Laporan Code Smell
“Workshop Rekayasa Ulang Kode”
Dosen : Adam Shidqul Aziz S.ST., M.T

Disusun Oleh :

Moch. Irham Kafi Billah

3122600009

3 D4 Teknik Informatika-A

PROGRAM STUDI TEKNIK INFORMATIKA

DEPARTEMEN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

TAHUN 2024

Praktikum

A. Soal 1

1.

```java
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) {
        ....
    }

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

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

    private void sendNotification(String phoneNumber, String customerName, String message) {
        ....
    }
}
```

a. Code Smell

Large Class Karena pada satu class menangani banyak tugas yang seharusnya bisa hanya satu tugas yaitu proses order saja

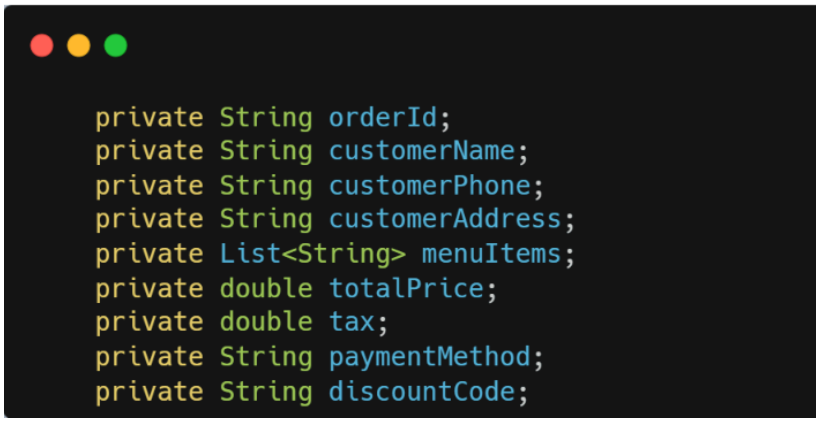
b. Principle yang dilanggar

SRP karena memiliki lebih dari satu tanggung jawab (memproses pesanan, menghitung diskon, menyimpan pesanan, dan mengirim notifikasi).

c. Cara Refactoring

membuat class baru untuk masing masing tugas

2.



```
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;
```

a. Code Smell

Dead Code Karena atribute tidak pernah dipanggil pada tiap method

b. Principle yang dilanggar

Yagni "You Aren't Gonna Need It" karena attribute tidak digunakan sama sekali

c. Cara Refactoring

Hapus dead code yang ada

3.

```
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);
    }
```

a. Code Smell

Long Parameter List karena method processOrder memiliki parameter lebih dari 3

b. Principle yang dilanggar

Kiss karena memiliki parameter yang panjang yang seharusnya bisa dibuat simpel

c. Cara refactoring

membuat class baru yang berisikan mulai dari order id sampai discount code

4.

```
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;
    }
}
```

a. Code Smell

Magic String karena “DISC10” dan “DISC20” di define langsung

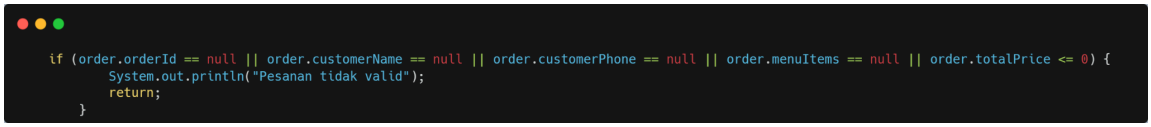
b. Principle yang dilanggar

Open/Closed Principle (OCP) karena jika ingin menambahkan lebih banyak discount akan perlu mengubah kode yang sudah ada

c. cara refactoring

membuat konstanta untuk DISC10 dan DISC20

5.



```
if (order.orderId == null || order.customerName == null || order.customerPhone == null || order.menuItems == null || order.totalPrice <= 0) {
        System.out.println("Pesanan tidak valid");
        return;
    }
```

a. Code Smell

Long Method karena memiliki banyak kondisi untuk memvalidasi order

b. Principle yang dilanggar

- Keep It Simple Stupid(Kiss) karena kode diatas bisa disederhanakan dan tidak sepanjang dan sekompleks itu
- Open/Closed Principle (OCP) karena jika ingin menambahkan lebih banyak aturan validasi di masa depan, Anda harus mengubah kode yang ada. Misalnya, jika Anda ingin menambahkan validasi baru untuk memastikan bahwa totalPrice tidak hanya lebih besar dari 0 tetapi juga tidak lebih dari 1000, Anda perlu mengubah kode di dalam metode yang sama. Ini menunjukkan bahwa kode tidak terbuka untuk ekstensi.
- SRP karena pada code diatas memiliki tanggung jawab yang banyak

c. Cara Refactoring

membuat method valid order

B. Soal 2

1.

```java
public class TicketBooking {

    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.");
    }


    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.");
    }


    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.");
    }
}
```

a. Code Smell

Large class karena Kelas TicketBooking memiliki beberapa metode yang menangani berbagai aspek pemesanan tiket yaitu Memesan tiket, Membatalkan tiket, Mengganti jadwal tiket

b. Principle yang dilanggar

Single Responsibility Principle (SRP) karena sebuah kelas seharusnya hanya memiliki satu alasan untuk berubah. Jika ada perubahan dalam logika pemesanan, pembatalan, atau penggantian jadwal, Anda harus memodifikasi kelas ini.

c. Cara refactoring

membagi method menjadi beberapa class sesuai dengan SRP

2.

```java
    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.");
    }
```

a. Code Smell

Long Parameter List karena parameter pada method di atas berjumlah 7 parameter

b. Principle yang dilanggar

KISS karena memiliki parameter yang panjang yang seharusnya bisa dibuat simpel

c. Cara Refactoring

membuat class baru yang bisa menampung parameter yang memiliki keterkaitan

3.

```java
public class TicketBooking {

    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.");
    }


    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.");
    }


    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.");
    }
}
```

a. Code Smell

Duplicate code karena ada beberapa baris kode yang sama pada tiap tiap method

b. Principle yang dilanggar

DRY karena adanya redudansi code pada tiap tiap method melanggar prinsip DRY

c. Cara Refactoring

dipisahkan menjadi sebuah method tersendiri atau class tersendiri

4.
```java
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.");
}
```

a. Code Smell

Long Method karena metode dalam kelas ini cukup panjang dan melakukan banyak tugas dalam satu metode. dalam metode bookTicket, Anda melakukan pemrosesan dan mencetak beberapa informasi. Ini membuat metode sulit untuk dibaca dan dipahami

b. Principle yang dilanggar

- Kiss karena baris kode yang cukup panjang
- SRP karena terdapat 2 tugas yaitu melakukan pemrosesan dan mencetak beberapa informasi

c. Cara Refactoring

memotong method menjadi beberapa method

5.
```java
public void bookTicket(String customerName, String customerPhone, String customerEmail,
                       String movieTitle, String movieDate, String movieTime,
                       String seatNumber) {}
```

a. Code Smell

Data Clumps karen Parameter yang sering muncul bersama (seperti customerName, customerPhone, customerEmail) menunjukkan bahwa mereka mungkin perlu dikelompokkan dalam objek terpisah, misalnya, kelas Customer. Ini akan mengurangi jumlah parameter yang harus diteruskan ke setiap metode dan membuat kode lebih bersih.

b. Principle yang dilanggar

- Single Responsibility Principle (SRP) Karena tanggung jawab data tidak terpusat.

c. Cara Refactoring

membuatkan sebuah class tersendiri

C. Soal 3

1. 

```java
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;
    }
}
```

a. Code Smell

Primitive Obsession karena Penggunaan string literal untuk paymentMethod adalah contoh primitive obsession. Menggunakan enum atau objek untuk merepresentasikan jenis metode pembayaran akan lebih baik.

b. Prinsip yang Dilanggar

KISS (Keep It Simple, Stupid) karena Menggunakan tipe primitif untuk merepresentasikan konsep yang lebih kompleks dapat membuat kode lebih rumit dan sulit dipahami. Dengan menggunakan objek atau struktur data yang lebih terorganisir, Anda dapat menjaga kode tetap sederhana dan lebih mudah dipahami.

c. Cara Refactoring

mengubah menjadi beberapa class dengan interface

2.

```java
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;
    }
}
```

a. Code Smell

Switch Statements karena Penggunaan switch dalam processPayment adalah code smell. Jika metode pembayaran baru ditambahkan, Anda harus memodifikasi kode ini, melanggar Open-Closed Principle. Refactoring menggunakan polymorphism akan lebih baik.

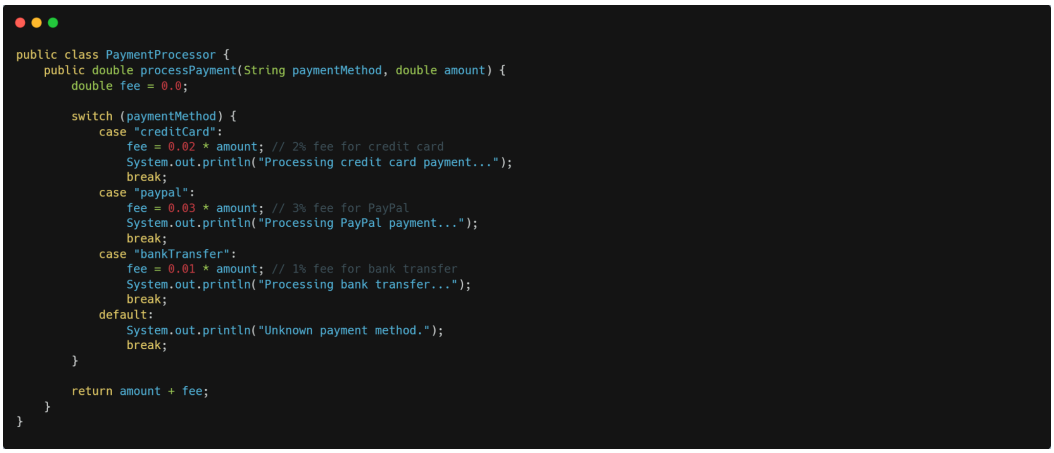
b. Prinsip yang Dilanggar

Open-Closed Principle karena perlu mengubah seluruh method jika ingin menambah metode pembayaran

c. Cara Refactoring

menggunakan polymorphism untuk menggantikan switch.

3.



```java
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;
    }
}
```

a. Code Smell

Divergent Changes karena Jika kebutuhan pembayaran berubah (misalnya, penambahan metode pembayaran baru), Anda harus memodifikasi processPayment, yang membuat kelas ini rentan terhadap divergent change. Solusi yang lebih modular akan mengurangi risiko ini.

b. Prinsip yang Dilanggar

Open-Closed Principle karena perlu mengubah seluruh method jika ingin menambah metode pembayaran

c. Cara Refactoring

Pecah tanggung jawab menjadi kelas terpisah untuk setiap metode pembayaran.

D. Soal 4

1. 

```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);
        }
    }
}
```

a. Code Smell

Temporary Field karena Properti specialInstructions adalah contoh temporary field. Properti ini hanya relevan dalam kasus tertentu, tetapi tetap ada sepanjang siklus hidup objek, sehingga dapat menyebabkan kebingungan.

b. Prinsip yang Dilanggar

Yagni "You Aren't Gonna Need It" karena specialInstructions tidak selalu dibutuhkan pada tiap order

c. Cara Refactoring

membuat kelas khusus untuk menangani instruksi tambahan, misalnya, SpecialOrder sebagai subkelas dari Order, atau gunakan objek tambahan untuk decorate pesanan.