LAPORAN PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
MODUL 4
Object Oriented Programming Java 1

Disusun Oleh:

Gerald Eberhard

(105223002)

PROGRAM STUDI ILMU KOMPUTER

FAKULTAS SAINS DAN KOMPUTER

UNIVERSITAS PERTAMINA

2025

1. Pendahuluan

Studi kasus praktikum ini adalah pembuatan sistem transaksi penjualan sederhana untuk sebuah toko. Sistem ini dirancang untuk mengelola data pelanggan, produk, dan transaksi penjualan menggunakan konsep Pemrograman Berorientasi Objek (PBO) dalam bahasa Java. Setiap transaksi mencakup informasi pelanggan (nama dan ID), produk (nama produk dan kode produk), dan detail transaksi (jumlah pembelian dan total harga). Program ini harus mampu memvalidasi input, memperbarui stok produk setelah transaksi, dan menampilkan laporan transaksi secara terstruktur.

Codingan ini terdiri dari empat kelas yaitu Main.java, Produk.java, Pelanggan.java, dan transaksi.java. Kelas Main.java akan berisi pemanggilan dan pembuatan objek serta pemanggilan perintah-perintah yang terdapat pada kelas produk, pelanggan, dan transaksi. Kemudian ada juga kelas Produk.java yang berisi informasi mengenai method-method dan informasi-informasi yang akan disimpan terkait produk. Kemudian ada juga kelas Pelanggan.java yang berisi informasi mengenai method-method dan informasi yang akan disimpan terkait pelanggan. Kemudian yang terakhir ada kelas transaksi yang bertujuan untuk menyimpan semua informasi terkait transaksi dan operasi-operasi yang akan dilakukan dalam transaksi.

Pembuatan codingan ini dilandaskan berdasarkan materi yang telah dipelajari selama praktikum. Kemudian karena tidak ada detail spesifik tentang harga produk atau aturan bisnis lainnya dalam modul, maka saya akan membuat asumsi sendiri untuk harga produk dan jumlah stok.

2. Variabel

No	Nama Variabel	Tipe Data	Fungsi
1	idProduk	String	Menyimpan ID unik produk dan terdapat di kelas Produk.
2	namaProduk	String	Menyimpan nama produk dan variabel ini terdapat di kelas Produk.
3	harga	double	Menyimpan harga per unit produk dan variabel ini terdapat di kelas Produk.
4	stok	Int	Menyimpan jumlah stok produk dan variabel ini terdapat di kelas Produk.
5	idPelanggan	String	Menyimpan kode identitas pelanggan. Terdapat di dalam kelas Pelanggan.
6	nama	String	Menyimpan nama pelanggan. Terdapat di dalam kelas pelanggan.
7	email	String	Menyimpan alamat email pelanggan. Terdapat di dalam kelas Pelanggan.
8	saldo	double	Menyimpan jumlah saldo yang dimiliki pelanggan dan variabel ini terdapat di kelas Pelanggan.
9	idTransaksi	String	Menyimpan kode identfikasi transaksi yang terdapat di dalam kelas Transaksi
10	pelanggan	Pelanggan (Kelas)	Menyimpan informasi terkait pelanggan yang melakukan transaksi. Variabel ini terdapat di dalam kelas Transaksi
11	Produk	Produk (Kelas)	Menyimpan informasi terkait produk yang dibeli oleh pelanggan. Variabel ini terdapat di dalam kelas Transaksi
12	jumlahBeli	Int	Menyimpan informasi mengenai jumlah barang yang dibeli. Variabel ini terdapat di dalam kelas Transaksi
13	totalHarga	Double	Menyimpan informasi mengenai jumlah total harga yang harus dubayarkan oleh

			pelanggan. Variabel ini terdapat di dalam kelas Transaksi.
14	Produk1	Produk	Ini adalah variabel yang akan menampun nilai object product ke - 1
15	Produk2	Produk	Ini adalah variabel yang akan menampun nilai object product ke - 2
16	pelanggan1	Pelanggan	Ini adalah variabel yang akan menampun nilai object pelanggan ke - 1
17	pelanggan2	Pelanggan	Ini adalah variabel yang akan menampun nilai object pelanggan ke - 2
18	transaksi1	Transaksi	Ini adalah variabel yang akan menampun nilai object transaksi ke - 1
19	transaksi2	Transaksi	Ini adalah variabel yang akan menampun nilai object transaksi ke - 2
20	transaksi3	Transaksi	Ini adalah variabel yang akan menampun nilai object transaksi ke - 3
21	jumlah	int	Menyimpan nilai jumlah barang yang telah dibeli pelanggan terdapat di dalam kelas Produk.
22	jumlah	double	Menyimpan nilai saldo yang ingin dikurangkan dari saldo rekening pelanggan terdapat di dalam kelas Pelanggan.
23	jumlah	double	Menyimpan nilai saldo yang ingin ditambahkan pada saldo rekening pelanggan terdapaat di dalam kelas Pelanggan.

3. Constructor dan Method

Nama Method	Jenis Method	Fungsi
Pelanggan	Constructor	Untuk membuat objek pelanggan
TampilkanInfoPelanggan	Method/Proedur	Untuk menampilkan informasi terkait pelanggan
topUpSaldo	Method/Proedur	Untuk menambahkan saldo pelanggan

kurangiSaldo	Fungsi	Ini adalah fungsi untuk mengurangi saldo pelanggan dan mengembalikan nilai boolean
getSaldo	Fungsi	Ini adalah fungsi untuk mengembalikan nilai saldo pelanggan.
getIdPelanggan	Fungsi	Ini adalah fungsi untuk mengembalikan nilai ID pelanggan.
getNama	Fungsi	Ini adalah fungsi untuk mengembalikan nilai nama pelanggan.
Transaksi	Constructor	Untuk membuat objek transaksi
prosesTransaksi	Fungsi	Ini adalah fungsi untuk melakukan validasi apakah transaksinya dapat dilakukan dengan cara memanggil method kurangiStok dan mengembalikan nilai boolean.
tampilkanDetailTransaksi	Method/Proedur	Method ini bertujuan untuk mengeluarkan informasi detail terkait transaksi. Mirip dengan nota atau struk pembelian sederhana.
Produk	Constructor	Untuk membuat objek produk
tampilkanInfoProduk	Method	Method ini bertujuan untuk menampilkan indormasi produk
kurangiStok	Function	Fungsi ini bertujuan untuk mengurangi jumlah stok barang kemudian mengembalikan nilai boolean
getHarga	Function	Untuk mengembalikan harga produk
getStok	Function	Untuk mengembalikan jumlah stok produk
getKodeProduk	Function	Untuk mengembalikan informasi kode produk
getNama	Function	Untuk mengembalikan informasi nama produk
main	Method	Sebagai tempat dimana objek dibuat, dipanggil dan sebagainya.

4. Dokumentasi dan Pembahasan Code

```
src > Main.java > Language Support for Java(TM) by Red Hat > Main > main(String[])
    Codeium: Refactor | Explain
1   public class Main {
        Run main | Debug main | Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ×
2       public static void main(String[] args) {
3
4           Produk produk1 = new Produk(kodeProduk:"P001", namaProduk:"Laptop", harga:10000000, stok:5);
5           Produk produk2 = new Produk(kodeProduk:"P002", namaProduk:"Mouse", harga:150000, stok:10);
```

Pada awal mula codingan pada kelas main. Akan dibuat objek produk dengan memanfaatkan konstruktor dari kelas produk berikut:

```java
public class Produk {
    private String kodeProduk;
    private String namaProduk;
    private double harga;
    private int stok;

    public Produk(String kodeProduk, String namaProduk, double harga, int stok) {
        this.kodeProduk = kodeProduk;
        this.namaProduk = namaProduk;
        this.harga = harga;
        this.stok = stok;
    }
```

Kemudian setelah itu dengan memanfaatkan method tampilkanInfoProduk pada kelas produk berikut ini:

```java
Codeium: Refactor | Explain | Generate Javadoc | ×
public void tampilkanInfoProduk() {
    System.out.println("Kode Produk: " + kodeProduk);
    System.out.println("Nama Produk: " + namaProduk);
    System.out.println("Harga: Rp " + harga);
    System.out.println("Stok: " + stok);
}
```

Pada kelas main akan dijalankan perintah berikut untuk menampilkan informasi produk pada object produk1 dan produk2 sebagai berikut:

```java
System.out.println(x:"=== Daftar Produk ===");
produk1.tampilkanInfoProduk();
System.out.println();
produk2.tampilkanInfoProduk();
System.out.println();
```

Kemudian selanjutnya kelas main akan memanggil contructor dari kelas pelanggan untuk membuat object pelanggan1 dan pelanggan2 sebagai berikut:

```java
Pelanggan pelanggan1 = new Pelanggan(idPelanggan:"C001", nama:"Gerald", email:"gerald@email.com", saldo:5000000);
Pelanggan pelanggan2 = new Pelanggan(idPelanggan:"C002", nama:"Dion", email:"dion@email.com", saldo:20000000);
```

Hal ini dapat dilakukan dengan memanfaatkan method pelanggan yang telah di buat pada kelas pelanggan sebagai berikut:

```java
Pelanggan.java > Language Support for Java(TM) by Red Hat > Pelanggan > tampilkanInfoPelanggan()
Codeium: Refactor | Explain
public class Pelanggan {
    private String idPelanggan;
    private String nama;
    private String email;
    private double saldo;

    public Pelanggan(String idPelanggan, String nama, String email, double saldo) {
        this.idPelanggan = idPelanggan;
        this.nama = nama;
        this.email = email;
        this.saldo = saldo;
    }
```

Kemudian dengan memanfaatkan method tampilkanInfoPelanggan yang telah dibuat sebagaimana telampir pada gambar di bawah ini maka program main akan memanggil program tersebut untuk menampilkan informasi terkait dengan pelanggan.

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void tampilkanInfoPelanggan() {
    System.out.println("ID Pelanggan: " + idPelanggan);
    System.out.println("Nama: " + nama);
    System.out.println("Email: " + email);
    System.out.println("Saldo: Rp " + saldo);
}
```

Berikut ini pemanggilan method ini yang dilakukan pada kelas main:

```
System.out.println(x:"=== Daftar Pelanggan ===");
pelanggan1.tampilkanInfoPelanggan();
System.out.println();
pelanggan2.tampilkanInfoPelanggan();
System.out.println();
```

Kemudian saya merasa bahwa saldo yang dimiliki oleh objek pelanggan1 sepertinya terlalu sedikit oleh karena itu saya ingin menambahkan jumlah saldonya dengan memanggil method topUpSaldo yang telah saya buat pada kelas pelanggan. Agar dapat melakukan hal ini saya pun menuliskan bari codingan seperti ini pada kelas main:

```
pelanggan1.topUpSaldo(jumlah:5000000);
System.out.println();
```

oia tidak lupa juga saya selalu menggunakan System.out.println() untuk menambahkan bari baru sehingga hasil output pada terminal akan terlihat lebih rapih karena ada jaraknya. Begitu juga saya lakukan pada saat melakukan pemanggilan informasi pada object-object produk dan object-object pelanggan.

Berikut ini adalah method topUpSaldo yang digunakan pada kelas pelanggan:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void topUpSaldo(double jumlah) {
    if (jumlah > 0) {
        saldo += jumlah;
        System.out.println("Top-up berhasil! Saldo baru: Rp " + saldo);
    } else {
        System.out.println(x:"Error: Jumlah top-up harus lebih dari 0!");
    }
}
```

Kemudian program berlanjut, setelah menambahkan saldo pada pelanggan 1 saya ingin memulai transaksi oleh karena itu langkah berikut yang akan saya lakukan adalah saya akan membuat object transaksi dengan menggunakan constructor yang telah dibuat pada kelas transaksi sebagai berikut:

```
Transaksi transaksi1 = new Transaksi(idTransaksi:"T001", pelanggan1, produk1, jumlahBeli:1);
Transaksi transaksi2 = new Transaksi(idTransaksi:"T002", pelanggan2, produk2, jumlahBeli:3);
Transaksi transaksi3 = new Transaksi(idTransaksi:"T003", pelanggan1, produk1, jumlahBeli:10);
Transaksi transaksi4 = new Transaksi(idTransaksi:"T004", pelanggan1, produk2, jumlahBeli:5);
```

Method Transaksi yang telah saya buat pada kelas Transaksi adalah sebagai berikut:

```
Transaksi.java > Language Support for Java(TM) by Red Hat > Transaksi > prosesTransaksi()
Codeium: Refactor | Explain
public class Transaksi {
    private String idTransaksi;
    private Pelanggan pelanggan;
    private Produk produk;
    private int jumlahBeli;
    private double totalHarga;

    public Transaksi(String idTransaksi, Pelanggan pelanggan, Produk produk, int jumlahBeli) {
        this.idTransaksi = idTransaksi;
        this.pelanggan = pelanggan;
        this.produk = produk;
        this.jumlahBeli = jumlahBeli;
        this.totalHarga = produk.getHarga() * jumlahBeli;
    }
```

```
public class Pelanggan
extends Object

Pelanggan
```

Codingan pun berlanjut kembali pada kelas main, saya akan melanjutkan proses transaksi. Berbagai method dari kelas transaksi pun dipanggil di dalam kelas main sebagai berikut:

```
System.out.println(x:"=== Proses Transaksi ===");
if (transaksi1.prosesTransaksi()) {
    transaksi1.tampilkanDetailTransaksi();
}
System.out.println();

if (transaksi2.prosesTransaksi()) {
    transaksi2.tampilkanDetailTransaksi();
}
System.out.println();

if (transaksi3.prosesTransaksi()) {
    transaksi3.tampilkanDetailTransaksi();
}
System.out.println();

if (transaksi4.prosesTransaksi()) {
    transaksi4.tampilkanDetailTransaksi();
}
System.out.println();
```

Object transaksi1, transaksi2, transaksi3, dan transaksi4 pun akan dibuat. Disini untuk melakukan proses transaksi pada setiap object setiap object akan ditransaksikan di dalam method prosesTransaksi sebagai berikut:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public boolean prosesTransaksi() {
    if (!produk.kurangiStok(jumlahBeli)) {
        return false;
    }

    if (!pelanggan.kurangiSaldo(totalHarga)) {
        produk.kurangiStok(-jumlahBeli);
        return false;
    }

    System.out.println(x:"Transaksi berhasil!");
    return true;
}
```

Method proses transaksi ini sendiri merupakan fungsi yang akan melakukan operasi operasi mengurangi stok barang karena barang tersebut akan diambil atau istilahnya sebelum barang ini diserahkan ke penjual barang itu pasti akan dikeluarkan dari inventory terlebih dahulu. Misalnya ada pembeli datang mau beli pistol maka pasti penjual akan mengeluarkan beberapa koleksi pistol dari displaynya. Nah inilah yang saya asumsikan sehingga akan dijalan produk.kurangiStok. Namun disini saya menggunakan tanda seru hal ini berhubungan dengan konsep dari method kurangiStok yang telah saya buat pada kelas produk sebagai berikut:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public boolean kurangiStok(int jumlah) {
    if (jumlah <= stok) {
        stok -= jumlah;
        return true;
    } else {
        System.out.println("Error: Stok tidak mencukupi! Stok tersedia: " + stok);
        return false;
    }
}
```

Okay demikianlah logic pada method kurangi stok kemudian otomatis apabila barang tersebut masih tersedia maka program pada struktur if ajan dijalankan dan otomatis stok akan dikurangi dengan jumlah barang yang dibeli.Namun, apabila jumlah stok tidak mencukupi maka akan muncur pesan eror stok tidak mencukupi.

Jika demikian kita akan kembali lagi ke method prosesTransaksi pada kelas Transaksi Berikut:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public boolean prosesTransaksi() {
    if (!produk.kurangiStok(jumlahBeli)) {
        return false;
    }

    if (!pelanggan.kurangiSaldo(totalHarga)) {
        produk.kurangiStok(-jumlahBeli);
        return false;
    }

    System.out.println(x:"Transaksi berhasil!");
    return true;
}
```

Nah jangan bingung pada bagian ini ya kok objek yang di validasi adalah object pelanggan tapi kok ketika validasinya bernilai false program malah mengurangi stok. Nah teman-teman ingat bahwa tadi kan pistolnya sudah dikeluarkan untuk ditunjukkan kepada pembeli. Namun apa yang harus dilakukan kalo seandainya pembeli tidak jadi membeli produk kita? Yap benar produk tersebut akan disimpan kembali. Karena sebelumnya logik dari kurangiStok adalah stok-jumlah. Maka sekarang agar mengembalikan jumlah stok ke jumlah semula maka kita harus menambahkan stok dengan jumlah. Hal ini dapat dilakukan dengan mengubah jumlah menjadi minus sehingga operasinya menjadi stok – (- jumlah). Oke kalau hal ini sudah jelas. Saya akan menjelaskan bagian fungsi untuk validasi yaitu fungsi kurangiSaldo dari kelas pelanggan. Programnya itu sebagai berikut:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public boolean kurangiSaldo(double jumlah) {
    if (jumlah <= saldo) {
        saldo -= jumlah;
        return true;
    } else {
        System.out.println("Error: Saldo tidak mencukupi! Saldo tersedia: Rp " + saldo);
        return false;
    }
}
```

Mirip seperti fungsi mengurangi stok demikian juga apabila barang tersebut terjual maka program akan mengurangi jumlah saldo yang dimiliki oleh object pelanggan. Tapi bagaimana jika saldo yang dimiliki oleh pelanggan tidak cukup. Maka akan muncul pesan eror yang menandakan saldonya tidak cukup.

Kemudian program akan berlanjut ke bagian berikut :

```
Codeium: Refactor | Explain | Generate Javadoc | X
public boolean prosesTransaksi() {
    if (!produk.kurangiStok(jumlahBeli)) {
        return false;
    }

    if (!pelanggan.kurangiSaldo(totalHarga)) {
        produk.kurangiStok(-jumlahBeli);
        return false;
    }

    System.out.println(x:"Transaksi berhasil!");
    return true;
}
```

Apabila semuanya lancar-lancar saja maka transaksi berhasil. Akan ada mengeluarkan output string menyatakan bahwa “Transaksi Berhasil”. Kemudian kita akan kembali lagi ke kelas main.

```
System.out.println(x:"=== Proses Transaksi ===");
if (transaksi1.prosesTransaksi()) {
    transaksi1.tampilkanDetailTransaksi();
}
System.out.println();

if (transaksi2.prosesTransaksi()) {
    transaksi2.tampilkanDetailTransaksi();
}
System.out.println();

if (transaksi3.prosesTransaksi()) {
    transaksi3.tampilkanDetailTransaksi();
}
System.out.println();

if (transaksi4.prosesTransaksi()) {
    transaksi4.tampilkanDetailTransaksi();
}
System.out.println();
```

Apabila proses transaksi berhasil maka dari konsep if yang ada di atas. Maka nanti akan ada dijalankan method tampilkanDetailTransaksi. Method ini terdapat pada kelas Transaksi isinya juga hanya method untuk menampilkan output. Berikut codingannya:

```
Codeium: Refactor | Explain | Generate Javadoc | X
public void tampilkanDetailTransaksi() {
    System.out.println(x:"=== Detail Transaksi ===");
    System.out.println("ID Transaksi: " + idTransaksi);
    System.out.println("Pelanggan: " + pelanggan.getNama() + " (ID: " + pelanggan.getIdPelanggan() + ")");
    System.out.println("Produk: " + produk.getNamaProduk() + " (Kode: " + produk.getKodeProduk() + ")");
    System.out.println("Jumlah Beli: " + jumlahBeli);
    System.out.println("Total Harga: Rp " + totalHarga);
}
```

Dalam melakukan proses print nota atau informasi transaksi atau yang juga bisa disebut struk transaksi ini ada digunakan beberapa method yang berfungsi untuk melakukan return nilai dari kelas produk dan pelanggan sebagai berikut:

Dari kelas produk sebagai berikut:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public double getHarga() {
    return harga;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public int getStok() {
    return stok;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public String getKodeProduk() {
    return kodeProduk;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public String getNamaProduk() {
    return namaProduk;
}
```

Dari kelas Pelanggan sebagai berikut:

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public double getSaldo() {
    return saldo;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public String getIdPelanggan() {
    return idPelanggan;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public String getNama() {
    return nama;
}
```

Kemudian program di kelas main akan berlanjut dengan line code berikut:

```
System.out.println(x:"=== Informasi Terbaru ===");
System.out.println(x:"Produk setelah transaksi:");
produk1.tampilkanInfoProduk();
System.out.println();
produk2.tampilkanInfoProduk();
System.out.println();
```

Disini informasi terkait produk akan ditampilkan lagi dengan menggunakan method tampilkanInfoProduk. Line Codingan juga akan berlanjut dengan menampilkan status pelanggan setelah melakukan transaksi sebagai berkut:

```
System.out.println(x:"Pelanggan setelah transaksi:");
pelanggan1.tampilkanInfoPelanggan();
System.out.println();
pelanggan2.tampilkanInfoPelanggan();
```

Demikian penjelasan codingan saya. Sekian dan terimakasih

5. Kesimpulan

Praktikum ini berhasil mengimplementasikan sistem transaksi penjualan sederhana dengan konsep PBO seperti *encapsulation* (melalui atribut private dan getter/setter) dan hubungan antar kelas. Program memenuhi kebutuhan studi kasus dengan mengelola data produk, pelanggan, dan transaksi secara terstruktur. Pengalaman ini sangat membantu untuk memperdalam pemahaman saya tentang penerapan materi PBO dalam menyelesaikan masalah nyata. Sekian dan terima kasih.

6. Daftar Pustaka

Modul 4: Obejct Oriented Programming Java 1