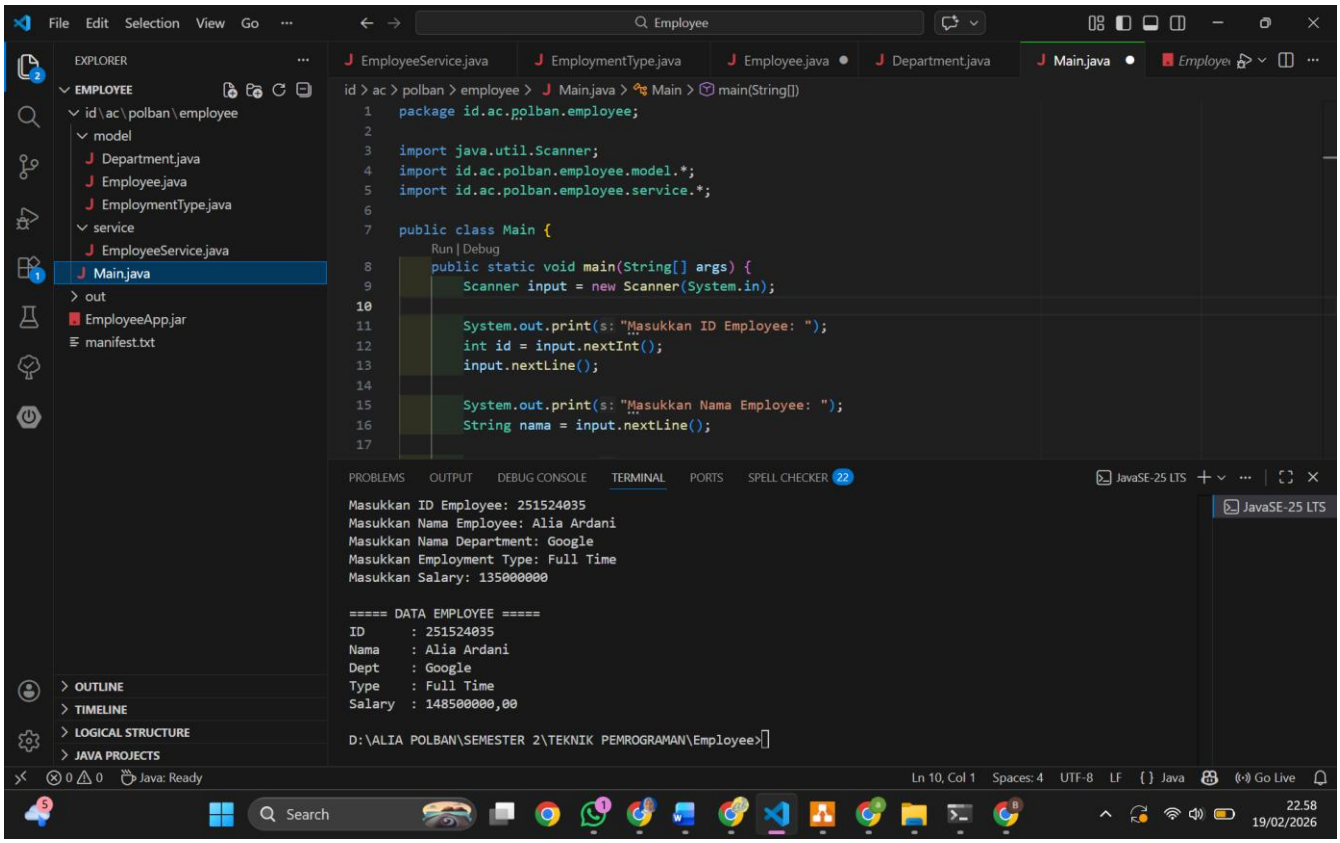
```
package id.ac.polban.employee.model;

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;
    }
}
```

```
Masukkan ID Employee: 251524035
Masukkan Nama Employee: Alia Ardani
Masukkan Nama Department: Google
Masukkan Employment Type: Full Time
Masukkan Salary: 135000000

===== DATA EMPLOYEE =====
ID      : 251524035
Nama    : Alia Ardani
Dept    : Google
Type    : Full Time
Salary  : 148500000,00

D:\ALIA POLBAN\SEMESTER 2\TEKNIK PEMROGRAMAN\Employee>
```

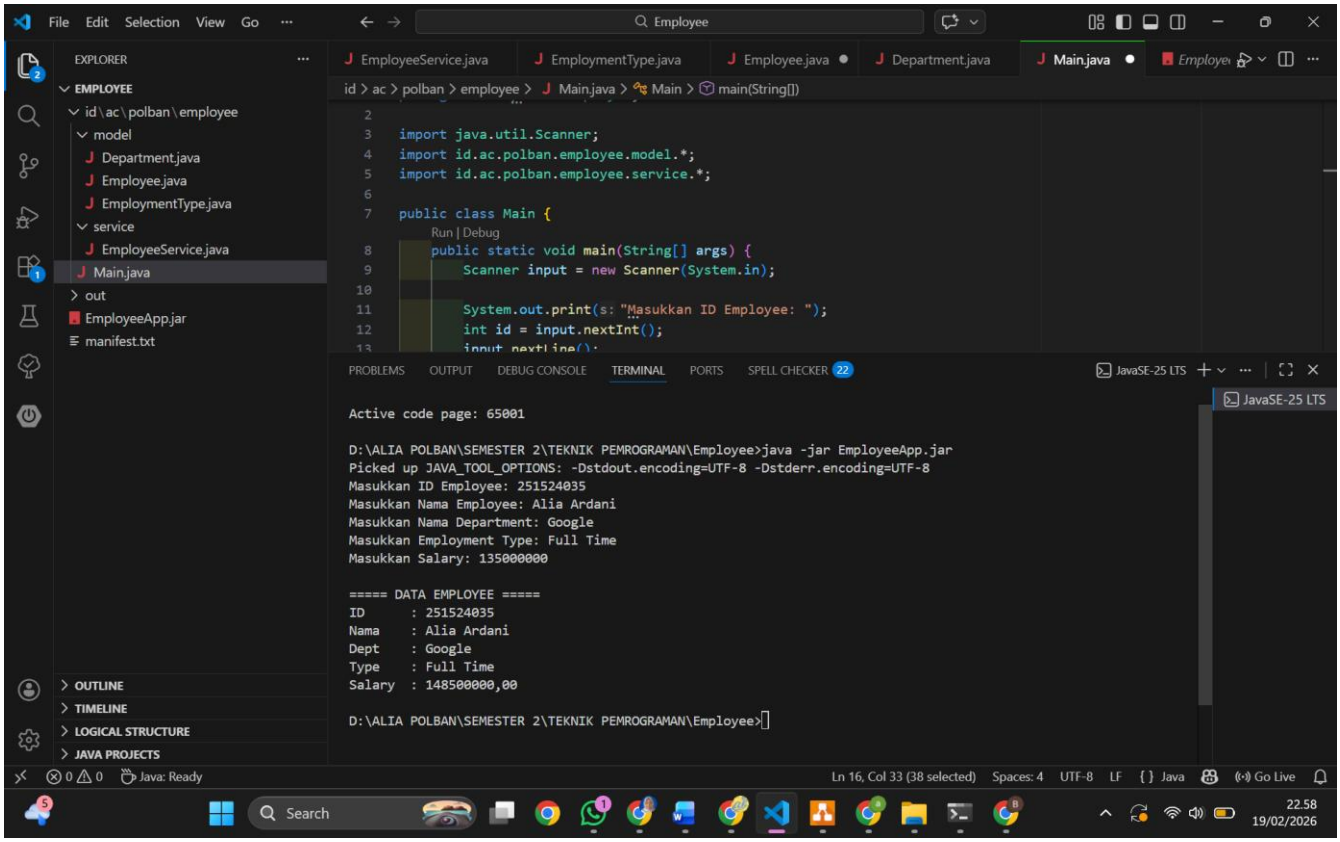
EmployeeService.java



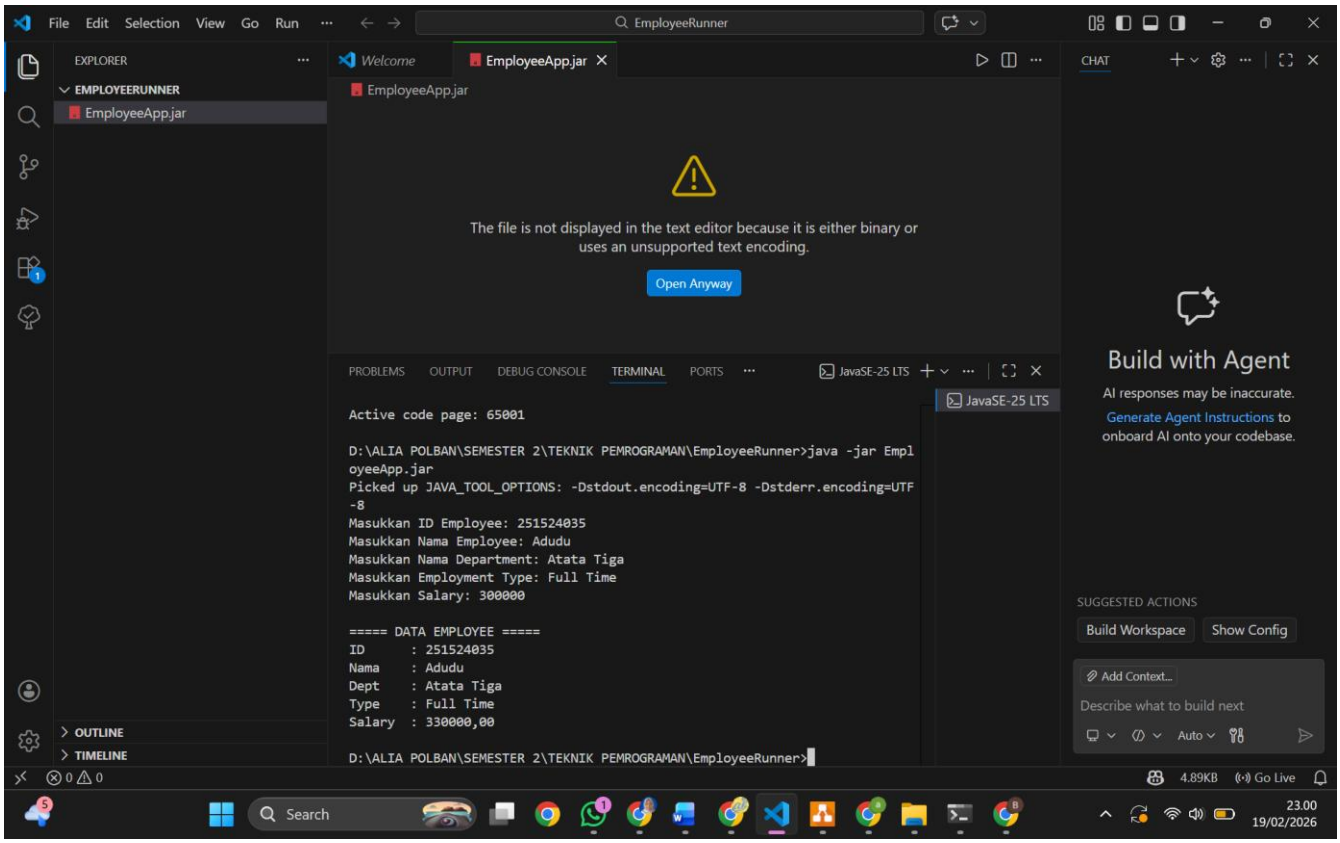
Main.java



Output



File JAR



Penjelasan Permasalahan dan Solusi

A. Instruksi – Tulis Ulang Kode Program Employee

Pada tahap ini dilakukan penulisan ulang kode program Employee dengan terlebih dahulu membuat struktur package id.ac.polban.employee.model dan id.ac.polban.employee.service.

B. Instruksi – Lengkapi Studi Kasus, Rancangan Class Diagram & Penjelasan

1. Lengkapi studi kasus yang telah dibuat dengan menerapkan penggunaan *static field* dan *static method* secara tepat!

   Jawab:

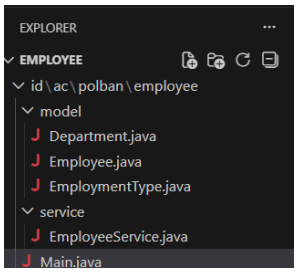
   Penggunaan static field dan static method diterapkan dalam di dalam class Employee, karena informasi yang dikelola bersifat milik bersama seluruh objek Employee, bukan milik salah satu objek

   - Static field digunakan untuk menyimpan jumlah total employee yang dibuat.
   - Static method digunakan untuk mengakses nilai tersebut tanpa perlu membuat object Employee.

2. Terapkan konsep package dengan membuat minimal dua package, yaitu:

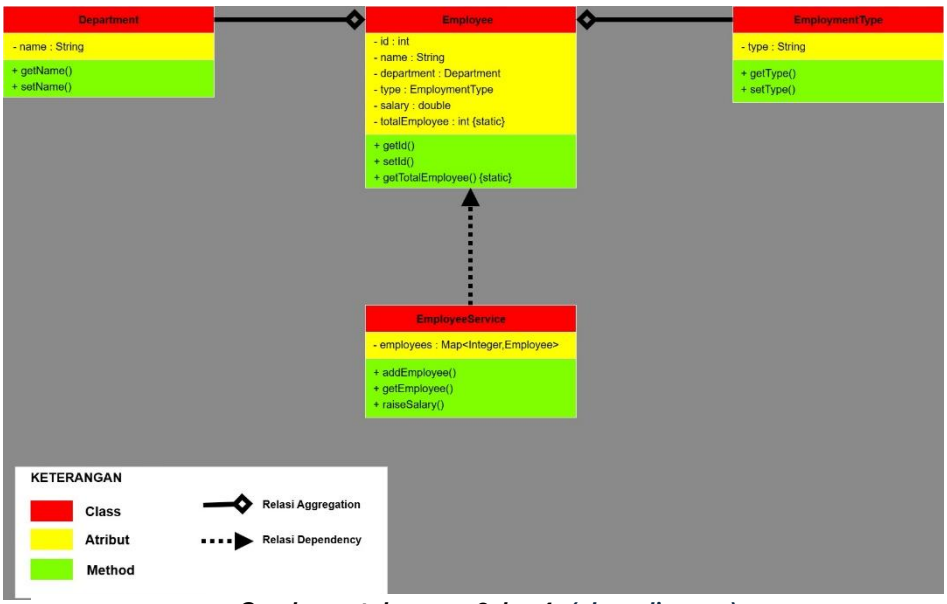
   - `id.ac.polban.employee.model`
   - `id.ac.polban.employee.service`

   Dua package dibuat masing-masing berada di dalam folder employee.



3. Buat diagram kelas (class diagram) yang menggambarkan struktur class, attribute, method, serta hubungan antar class dalam sistem.

4. Implementasikan relasi antar class pada program, yang mencakup:
   - Dependency
   - Aggregation



Gambar untuk nomor 3 dan 4 : (class diagram)

https://drive.google.com/file/d/141ydDiMFEg7oP-DbkQzltoUNXKZEdYwd/view?usp=sharing (Drive Gambar Class Diagram)

5. Jelaskan perbedaan dan fungsi masing-masing jenis relasi tersebut berdasarkan kasus yang dibuat!

   a. Aggregation

   Aggregation adalah hubungan “has-a” (memiliki), di mana suatu class memiliki referensi terhadap class lain, tetapi class tersebut tetap bisa berdiri sendiri.

   Pada kasus ini, relasi aggregation terjadi antara Employee dengan Department dan Employee dengan EmploymentType. Bisa dikatakan bahwa Setiap Employee memiliki Department dan Setiap Employee memiliki EmploymentType. Namun, objek Department dan EmploymentType tetap dapat ada walaupun objek Employee dihapus.

   Fungsi Aggregation ini digunakan untuk menunjukkan bahwa Employee membutuhkan informasi Department dan EmploymentType sebagai bagian dari datanya, tetapi keduanya bukan bagian yang bergantung secara penuh pada Employee.

   b. Dependency

   Dependency adalah hubungan “uses” (menggunakan), di mana suatu class hanya menggunakan class lain untuk menjalankan fungsi tertentu.

   Pada kasus ini, relasi dependency terjadi antara EmployeeService dengan Employee. Bisa dikatakan bahwa EmployeeService menggunakan objek Employee untuk dikelola (ditambahkan, dicari, atau dinaikkan gajinya). Namun, EmployeeService tidak memiliki Employee sebagai bagian permanen dari strukturnya seperti pada aggregation.

   Fungsi Dependency digunakan agar EmployeeService dapat mengelola data Employee tanpa menjadi bagian dari struktur internal Employee itu sendiri.

6. Lakukan proses generate aplikasi ke dalam file JAR!

   Prose generate aplikasi ke dalam file JAR terlebih dahulu mengompilasi seluruh source code Java menjadi file .class menggunakan perintah javac. Setelah itu, membuat file txt yang berisi informasi Main-Class. Selanjutnya semua file hasil kompilasi dikemas menjadi satu file JAR menggunakan perintah jar cfm. File JAR yang dihasilkan kemudian dapat dijalankan secara mandiri menggunakan perintah java -jar.

7. 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!

   File hasil generate dengan nama EmployeeRunner dan dapat dijalankan walaupun tidak ada source code yang ditampilkan.

Nama Teman dan Hal yang Dibantu (Opsional)
-