LAPORAN PRAKTIKUM MINGGU 1

TOPIK

Class dan Object dalam OOP

NAMA : HILDA SAVA ALZENA

NIM : 240202865

KELAS : 3 IKRA

TUJUAN

1. Mahasiswa memahami perbedaan paradigma pemrograman prosedural, OOP, dan fungsional.
2. Mahasiswa mampu membuat program sederhana dengan tiga pendekatan berbeda menggunakan studi kasus Point of Sale (POS).
3. Mahasiswa memahami konsep class dan object dalam OOP serta dapat membuat class Produk.

DASAR TEORI

1. Paradigma Prosedural adalah pendekatan pemrograman yang menekankan urutan langkah atau instruksi untuk menyelesaikan suatu masalah.
2. OOP (Object-Oriented Programming) menggunakan konsep class sebagai blueprint dan object sebagai instansiasi untuk membuat program lebih terstruktur dan modular.
3. Enkapsulasi digunakan untuk menyembunyikan detail data dalam class agar hanya bisa diakses melalui method tertentu.
4. Paradigma Fungsional menekankan pada penggunaan fungsi murni, lambda, dan stream untuk memproses data secara deklaratif.
5. Maintainability dan scalability aplikasi dapat ditingkatkan dengan pemilihan paradigma yang tepat sesuai kompleksitas program.

LANGKAH PRAKTIKUM

1. Setup Project

1. Pastikan sudah menginstall JDK (Java Development Kit), IDE (misal: IntelliJ IDEA, VS Code, NetBeans), Git, PostgreSQL, dan JavaFX di komputer.
2. Buat folder project oop-pos-<nim>.

```
C:\Users\Hilda>D:

D:\>mkdir oop-pos-240202865

D:\>cd oop-pos-240202865
```

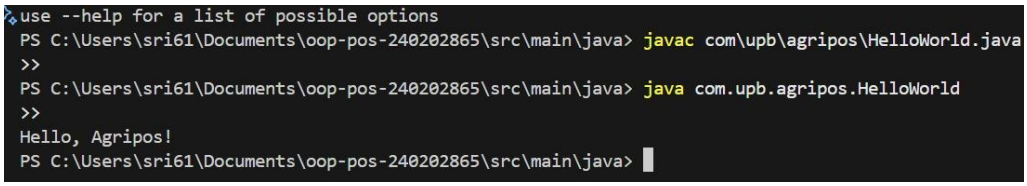
3. Inisialisasi repositori Git.

```
D:\oop-pos-240202865>git init
Initialized empty Git repository in D:/oop-pos-240202865/.git/
```

4. Buat struktur awal src/main/java/com/upb/agripos/.
5. Pastikan semua tools dapat berjalan (uji dengan membuat dan menjalankan program Java sederhana).

- Uji tools dengan membuat program sederhana HelloWorld.java:

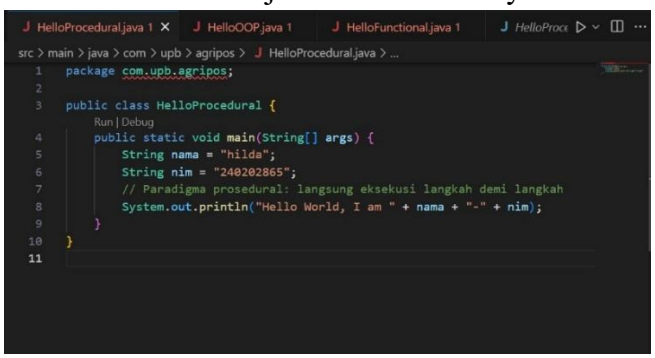
- Jalankan dan pastikan output yang di harapkan keluar



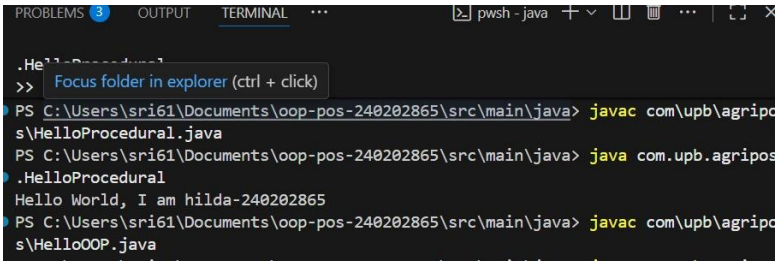
2. Buat Program Sederhana dalam 3 Paradigma

1. Paradigma Prosedural:

- Buat File HelloProcedural.java dan isi kodenya

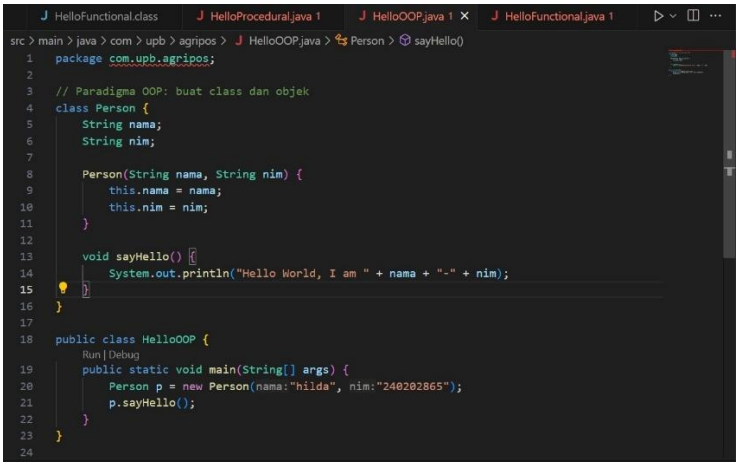


- Jalankan dan akan keluar output nya

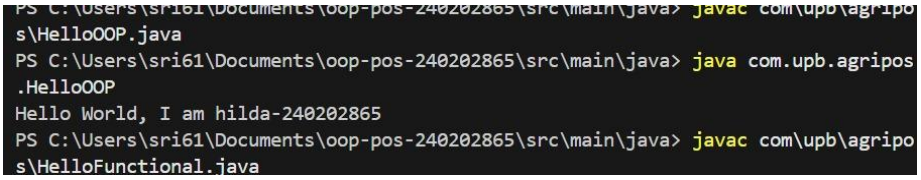


2. **Paradigma OOP:**

- Buat File HelloOOP.java dan isi kodenya



- Jalankan dan akan keluar outputnya



3. **Paradigma Fungsional:**

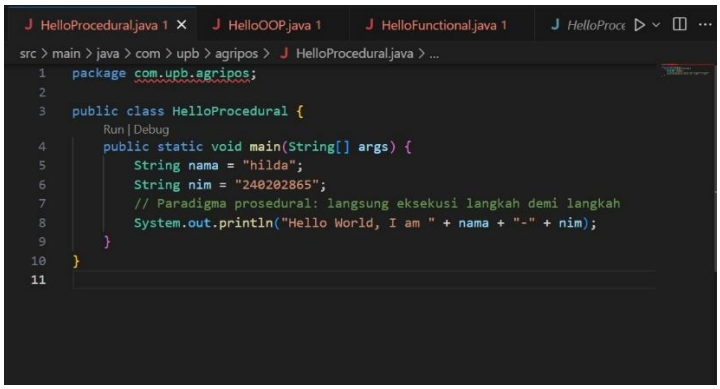
- Buat File HelloFunctional dan isi kodenya

- Jalankan dan akan keluar outputnya

```
PS C:\Users\sri61\Documents\oop-pos-240202865\src\main\java> javac com\upb\agripos\HelloFunctional.java
PS C:\Users\sri61\Documents\oop-pos-240202865\src\main\java> java com.upb.agripos.HelloFunctional
Hello World, I am hilda-240202865
PS C:\Users\sri61\Documents\oop-pos-240202865\src\main\java>
```

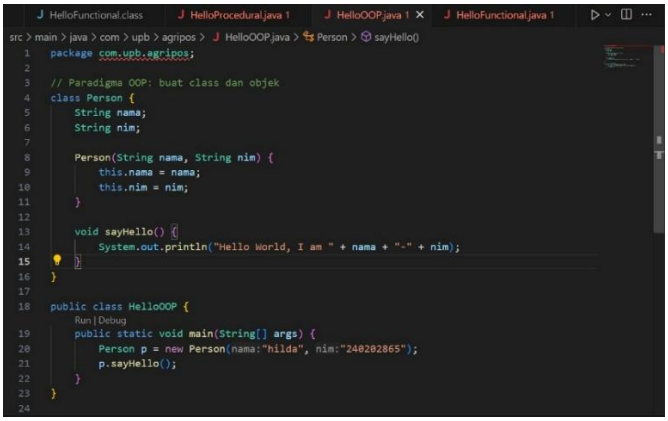
KODE PROGRAM

1. HelloProcedural



```java
package com.upb.agripos;

public class HelloProcedural {
    Run | Debug
    public static void main(String[] args) {
        String nama = "hilda";
        String nim = "240202865";
        // Paradigma prosedural: langsung eksekusi langkah demi langkah
        System.out.println("Hello World, I am " + nama + "-" + nim);
    }
}
```

2. HelloOOP



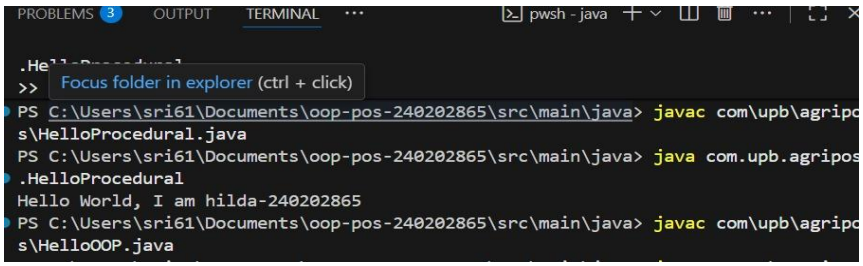
```java
package com.upb.agripos;

// Paradigma OOP: buat class dan objek
class Person {
    String nama;
    String nim;

    Person(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }

    void sayHello() {
        System.out.println("Hello World, I am " + nama + "-" + nim);
    }
}

public class HelloOOP {
    Run | Debug
    public static void main(String[] args) {
        Person p = new Person(nama:"hilda", nim:"240202865");
        p.sayHello();
    }
}
```

3. HelloFunctional

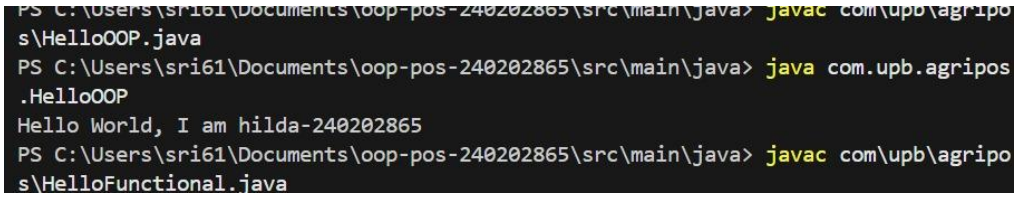
```java
package com.upb.agripos;

import java.util.function.Supplier;

public class HelloFunctional {
    Run | Debug
    public static void main(String[] args) {
        // Paradigma fungsional: gunakan lambda expression
        Supplier<String> hello = () -> "Hello World, I am hilda-240202865";

        System.out.println(hello.get());
    }
}
```

HASIL EKSEKUSI

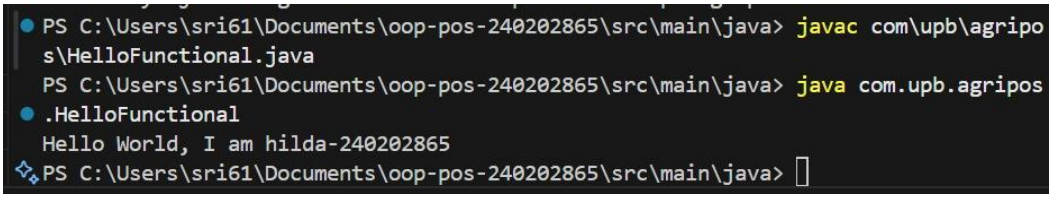
1. HelloProcedural



1. HelloOOP



1. HelloFunctional



ANALISIS

1 .Jelaskan bagaimana kode berjalan:

Paradigma Procedural:

a) Jalannya program dimulai dari main()
   - Di Java, setiap program akan selalu mulai dari method main.

b) Membuat variabel untuk menyimpan data
   - Program menyimpan data nama ("hilda") dan NIM ("240202865") ke dalam variabel bertipe String.

c) Paradigma prosedural
   - Paradigma ini artinya program dieksekusi langkah demi langkah secara urut, persis sesuai barisan kode yang ditulis.
   - Jadi setelah membuat variabel, langsung lanjut ke instruksi berikutnya, tanpa konsep objek atau class tambahan.

d) Mencetak output ke layar
    - Program menggabungkan teks "Hello World, I am " dengan isi variabel nama dan nim.
    - Hasil akhirnya ditampilkan di layar: Hello World, I am hilda-240202865
e) Program selesai
    - Setelah baris terakhir dijalankan, program berhenti karena tidak ada instruksi lain.

Paradigma OOP:

a) Program tetap dimulai dari main()
    - Seperti biasa di Java, method main adalah titik awal eksekusi.
b) Ada class Person
    - Class ini didefinisikan untuk mewakili objek "orang".
    - Di dalamnya ada atribut nama dan nim untuk menyimpan data.
    - Ada juga constructor yang digunakan untuk memberi nilai awal pada atribut saat objek dibuat.
    - Selain itu, ada method sayHello() yang berfungsi mencetak perkenalan menggunakan data dari objek tersebut.
c) Membuat objek Person di main
    - Di dalam main, program membuat sebuah objek baru dari class Person, dengan mengisi nama = "hilda" dan nim = "240202865".
    - Proses ini otomatis menjalankan constructor untuk menyimpan data tersebut ke dalam atribut objek.
d) Memanggil method dari objek
    - Setelah objek dibuat, program memanggil method sayHello() milik objek tersebut.
    - Method ini menggunakan data nama dan nim yang sudah tersimpan di objek, lalu mencetak pesan perkenalan ke layar.
e) Output
    - Hasil akhirnya yang tampil adalah: Hello World, I am hilda-240202865

Paradigma Functional:

a) Program dimulai dari main()
    - Sama seperti sebelumnya, eksekusi dimulai dari method main.
b) Menggunakan paradigma fungsional
    - Paradigma ini berfokus pada fungsi sebagai "nilai" yang bisa disimpan ke variabel atau dikirim sebagai parameter.
    - Di sini digunakan lambda expression, yaitu cara singkat untuk menuliskan sebuah fungsi anonim (fungsi tanpa nama).

c) Membuat fungsi dengan Supplier
    - Program menggunakan Supplier<String> dari Java, yaitu sebuah interface fungsional yang hanya memiliki satu method get().
    - Dengan lambda, dibuat sebuah fungsi yang jika dipanggil akan menghasilkan teks"Hello World, I am hilda-240202865".
    - Fungsi ini tidak dijalankan langsung, tetapi disimpan ke dalam variabel hello.
d) Menjalankan fungsi
    - Untuk menjalankan fungsi yang sudah disimpan tadi, dipanggil method hello.get().
    - Hasilnya adalah string "Hello World, I am hilda-240202865".
e) Output ke layar
    - Nilai yang dihasilkan fungsi kemudian dicetak dengan System.out.println.
    - Output yang muncul di layar adalah: Hello World, I am hilda-240202865

2. Apa perbedaan pendekatan minggu ini dibanding minggu sebelumnya.

- **Minggu sebelumnya** → hanya menyiapkan tools dan memastikan program Java dapat dijalankan.
- **Minggu ini** → mempelajari implementasi tiga paradigma pemrograman dengan studi kasus menghitung total harga produk.
  Dengan begitu, minggu ini lebih menekankan pada konsep pemrograman, bukan sekadar instalasi dan konfigurasi.

3. Kendala yang dihadapi dan cara mengatasinya.

- **Kendala:** Masih bingung membedakan antara pendekatan prosedural, OOP, dan fungsional dalam implementasi kode.
- **Cara mengatasi**: Membaca ulang materi paradigma pemrograman, mencoba menulis ulang kode sederhana dengan tiga pendekatan berbeda, lalu membandingkan hasilnya.

KESIMPULAN

Praktikum minggu ini menunjukkan bahwa penerapan paradigma yang berbeda dapat menghasilkan output yang sama namun dengan cara yang berbeda. Prosedural berfokus pada langkah-langkah instruksi, OOP berfokus pada pemodelan objek, sedangkan fungsional menekankan penggunaan fungsi dan stream. Hal ini membantu saya lebih fleksibel dalam memilih pendekatan pemrograman sesuai kebutuhan sistem.

CHECKLIS KEBERHASILAN

- Lingkungan kerja sudah siap (JDK, IDE, Git, PostgreSQL, JavaFX).
- Repositori Git sudah dibuat dan commit awal berhasil (week1-setup-hello-pos).
- Program "Hello POS World" berjalan di tiga paradigma dan menampilkan NIM serta nama mahasiswa.
- Versi OOP dan fungsional menggunakan minimal tiga objek/entri produk.
- Tangkapan layar hasil eksekusi ketiga program telah disertakan.
- Laporan singkat telah dilampirkan.
- Perbedaan paradigma sudah dipahami dan dijelaskan pada laporan.

QUIZ

1. Apakah OOP selalu lebih baik dari prosedural?

Jawaban: Tidak selalu. OOP lebih baik untuk aplikasi yang kompleks karena memberikan struktur, modularitas, dan kemudahan pengembangan jangka panjang. Namun, untuk program kecil dan sederhana, pendekatan prosedural bisa lebih cepat dan mudah ditulis.

2. Kapan functional programming lebih cocok digunakan dibanding OOP atau prosedural?

Jawaban: Functional programming lebih cocok digunakan ketika bekerja dengan data dalam jumlah besar, pemrosesan paralel, atau operasi koleksi yang berulang. Paradigma ini membuat kode lebih ringkas, deklaratif, dan mudah dibaca saat melakukan transformasi data.

3.Bagaimana paradigma (prosedural, OOP, fungsional) memengaruhi maintainability dan scalability aplikasi?

Jawaban:

- **Prosedural** → mudah dipahami untuk aplikasi kecil, tetapi sulit dipelihara jika program semakin besar karena kode cenderung bercampur.
- **OOP** → lebih maintainable dan scalable karena kode terorganisasi dalam class/objek, sehingga mudah diperluas dan dikelola.
- **Fungsional** → mempermudah pengolahan data kompleks dengan kode yang lebih singkat, sehingga maintainability meningkat, terutama untuk operasi data berulang.

4.Mengapa OOP lebih cocok untuk mengembangkan aplikasi POS dibanding prosedural?

Jawaban: Karena aplikasi POS melibatkan banyak entitas (produk, pelanggan, transaksi, kasir) yang bisa dimodelkan sebagai objek dengan atribut dan perilaku. Dengan OOP, hubungan antar entitas lebih jelas, program lebih modular, mudah dikembangkan, serta sesuai dengan kebutuhan aplikasi bisnis yang kompleks.

5. Bagaimana paradigma fungsional dapat membantu mengurangi kode berulang (boilerplate code)?

Jawaban: Functional programming menggunakan fungsi murni, lambda, dan stream API sehingga banyak operasi dapat ditulis dengan satu baris kode tanpa perlu loop atau variabel tambahan. Hal ini mengurangi boilerplate code dan membuat program lebih ringkas serta mudah dipelihara.