LAPORAN PRAKTIKUM

Object & Class in Java Part II

Diajukan untuk memenuhi salah satu tugas praktikum Mata kuliah Pemrograman Berorientasi Objek (Praktikum)

Rizky Satria Gunawan
241511089

PROGRAM STUDI D3 TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

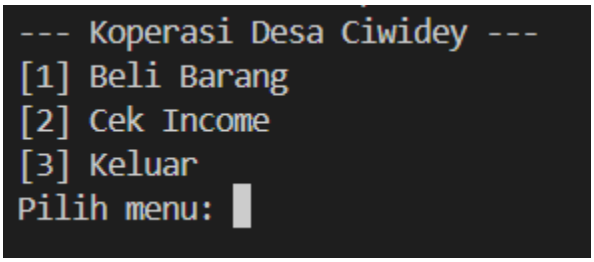
POLITEKNIK NEGERI BANDUNG

2025

1. **Deskripsi Aplikasi**

Aplikasi ini adalah sistem manajemen koperasi sederhana berbasis konsol yang bernama "Koperasi Desa Ciwidey". Aplikasi ini memungkinkan pengguna untuk melakukan pembelian barang yang tersedia dan memeriksa total pendapatan koperasi. Aplikasi ini dibangun dengan menggunakan bahasa pemrograman Java dan menerapkan konsep-konsep dasar Pemrograman Berorientasi Objek (PBO).
Saat dijalankan, aplikasi akan menampilkan menu utama kepada pengguna.

Tampilan Awal / Menu Utama
--- Koperasi Desa Ciwidey ---
[1] Beli Barang
[2] Cek Income
[3] Keluar
Pilih menu:



Fitur-fitur Utama:

- **Beli Barang**: Pengguna dapat melihat daftar barang yang tersedia beserta harganya, kemudian memilih barang dan jumlah yang ingin dibeli. Sistem akan memvalidasi stok, mengurangi stok jika mencukupi, dan mencatat pemasukan ke koperasi.
- **Cek Income**: Fitur ini berfungsi untuk menampilkan total pemasukan atau pendapatan (*income*) yang telah diterima oleh koperasi dari semua transaksi penjualan barang.
- **Keluar**: Opsi untuk menghentikan dan keluar dari aplikasi.

Aplikasi ini dirancang dengan struktur yang memisahkan antara data (model), layanan (logika bisnis), dan tampilan utama, sehingga memudahkan dalam pengembangan dan pemeliharaan.

---

2. **Penerapan static dan package**

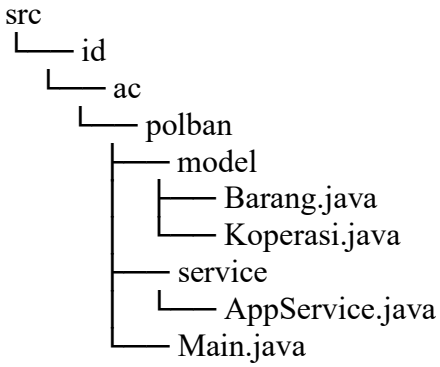
A. **Package**

Proyek ini distrukturkan menggunakan package untuk mengelompokkan kelas-kelas berdasarkan fungsinya, sesuai dengan kaidah PBO untuk meningkatkan *maintainability* dan keterbacaan kode.

- id.ac.polban.model: *Package* ini berisi kelas-kelas yang merepresentasikan entitas atau data dalam aplikasi.
    - Barang.java: Merepresentasikan data barang yang dijual, meliputi nama, harga, dan stok.
    - Koperasi.java: Merepresentasikan koperasi itu sendiri, yang memiliki daftar barang dan catatan pendapatan.

- id.ac.polban.service: *Package* ini berisi kelas yang menyediakan logika bisnis atau layanan aplikasi.
    - AppService.java: Mengatur alur interaksi dengan pengguna, seperti menampilkan menu dan memproses transaksi pembelian.
- id.ac.polban: *Package* ini berisi kelas utama yang menjadi titik masuk (*entry point*) dari aplikasi.
    - Main.java: Kelas yang berisi method main untuk memulai eksekusi aplikasi.

Struktur Folder Proyek:



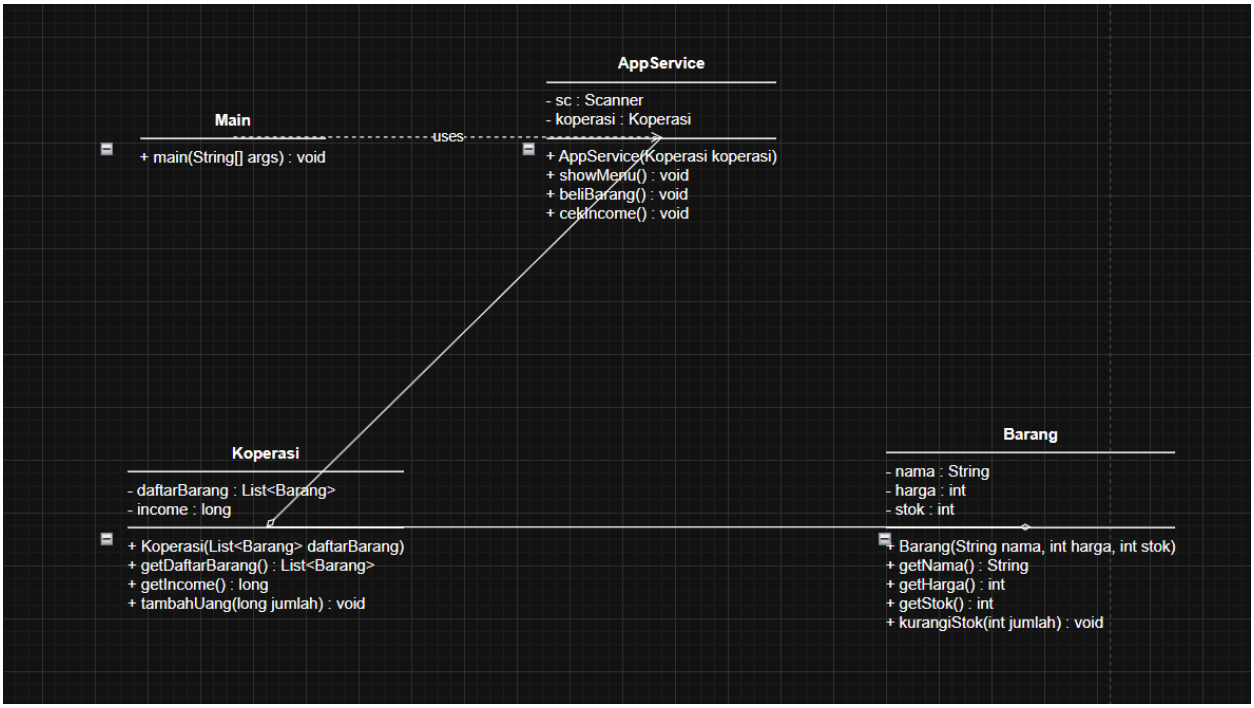
B. Static

Kata kunci static digunakan pada *method* yang menjadi milik kelas itu sendiri, bukan milik *instance* (objek) dari kelas tersebut. Sehingga, *method* tersebut dapat dipanggil tanpa perlu membuat objek dari kelasnya.
Pada proyek ini, static digunakan pada method main di kelas Main.java.
public static void main(String[] args)
Method main dideklarasikan sebagai static karena Java Virtual Machine (JVM) harus dapat memanggilnya untuk memulai eksekusi program tanpa harus membuat objek dari kelas Main terlebih dahulu. Ini adalah titik awal standar untuk semua aplikasi Java.

---

3. Relasi Antar Kelas dan Class Diagram

Diagram kelas di bawah ini menggambarkan struktur dan hubungan antar kelas dalam aplikasi Koperasi Desa Ciwidey.
Class Diagram



A. Dependency (Ketergantungan)

Dependency adalah relasi "uses-a" yang terjadi ketika sebuah kelas menggunakan fungsionalitas dari kelas lain, biasanya sebagai variabel lokal atau parameter dalam sebuah *method*. Kelas yang satu tidak menyimpan *instance* dari kelas yang lain sebagai atribut.

Contoh pada Proyek: Relasi antara kelas Main dan AppService

```
// File: id/ac/polban/Main.java
public class Main {
    public static void main(String[] args) {
        // ... inisialisasi koperasi
        AppService service = new AppService(koperasi); // `Main` membuat dan menggunakan `AppService`

        do {
            service.showMenu(); // `Main` memanggil method dari `AppService`
            // ...
        } while (!choice.equals("3"));
    }
}
```

```
Koperasi koperasi = new Koperasi(daftarBarang);
AppService service = new AppService(koperasi);
```

```
do {
    service.showMenu();
    System.out.print(s:"Pilih menu: ");
```

Dalam *method* main, sebuah objek AppService dibuat dan disimpan dalam variabel lokal service. Kelas Main kemudian menggunakan objek ini untuk memanggil *method* seperti showMenu() dan beliBarang(). Relasi ini bersifat sementara hanya selama eksekusi *method* main. Kelas Main tidak memiliki atribut AppService, sehingga ini merupakan relasi *dependency*.

B. Aggregation (Agregasi)

Aggregation adalah relasi "has-a" di mana sebuah kelas (keseluruhan/*whole*) memiliki atau terdiri dari satu atau lebih *instance* kelas lain (bagian/*part*). Namun, siklus hidup kelas "part" tidak bergantung pada kelas "whole". Jika kelas "whole" dihancurkan, kelas "part" masih bisa tetap ada.
Contoh pada Proyek: Relasi antara kelas Koperasi dan Barang

```
public class Koperasi {
    private List<Barang> daftarBarang; // Aggregation
```

```
public Koperasi(List<Barang> daftarBarang) {
    this.daftarBarang = daftarBarang;
    this.income = 0;
```

```
// File: id/ac/polban/model/Koperasi.java
public class Koperasi {
    private List<Barang> daftarBarang; // Aggregation

    public Koperasi(List<Barang> daftarBarang) {
        this.daftarBarang = daftarBarang;
        //...
    }
}

// File: id/ac/polban/Main.java
public class Main {
    public static void main(String[] args) {
        ArrayList<Barang> daftarBarang = new ArrayList<>(); // Objek 'Barang' dibuat di sini
        daftarBarang.add(new Barang("Pulpen", 2000, 100));
        // ... barang lainnya ditambahkan
```

```
public static void main(String[] args) {
    ArrayList<Barang> daftarBarang = new ArrayList<>();
    daftarBarang.add(new Barang(nama:"Pulpen", harga:2000, stok:100));
    daftarBarang.add(new Barang(nama:"Pensil", harga:2000, stok:150));
```

```
        Koperasi koperasi = new Koperasi(daftarBarang); // 'daftarBarang' dimasukkan ke 'koperasi'
```

```
    }
}
```

Kelas Koperasi memiliki sebuah List<Barang> sebagai atribut. Objek-objek Barang dibuat secara terpisah di dalam kelas Main, kemudian *list* yang berisi objek-objek tersebut dimasukkan ke dalam objek Koperasi melalui konstruktor. Artinya, objek Barang tidak diciptakan oleh Koperasi. Jika objek koperasi dihancurkan, daftarBarang yang dibuat di Main tidak akan ikut hancur. Ini adalah contoh relasi *Aggregation*.

---

4. Membuat dan Menjalankan *JAR File*

Aplikasi Java dapat dikemas ke dalam sebuah *Java Archive* (.jar) *file*. *File* ini dapat berisi semua *file* .class yang telah dikompilasi, *resource*, dan metadata yang diperlukan untuk menjalankan aplikasi, sehingga membuatnya portabel dan mudah didistribusikan.

1. Kompilasi

Langkah pertama adalah mengkompilasi semua *file* .java menjadi *bytecode* (.class). Perintah ini dijalankan dari direktori utama proyek (di atas folder src) dan menempatkan hasilnya di direktori bin.

Bash

```
javac -d bin src/id/ac/polban/model/*.java src/id/ac/polban/service/*.java src/id/ac/polban/*.java
```

2. Pembuatan JAR

Setelah semua kelas dikompilasi, langkah selanjutnya adalah membuat *file* app.jar. Kita memerlukan sebuah *manifest file* (Manifest.txt) untuk menentukan kelas mana yang berisi *method* main sebagai *entry point*.

Isi Manifest.txt:

```
Main-Class: id.ac.polban.Main
```

Gunakan perintah berikut untuk membuat *file* JAR yang dapat dieksekusi (*executable*):

Bash

```
jar cfm app.jar Manifest.txt -C bin .
```

- c: *create a new archive*.
- f: *specify archive file name* (app.jar).
- m: *include manifest information from the given manifest file* (Manifest.txt).
- -C bin .: Mengubah direktori ke bin dan mengambil semua (.) *file* di dalamnya untuk dimasukkan ke JAR.

3. Menjalankan Aplikasi dari JAR

Setelah app.jar berhasil dibuat, aplikasi dapat dijalankan dari terminal manapun menggunakan perintah berikut, tanpa perlu mengakses kode sumbernya lagi.

Bash

```
java -jar app.jar
```

Perintah ini akan dieksekusi oleh JVM, yang akan membaca *manifest* di dalam app.jar untuk menemukan Main-Class dan memulai program.

---

5. *Lesson Learned*

Melalui praktikum ini, didapatkan pemahaman yang lebih dalam mengenai konsep-konsep fundamental dalam Pemrograman Berorientasi Objek dan siklus hidup pengembangan aplikasi Java. [7]

1. **Struktur Proyek dengan package**: Penggunaan package terbukti sangat penting untuk mengorganisir kode ke dalam direktori logis (model, service). Hal ini tidak hanya membuat

proyek lebih rapi, tetapi juga meningkatkan keterbacaan dan mempermudah pengelolaan kode seiring dengan bertambahnya skala aplikasi. [8]

2. **Pemodelan Relasi Antar Kelas**: Pemahaman mengenai relasi antar kelas seperti *Dependency* dan *Aggregation* membantu dalam merancang struktur aplikasi yang logis dan efisien. Kemampuan untuk mengidentifikasi dan mengimplementasikan relasi yang tepat adalah kunci untuk membangun sistem yang kuat dan fleksibel, yang mencerminkan hubungan antar entitas di dunia nyata. [9]
3. **Distribusi Aplikasi dengan JAR**: Proses mengkompilasi, mengemas aplikasi ke dalam *file* JAR, dan menjalankannya memberikan wawasan praktis tentang bagaimana aplikasi Java didistribusikan dan dieksekusi. Ini menyoroti pentingnya

manifest file dalam mendefinisikan *entry point* aplikasi dan menjadikan aplikasi portabel di berbagai lingkungan yang memiliki Java Runtime Environment (JRE). [10101010]