WORKSHOP REKAYASA ULANG KODE
Studi Kasus Code Smell
Dosen Pengampu : Adam Shidqul Aziz, S.ST, M.T

Disusun oleh :
Fatimah Az-zahra' Mahros
3122600002
3 D4 Teknik Informatika A

PROGRAM STUDI TEKNIK INFORMATIKA
DEPARTEMEN TEKNIK INFORMATIKA DAN KOMPUTER
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

TAHUN 2024

Studi Kasus #1

OrderProcessor.java

```
import java.util.List;

public class OrderProcessor {
    private String orderId;
    private String customerName;
    private String customerPhone;
    private String customerAddress;
    private List<String> menuItems;
    private double totalPrice;
    private double tax;
    private String paymentMethod;
    private String discountCode;

    // Method untuk memproses pesanan
    public void processOrder(String orderId, String customerName, String customerPhone, String
customerAddress, List<String> menuItems, double totalPrice, double tax, String paymentMethod,
String discountCode) {
        // Proses validasi pesanan
        if (orderId == null || customerName == null || customerPhone == null || menuItems ==
null || totalPrice <= 0) {
            System.out.println("Pesanan tidak valid");
            return;
        }

        // Kalkulasi total harga dan pajak
        double finalPrice = totalPrice + tax;
        if (discountCode != null && !discountCode.isEmpty()) {
            finalPrice -= applyDiscount(discountCode, totalPrice);
        }

        // Menyimpan pesanan
        saveOrder(orderId, customerName, customerPhone, customerAddress, menuItems,
finalPrice, paymentMethod);

        // Mengirimkan notifikasi
        sendNotification(customerPhone, customerName, "Pesanan Anda telah diproses dengan
total: " + finalPrice);
    }

    private double applyDiscount(String discountCode, double totalPrice) {

        if (discountCode.equals("DISC10")) {
            return totalPrice * 0.10;
        } else if (discountCode.equals("DISC20")) {
            return totalPrice * 0.20;
        } else {
            return 0;
        }
    }

    private void saveOrder(String orderId, String customerName, String customerPhone, String
customerAddress, List<String> menuItems, double finalPrice, String paymentMethod) {

        System.out.println("Pesanan disimpan ke database: " + orderId);
    }

    private void sendNotification(String phoneNumber, String customerName, String message) {

        System.out.println("Mengirim pesan ke " + phoneNumber + ": " + message);
    }
}
```

Code Smells

1. Comments

```
        // Method untuk memproses pesanan
        // Proses validasi pesanan
        // Kalkulasi total harga dan pajak
        // Menyimpan pesanan
        // Mengirimkan notifikasi
```

OO Principle yang dilanggar

Melanggar OO Principle YAGNI, dimana terdapat comment yang tidak berguna dan tidak memberikan tambahan bernilai terhadap kode yang ada.

Penyebab

Adanya comment yang menjelaskan kegunaan method dan baris kode yang sederhana, dimana tanpa adanya comment tersebut, kegunaan kode bisa dipahami dari penamaan method dan atribut yang digunakan.

Solusi

Menghapus comment yang tidak diperlukan, dimana kode tersebut tidak perlu dijelaskan dengan comment apapun karena sudah *self-explanatory* berdasarkan penamaannya.

2. Dead Code

```
private String orderId;
    private String customerName;
    private String customerPhone;
    private String customerAddress;
    private List<String> menuItems;
    private double totalPrice;
    private double tax;
    private String paymentMethod;
    private String discountCode;
```

OO Principle yang dilanggar

Melanggar OO Principle YAGNI, dimana terdapat atribut yang tidak pernah dipakai sehingga tidak dibutuhkan.

Penyebab

Adanya atribut yang tidak pernah digunakan pada method manapun.

Solusi

Menghapus atribut yang tidak diperlukan

3. Large Class

```
public class OrderProcessor {
```

```
    private String orderId;
    private String customerName;
    private String customerPhone;
    private String customerAddress;
    private List<String> menuItems;
    private double totalPrice;
    private double tax;
    private String paymentMethod;
    private String discountCode;

    public void processOrder(...) {
        ...
    }

    private double applyDiscount(String discountCode, double totalPrice) {
        ...
    }
    private void saveOrder(...) {

        ...
    }
    private void sendNotification(...) {
        ...
    }
}
```

OO Principle yang dilanggar

Melanggar OO Principle SRP, dimana class memiliki lebih dari satu tanggung jawab.

Penyebab

Class OrderProcessor memiliki lebih dari satu tanggung jawab, dimana yang harusnya hanay memprosses order, dia juga memiliki tugas untuk mengirim notifikasi, menghitung pajak, dan menghitung diskon.

Solusi

Memecah class OrderProcessor menjadi beberapa class lebih kecil untuk tugas yang sesuai, sehingga OrderProcessor hanya perlu memanggil class yang sesuai untuk tugas di luar tanggung jawabnya, seperti mengirimkan notifikasi.

4. Long Parameter List

```
public void processOrder(String orderId, String customerName, String customerPhone,
String customerAddress, List<String> menuItems, double totalPrice, double tax, String
paymentMethod, String discountCode) {

    private void saveOrder(String orderId, String customerName, String customerPhone,
String customerAddress, List<String> menuItems, double finalPrice, String
paymentMethod)     {...}
}
```

OO Principle yang dilanggar

Melanggar OO Principle KISS, dimana parameter dengan jumlah banyak yang bisa disederhanakan atau diwakilkan.

Penyebab

Jumlah parameter pada beberapa method memilki lebih dari 3 parameter, dimana parameter-parameter ini digunakan di beberapa method berbeda, dan bisa disederhanakan menjadi bentuk parameter yang lebih sedikit.

Solusi

Membuat objek untuk order agar atribut objeknya bisa digunakan hanya dengan memanggil objek tersebut, bukan memanggil setiap atribut secara sendiri-sendiri.

5. Complex Conditionals

```
if (orderId == null || customerName == null || customerPhone == null || menuItems == null || totalPrice <= 0) {...}
```

OO Principle yang dilanggar

Melanggar OO Principle KISS dan OCP, karena bisa disederhanakan dan membuat rawan akan adanya modifikasi apabila terdapat kondisi baru yang perlu dicek saat validasi.

Penyebab

Jumlah syarat yang banyak pada statement if untuk pengecekankevalidasian order.

Solusi

Membuat method baru untuk penegcekan validitas order yang mengembalikan nilai boolean untuk mneyatakan valid atau tidaknya order tersebut, sehingga kondisional if hanya mengecek terhadao nilai yang dikembalikan.

Studi Kasus #2

TicketBooking.java

```
public class TicketBooking {

    // Method untuk melakukan booking tiket bioskop
    public void bookTicket(String customerName, String customerPhone, String customerEmail,
                           String movieTitle, String movieDate, String movieTime,
                           String seatNumber) {

        // Simulasi pemrosesan pemesanan tiket
        System.out.println("Memproses pemesanan tiket...");
        System.out.println("Pelanggan: " + customerName + ", " + customerPhone + ", " + customerEmail);
        System.out.println("Film: " + movieTitle + " pada " + movieDate + " jam " + movieTime);
        System.out.println("Kursi: " + seatNumber);
        System.out.println("Pesanan selesai.");
    }

    // Method untuk membatalkan tiket yang sudah dipesan
    public void cancelTicket(String customerName, String customerPhone, String customerEmail,
                             String movieTitle, String movieDate, String movieTime,
```

```
                            String seatNumber) {
        // Simulasi pembatalan pemesanan tiket
        System.out.println("Memproses pembatalan tiket...");
        System.out.println("Pelanggan: " + customerName + ", " + customerPhone + ", " +
customerEmail);
        System.out.println("Film: " + movieTitle + " pada " + movieDate + " jam " +
movieTime);
        System.out.println("Kursi: " + seatNumber);
        System.out.println("Pesanan dibatalkan.");
    }

    // Method untuk mengganti jadwal tiket
    public void rescheduleTicket(String customerName, String customerPhone, String
customerEmail,
                                 String movieTitle, String oldMovieDate, String oldMovieTime,
                                 String newMovieDate, String newMovieTime,
                                 String seatNumber) {
        // Simulasi penggantian jadwal tiket
        System.out.println("Memproses penggantian jadwal tiket...");
        System.out.println("Pelanggan: " + customerName + ", " + customerPhone + ", " +
customerEmail);
        System.out.println("Film: " + movieTitle);
        System.out.println("Jadwal lama: " + oldMovieDate + " jam " + oldMovieTime);
        System.out.println("Jadwal baru: " + newMovieDate + " jam " + newMovieTime);
        System.out.println("Kursi: " + seatNumber);
        System.out.println("Jadwal berhasil diganti.");
    }
}
```

Code Smells

1. Comments

```
    // Method untuk melakukan booking tiket bioskop
        // Simulasi pemrosesan pemesanan tiket
    // Method untuk membatalkan tiket yang sudah dipesan
        // Simulasi pembatalan pemesanan tiket
    // Method untuk mengganti jadwal tiket
        // Simulasi penggantian jadwal tiket
```

OO Principle yang dilanggar

Melanggar OO Principle YAGNI, dimana terdapat comment yang tidak berguna dan tidak memberikan tambahan bernilai terhadap kode yang ada.

Penyebab

Adanya comment yang menjelaskan kegunaan method yang dimana tanpa adanya comment tersebut, kegunaan kode bisa dipahami dari penamaan method dan atribut yang digunakan.

Solusi

Menghapus comment pada kode yang tidak perlu dijelaskan karena sudah *self-explanatory* berdasarkan penamaannya.

2. Large Class

```
public class TicketBooking {

    public void bookTicket(...) {...}

    public void cancelTicket(...) {...}

    public void rescheduleTicket(...) {...}
}
```

OO Principle yang dilanggar

Melanggar OO Principle SRP, dimana class memiliki lebih dari satu tanggung jawab.

Penyebab

Class TicketBooking memiliki lebih dari satu tanggung jawab, dimana yang harusnya hanay membooking tiket, dia juga memiliki tugas untukmembatalkan dan mengubah jadwal dari tiket.

Solusi

Memecah class TicketBooking menjadi tiga kelas berbeda untuk booking, cancelling, dan rescheduling.

3. Long Parameter List

```
    public void bookTicket(String customerName, String customerPhone, String
customerEmail,String movieTitle, String movieDate, String movieTime,String seatNumber)
{...}

    public void cancelTicket(String customerName, String customerPhone, String
customerEmail,String movieTitle, String movieDate, String movieTime,String seatNumber)
{...}

    public void rescheduleTicket(String customerName, String customerPhone, String
customerEmail,String movieTitle, String oldMovieDate, String oldMovieTime,String
newMovieDate, String newMovieTime,String seatNumber) {...}
```

OO Principle yang dilanggar

Melanggar OO Principle KISS, dimana parameter dengan jumlah banyak yang bisa disederhanakan atau diwakilkan.

Penyebab

Jumlah parameter pada beberapa method memilki lebih dari 3 parameter dan bisa disederhanakan menjadi bentuk objek yang menjadikan parameter lebih sedikit.

Solusi

Membuat objek untuk ticket agar atribut objeknya bisa digunakan hanya dengan memanggil objek tersebut, bukan memanggil setiap atribut sendiri-sendiri.

4. Duplicate Code

```
        // Simulasi pemrosesan pemesanan tiket
        System.out.println("Memproses pemesanan tiket...");
        System.out.println("Pelanggan: " + customerName + ", " + customerPhone + ", " +
customerEmail);
        System.out.println("Film: " + movieTitle + " pada " + movieDate + " jam " +
movieTime);
        System.out.println("Kursi: " + seatNumber);
        System.out.println("Pesanan selesai.");


        // Simulasi pembatalan pemesanan tiket
        System.out.println("Memproses pembatalan tiket...");
        System.out.println("Pelanggan: " + customerName + ", " + customerPhone + ", " +
customerEmail);
        System.out.println("Film: " + movieTitle + " pada " + movieDate + " jam " +
movieTime);
        System.out.println("Kursi: " + seatNumber);
        System.out.println("Pesanan dibatalkan.");


        // Simulasi penggantian jadwal tiket
        System.out.println("Memproses penggantian jadwal tiket...");
        System.out.println("Pelanggan: " + customerName + ", " + customerPhone + ", " +
customerEmail);
        System.out.println("Film: " + movieTitle);
        System.out.println("Jadwal lama: " + oldMovieDate + " jam " + oldMovieTime);
        System.out.println("Jadwal baru: " + newMovieDate + " jam " + newMovieTime);
        System.out.println("Kursi: " + seatNumber);
        System.out.println("Jadwal berhasil diganti.");
```

OO Principle yang dilanggar

Melanggar OO Principle DRY, dimana terdapat perulangan kode yang sama yang seharusnya bisa dihindari.

Penyebab

Adanya perulangan kode yang hampir sama pada setiap method yang berinteraksi dengan tiket, dimana selain menggunakan parameter yang sama, pesan yang dihasilkan juga hampir sama, kecuali pada rescheduling yang terdapat satu baris ppesan tambahan saja.

Solusi

Membuat method untuk menampilkan pesan yang sama tersebut sehingga untuk menampilkan pesan tersebut hanya perlu memanggil method yang sudah dibuat, bukan menuliskan beberapa baris kode yang sama di tempat yang berbeda.

Studi Kasus #3

```
public class PaymentProcessor {
    public double processPayment(String paymentMethod, double amount) {
        double fee = 0.0;

        switch (paymentMethod) {
            case "creditCard":
```

```
                fee = 0.02 * amount; // 2% fee for credit card
                System.out.println("Processing credit card payment...");
                break;
            case "paypal":
                fee = 0.03 * amount; // 3% fee for PayPal
                System.out.println("Processing PayPal payment...");
                break;
            case "bankTransfer":
                fee = 0.01 * amount; // 1% fee for bank transfer
                System.out.println("Processing bank transfer...");
                break;
            default:
                System.out.println("Unknown payment method.");
                break;
        }

        return amount + fee;
    }
}
```

Code Smells

1. Switch Statements

```
        switch (paymentMethod) {
            case ...
            case ...
            case ...
            default:
                ...
        }
```

OO Principle yang dilanggar

Melanggar OO Principle OCP, dimana jika terjadi perubahan seperti penambahan metode pembayaran, maka harus memodifikasi kode.

Penyebab

Penggunaan swtch case untuk menentukan apa yang harus dilakukan pada setiap metode pembayaran yang berbeda, yang manan jika nanti ada metode pembayaran baru, maka akan perlu dilakukan modifikasi pada switch statement yang ada.

Solusi

Membuat interface untuk metode pembayaran, dimana class untuk metode pembayaran spesifiknya merealisasikan interface tersebut. Dileknggkapi dnegan exception hendling untuk mengatasi nilai yang tidak termasuk metode pembayaran yang ada, seperti pada saat penggunaan case default pada switch statemnt.

2. Duplicate Code

```
                fee = 0.02 * amount; // 2% fee for credit card
                System.out.println("Processing credit card payment...");
                break;
```

```
                fee = 0.03 * amount; // 3% fee for PayPal
                System.out.println("Processing PayPal payment...");
                break;

                fee = 0.01 * amount; // 1% fee for bank transfer
                System.out.println("Processing bank transfer...");
                break;
```

OO Principle yang dilanggar

Melanggar OO Principle DRY, dimana terdapat perulangan kode yang hampir sama yang seharusnya bisa dihindari.

Penyebab

Adanya perulangan kode yang hampir sama pada setiap case untuk perhitungan fee dan memunculkan pesan yang menyatakan pemrosesan pembayaran.

Solusi

Melakukan ekstraksi dan melakukan override method untuk mengubah bagian pesan yang berbeda dan perhitungan berbeda tersebut.

3. Primitive Obsession

```
        switch (paymentMethod) {
            case "creditCard":
                ...
            case "paypal":
                ...
            case "bankTransfer":
                ...
        }
```

OO Principle yang dilanggar

Melanggar OO Principle KISS dan OCP, dimana dengan penggunaan string untuk pernyataan sebuah tipe membuat kode lebih rumit dan rawan untuk lebih sulit dipahami, juga membuat kode rawan akan terkena perubahan.

Penyebab

Digunakannya string untuk membedakan antar jenis pembayaran yang ada, yang juga menyebabkan rawan akan terjadinya modifikasi terhadap string tersebut. Pembedaan domain tersebut lebih baik untuk diganti sebagai objek atau enum.

Solusi

Mengekstraksi menjadi interface dimana tipe metode pembayaran yang berbeda merealisasikan interface yang dibuat sebagai objek yang berbedaa.

Studi Kasus #4

```java
public class Order {
    private String customerName;
    private double orderAmount;

    // Temporary field, only used in specific cases
    private String specialInstructions;

    public Order(String customerName, double orderAmount) {
        this.customerName = customerName;
        this.orderAmount = orderAmount;
    }

    // Method to add special instructions (only used in rare cases)
    public void addSpecialInstructions(String specialInstructions) {
        this.specialInstructions = specialInstructions;
    }

    // Method to print order details
    public void printOrderDetails() {
        System.out.println("Customer: " + customerName);
        System.out.println("Amount: $" + orderAmount);
        if (specialInstructions != null) {
            System.out.println("Special Instructions: " + specialInstructions);
        }
    }
}
```

Code Smells

1. Comments

```java
    // Temporary field, only used in specific cases
    // Method to add special instructions (only used in rare cases)
    // Method to print order details
```

OO Principle yang dilanggar

Melanggar OO Principle YAGNI, dimana terdapat comment yang tidak berguna dan tidak memberikan tambahan bernilai terhadap kode yang ada.

Penyebab

Adanya comment yang menjelaskan kegunaan method yang dimana tanpa adanya comment tersebut, kegunaan kode bisa dipahami dari penamaan method dan atribut yang digunakan.

Solusi

Menghapus comment pada kode yang tidak perlu dijelaskan karena sudah *self-explanatory* berdasarkan penamaannya.

2. Temporary Fields

```java
    // Temporary field, only used in specific cases
```

```
    private String specialInstructions;
```

OO Principle yang dilanggar

Melanggar OO Principle YAGNI, dimana terdapat atribut yang memiliki kemungkina tidak terpakai dalam class tersebut.

Penyebab

Adanya atribut yang hanya digunakan pada kasus tertentu saja, sehingga selain pada kasus tertentu tersebut saja, akan bernilai kosong.

Solusi

Melakukan ekstraksi class untuk mengubahnya menjadi method object.

3. Speculative Generailty

```
        if (specialInstructions != null) {
            System.out.println("Special Instructions: " + specialInstructions);
        }
```

OO Principle yang dilanggar

Melanggar OO Principle YAGNI, dimana terdapat kode dalam class yang hanya digunakan dalam kasus tertentu saja.

Penyebab

Adanya pengecekan terhadap temporary field, dimana jika temporary field tersebut, maka baris kode tersebut tidak dieksekusi, yang menjadikannya hanya untuk kebutuhan spesifik saja.

Solusi

Mengekstraksi menjadi subclass yang melakukan ekstensi terhadap class order, sehingga untuk kasus adanya parameter berupa specialInstructions(si temporary field), bukan dilakukan penegcekan untuk melakukan eksekusi kode melainkan sudah pasti akan mengeksekusi kode tersebut.

Link Github

Hasil refactoring bisa dilihat pada:
[fatimahAMahros/Refactoring_Study_Case]

Referensi

- https://refactoring.guru/

- https://wiki.c2.com/?CodeSmell