

VERSION 1.0

JULI 18, 2023



# PEMROGRAMAN LANJUT

## MODUL 2 – SIMPLE REFACTORING

TIM PENYUSUN:

- WILDAN SUHARSO, S.KOM., M.KOM

- LARYNT SAWFA KENANGA

- AHYA NIKA SALSABILA

LAB. INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

## PEMROGRAMAN LANJUT

---

### PERSIAPAN MATERI

Mahasiswa harus memahami konsep OOP yang telah dipelajari sebelumnya

---

### TUJUAN

1. Mahasiswa mampu memahami refactor.
2. Mahasiswa mampu memahami teknik-teknik refactor.
3. Mahasiswa mampu melakukan/mengimplementasikan refactoring.

---

### TARGET MODUL

1. Mahasiswa dapat memahami konsep refactoring.
2. Mahasiswa mampu memahami dan mengimplementasikan konsep refactoring.

---

### PERSIAPAN SOFTWARE/APLIKASI

1. Java Development Kit
2. Text Editor / IDE (**Preferably** IntelliJ IDEA, Visual Studio Code, Netbeans, etc).

---

### MATERI POKOK

#### A. Apa itu Refactoring

Refactoring adalah proses mengubah kode dalam sebuah perangkat lunak dengan tujuan untuk meningkatkan kualitas struktur dan desain tanpa mengubah perilaku fungsional dari perangkat lunak tersebut. Praktik refactoring dilakukan untuk membuat kode menjadi lebih bersih, mudah dipahami, dan lebih mudah diubah tanpa mengorbankan fungsionalitasnya.

#### B. Kapan Perlu Dilakukan Refactoring

1. Saat menambahkan fitur

Ketika kita menambahkan fitur baru ke dalam perangkat lunak, ada peluang untuk melakukan refactoring pada kode yang sudah ada. Refactoring membantu membersihkan dan meningkatkan struktur kode sehingga lebih mudah untuk menambahkan fitur baru tanpa mengganggu fungsionalitas yang sudah ada. Dengan memastikan kode sudah bersih dan terorganisir dengan baik sebelum menambahkan fitur baru, kita dapat menghindari kemungkinan konflik dan masalah di masa mendatang.

2. Saat memperbaiki bug

Ketika kita menemukan bug atau kesalahan dalam perangkat lunak, refactoring dapat menjadi langkah penting dalam proses perbaikan. Menggunakan refactoring untuk membersihkan kode yang terlibat dalam bug dapat membantu kita lebih mudah menemukan sumber kesalahan dan membuat perbaikan dengan lebih efisien. Kode yang bersih dan

terstruktur dengan baik akan memudahkan identifikasi masalah dan mencegah kemungkinan bug di tempat lain yang mungkin tersembunyi dalam kode yang berantakan.

### 3. Selama proses ulasan code (code review)

Proses ulasan kode adalah kesempatan untuk mendapatkan masukan dari anggota tim. Selama ulasan kode, refactoring dapat dilakukan untuk meningkatkan kualitas kode sebelum kode tersebut diterima dan digabungkan ke dalam repositori utama. Dengan melakukan refactoring selama proses ulasan kode, kita dapat menghindari masalah potensial, meningkatkan kualitas kode, dan memastikan kode mematuhi standar dan pedoman yang telah ditetapkan.

## C. Mengapa Perlu Melakukan Refactoring?

Refactoring dilakukan untuk mencapai kesederhanaan (simplicity) dan kekompakan (brevity) dalam kode. Maksudnya dengan melakukan refactoring, kode akan menjadi lebih mudah dipahami, lebih sederhana, dan lebih ringkas, tanpa mengorbankan fungsionalitasnya. Tetapi perlu diingat bahwa refactoring bukan berarti mengurangi. Tujuan utamanya adalah untuk meningkatkan struktur kode. Berikut adalah beberapa alasan mengapa refactoring diperlukan:

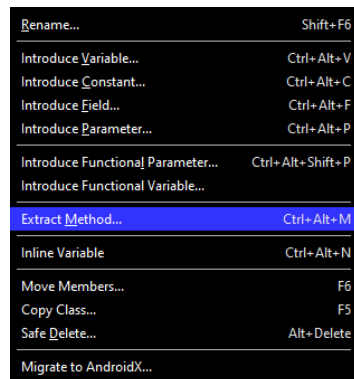
- Refactoring meningkatkan pemahaman terhadap kode yang ditulis oleh pengembang lain.
- Membantu menemukan dan memperbaiki bug.
- Dapat mempercepat kecepatan pengembangan perangkat lunak.
- Secara keseluruhan, meningkatkan desain perangkat lunak.

## D. Teknik Refactoring

Berikut ini adalah beberapa teknik refactoring yang umum digunakan:

### 1. Extract Method

Memisahkan bagian kode yang berfungsi menjadi metode terpisah untuk meningkatkan keterbacaan dan mengurangi duplikasi kode.



Contoh:

#### a. Before

Pada contoh sebelum refactoring, kita memiliki kode untuk menghitung total harga dari beberapa item yang dibeli. Total harga dihitung di dalam metode main() dengan menggunakan perulangan for. Kode untuk menghitung total harga ada di dalam metode main(), sehingga metode tersebut menjadi agak panjang dan kurang terstruktur.

```
package Modul2;

public class BelanjaanApp {
```

```

public static void main(String[] args) {
    String[] items = {"Item 1", "Item 2", "Item 3"};
    int[] prices = {10, 20, 30};

    int totalPrice = 0;
    for (int i = 0; i < items.length; i++) {
        totalPrice += prices[i];
    }

    System.out.println("Total Harga: " + totalPrice);
}

```

b. After

Setelah melakukan refactoring "Extract Method," kita memindahkan kode yang berfungsi untuk menghitung total harga ke metode terpisah yang dinamakan `getTotalPrice()`. Sekarang, metode `main()` hanya bertugas untuk menginisialisasi data dan memanggil metode `getTotalPrice()` untuk mendapatkan total harga. Metode `getTotalPrice()` mengambil array item dan array harga serta total harga sebagai parameter, dan mengembalikan total harga setelah melakukan perhitungan.

```

package Modul2;

public class BelanjaanApp {

    public static void main(String[] args) {
        String[] items = {"Item 1", "Item 2", "Item 3"};
        int[] prices = {10, 20, 30};

        int totalPrice = 0;
        totalPrice = getTotalPrice(items, prices, totalPrice);

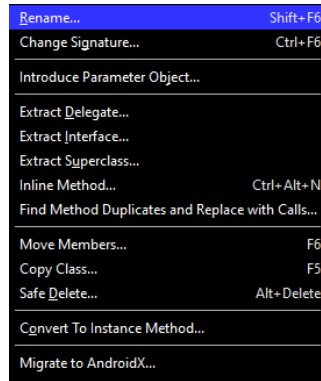
        System.out.println("Total Harga: " + totalPrice);
    }

    private static int getTotalPrice(String[] items, int[] prices,
int totalPrice) {
        for (int i = 0; i < items.length; i++) {
            totalPrice += prices[i];
        }
        return totalPrice;
    }
}

```

2. Rename Method/Variable

Mengubah nama metod atau variabel agar lebih deskriptif dan dapat dipahami dengan mudah.



Contoh:

a. Before

Pada kode di atas, kita memiliki sebuah method dengan nama `calcAvg`. Method ini bertugas untuk menghitung rata-rata dari elemen-elemen dalam array `numbers`. Namun, nama metode ini tidak cukup deskriptif. Nama `calcAvg` tidak menggambarkan tindakan atau fungsionalitas metode tersebut dengan jelas.

```
public void calcAvg(int[] numbers) {
    int sum = 0;
    for (int num : numbers) {
        sum += num;
    }
    int average = sum / numbers.length;
    System.out.println("The average is: " + average);
}
```

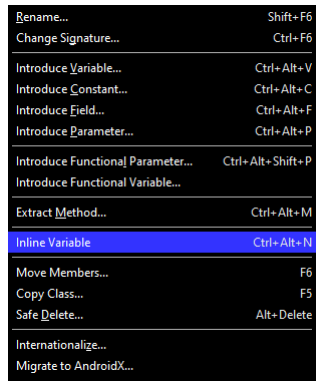
b. After

Setelah refactoring, kami mengubah nama method menjadi `calculateAverage`. Dengan perubahan ini, sekarang nama metode lebih deskriptif dan memberikan informasi yang lebih jelas tentang apa yang dilakukan metode ini, yaitu menghitung rata-rata dari elemen-elemen dalam array `numbers`.

```
public void calculateAverage(int[] numbers) {
    int sum = 0;
    for (int num : numbers) {
        sum += num;
    }
    int average = sum / numbers.length;
    System.out.println("The average is: " + average);
}
```

### 3. Inline Variable

Teknik refactoring ini digunakan untuk menggantikan penggunaan variabel dengan nilai yang sebenarnya. Ketika kita memiliki variabel yang hanya digunakan satu kali dan nilainya sederhana, kita dapat "mengganti" variabel tersebut dengan nilai aslinya untuk meningkatkan kejelasan dan mengurangi kompleksitas kode.



Contoh:

a. Before

Pada contoh sebelum refactoring di atas, kita memiliki dua variabel `a` dan `b` yang menyimpan nilai 10 dan 5. Kemudian, kita memanggil method `add(a, b)` untuk menjumlahkan `a` dan `b` dan menyimpan hasilnya dalam variabel `sum`. Barulah kemudian variabel `sum` tersebut digunakan untuk mencetak hasil penjumlahan.

```
package Modul2;

public class MathOperations {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;

        // Panggil metode untuk penjumlahan dan tampilkan hasil
        int sum = add(a, b);
        System.out.println("Sum: " + sum);
    }

    // Metode untuk penjumlahan
    public static int add(int a, int b) {
        return a + b;
    }
}
```

b. After

Setelah melakukan refactoring "Inline Variable", kita menggantikan penggunaan variabel `sum` dengan pemanggilan langsung ke method `add(a, b)` pada bagian `System.out.println("Sum: " + add(a, b));`. Karena variabel `sum` hanya digunakan satu kali dan nilainya sederhana, maka kita bisa langsung memasukkan hasil penjumlahan ke dalam pernyataan `System.out.println()` tanpa perlu menyimpannya dalam variabel.

```
package Modul2;

public class MathOperations {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
```

```

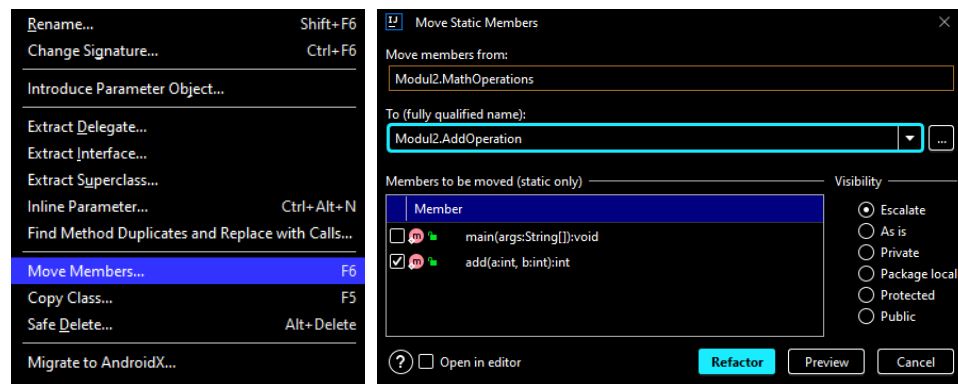
        // Panggil metode untuk penjumlahan dan tampilkan hasil
        System.out.println("Sum: " + add(a, b));
    }

    // Metode untuk penjumlahan
    public static int add(int a, int b) {
        return a + b;
    }
}

```

#### 4. Move Method/Field

Memindahkan metod atau atribut ke kelas yang lebih relevan dan sesuai dengan tanggung jawabnya.



Contoh:

##### a. Before

Pada contoh di atas, kita memiliki kelas MathOperations yang memiliki method add untuk melakukan penjumlahan dua bilangan. Namun, method add ini dapat lebih relevan dan sesuai dengan tanggung jawabnya jika dipindahkan ke kelas lain yang lebih tepat.

```

package Modul2;

public class MathOperations {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        // Panggil metode untuk penjumlahan dan tampilkan hasil
        System.out.println("Sum: " + add(a, b));
    }

    // Metode untuk penjumlahan
    public static int add(int a, int b) {
        return a + b;
    }
}

```

##### b. After

Setelah melakukan refactoring "Move Method/Field", kita memindahkan method add dari kelas MathOperations ke kelas baru bernama AddOperation. Dengan demikian, metode add sekarang berada di kelas yang lebih relevan dan sesuai dengan tanggung jawabnya untuk melakukan penjumlahan.

```
package Modul2;

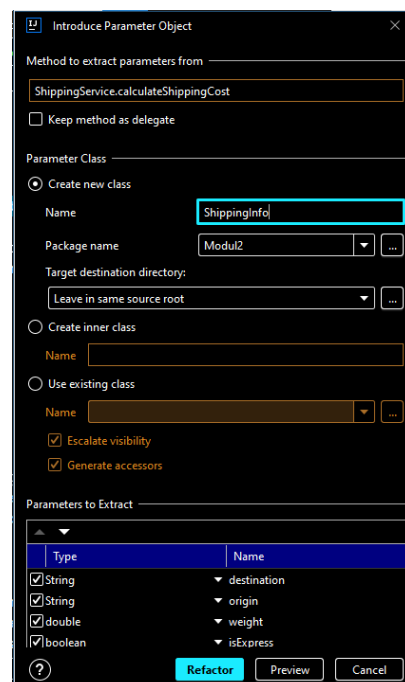
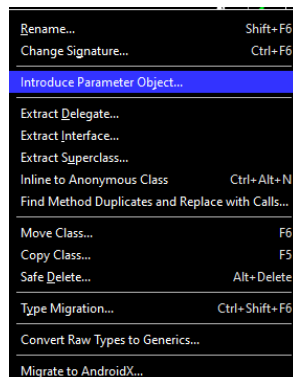
public class MathOperations {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        // Panggil metode untuk penjumlahan dan tampilkan hasil
        System.out.println("Sum: " + AddOperation.add(a, b));
    }
}
```

```
package Modul2;

public class AddOperation {
    // Metode untuk penjumlahan
    public static int add(int a, int b) {
        return a + b;
    }
}
```

##### 5. Introduce Parameter Object

Menggantikan beberapa parameter dengan objek parameter tunggal untuk memperjelas dan mengurangi jumlah parameter.





Contoh:

a. Before

Pada contoh sebelum refactoring, method `calculateShippingCost` memiliki banyak parameter yaitu `destination`, `origin`, `weight`, dan `isExpress`. Jumlah parameter yang banyak bisa mengakibatkan kebingungan dan sulit dipahami jika ada banyak parameter dengan tipe data yang sama.

```
public class ShippingService {

    public double calculateShippingCost(String destination, String
origin, double weight, boolean isExpress) {
        double baseCost = 10.0;
        double distance = calculateDistance(destination, origin);
        double cost = baseCost * weight * distance;

        if (isExpress) {
            cost *= 2.0;
        }

        return cost;
    }

    private double calculateDistance(String destination, String
origin) {
        // implement logic to calculate distance between two
locations
        return 100.0; // dummy value for simplicity
    }

    public static void main(String[] args) {
        ShippingService shippingService = new ShippingService();
        double shippingCost =
shippingService.calculateShippingCost("DestinationA", "OriginA",
5.0, false);
        System.out.println("Shipping cost: " + shippingCost);
    }
}
```

b. After

Setelah melakukan refactoring "Introduce Parameter Object", kita menggantikan beberapa parameter (`destination`, `origin`, `weight`, dan `isExpress`) pada method `calculateShippingCost` dengan objek parameter tunggal `ShippingInfo`. Objek `ShippingInfo` berisi data-data yang sebelumnya diwakilkan oleh parameter-parameter tersebut.

```
package Modul2;

public class ShippingService {

    public double calculateShippingCost(ShippingInfo shippingInfo) {
        double baseCost = 10.0;
```

```

        double distance =
calculateDistance(shippingInfo.destination(),
shippingInfo.origin());
        double cost = baseCost * shippingInfo.weight() * distance;

        if (shippingInfo.isExpress()) {
            cost *= 2.0;
        }

        return cost;
    }

    private double calculateDistance(String destination, String
origin) {
        // implement logic to calculate distance between two
locations
        return 100.0; // dummy value for simplicity
    }

    public static void main(String[] args) {
        ShippingService shippingService = new ShippingService();
        double shippingCost =
shippingService.calculateShippingCost(new
ShippingInfo("DestinationA", "OriginA", 5.0, false));
        System.out.println("Shipping cost: " + shippingCost);
    }
}

```

```

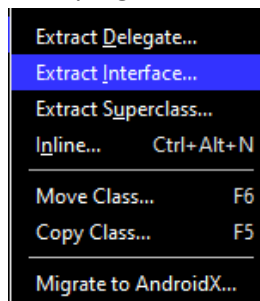
package Modul2;

public record ShippingInfo(String destination, String origin, double
weight, boolean isExpress) {
}

```

## 6. Extract Interface

Membuat antarmuka (interface) dari kelas untuk memfasilitasi abstraksi dan mempermudah penggunaan variasi kelas yang berbeda.



Contoh:

a. Before

Pada contoh sebelum refactoring, kita memiliki kelas Rectangle yang memiliki dua metode calculateArea dan calculatePerimeter untuk menghitung luas dan keliling persegi panjang (rectangle). Namun, tidak ada abstraksi yang terbuat dari kelas ini, sehingga tidak ada kemampuan untuk menggunakan variasi kelas yang berbeda dengan mudah.

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int calculateArea() {
        return width * height;
    }

    public int calculatePerimeter() {
        return 2 * (width + height);
    }
}
```

b. After

Dengan adanya antarmuka Shape, kita mencapai abstraksi dan mempermudah penggunaan variasi kelas yang berbeda. Misalnya, kita bisa membuat kelas lain seperti Circle, Square, atau Triangle yang juga mengimplementasikan antarmuka Shape. Dengan demikian, kita dapat menggunakan objek-objek dari berbagai bentuk geometri tersebut dengan konsisten menggunakan antarmuka Shape.

```
public interface Shape {
    int calculateArea();
    int calculatePerimeter();
}

public class Rectangle implements Shape {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public int calculateArea() {
        return width * height;
    }
}
```

```
@Override
public int calculatePerimeter() {
    return 2 * (width + height);
}
}
```

## 7. Replace Magic Number with Symbolic Constant

Menggantikan angka tetap (magic number) dengan konstanta simbolik untuk memperjelas makna angka tersebut.

Introduce Variable...	Ctrl+Alt+V
Introduce Constant...	Ctrl+Alt+C
Introduce Field...	Ctrl+Alt+F
Introduce Parameter...	Ctrl+Alt+P
Introduce Functional Parameter...	Ctrl+Alt+Shift+P
Introduce Functional Variable...	
Extract Method...	Ctrl+Alt+M
Inline...	Ctrl+Alt+N
Move Instance Method...	F6
Copy Class...	F5
Migrate to AndroidX...	

Contoh:

### a. Before

Pada contoh sebelum refactoring di atas, dalam metode calculateArea, kita mengalikan radius dengan nilai tetap 3.14 untuk menghitung luas lingkaran. Angka 3.14 dalam hal ini disebut sebagai "magic number" karena nilainya tidak dijelaskan atau diberi nama yang dapat memahami makna sebenarnya.

```
public class Circle {
    private double radius;

    // Constructors, getters, setters, etc.
    // ...

    public double calculateArea() {
        return 3.14 * radius * radius;
    }
}
```

### b. After

Setelah melakukan refactoring "Replace Magic Number with Symbolic Constant", kita menggantikan angka tetap (magic number) 3.14 dengan konstanta simbolik PI. Konstanta ini dideklarasikan sebagai private static final double PI = 3.14; dan digunakan dalam metode calculateArea untuk menghitung luas lingkaran.

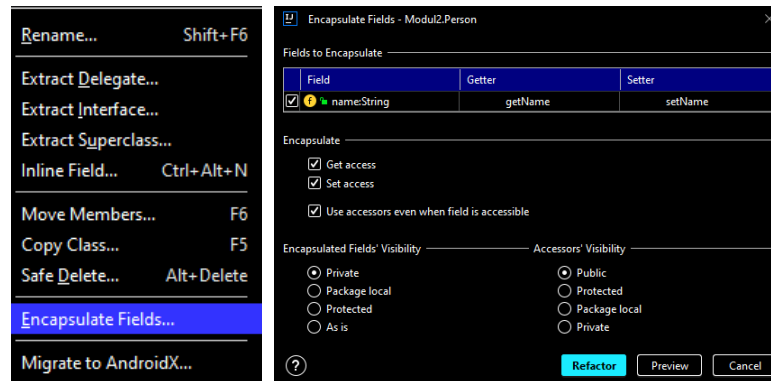
```
public class Circle {
    private static final double PI = 3.14;
    private double radius;
```

```
// Constructors, getters, setters, etc.
// ...

public double calculateArea() {
    return PI * radius * radius;
}
}
```

## 8. Encapsulate Field

Membungkus atribut dengan metode getter dan setter untuk melindungi data dan menerapkan prinsip encapsulation.



Contoh:

### a. Before

```
public class Person {
    public String name;

    // Constructors, etc.
    // ...
}
```

### b. After

```
public class Person {
    private String name;

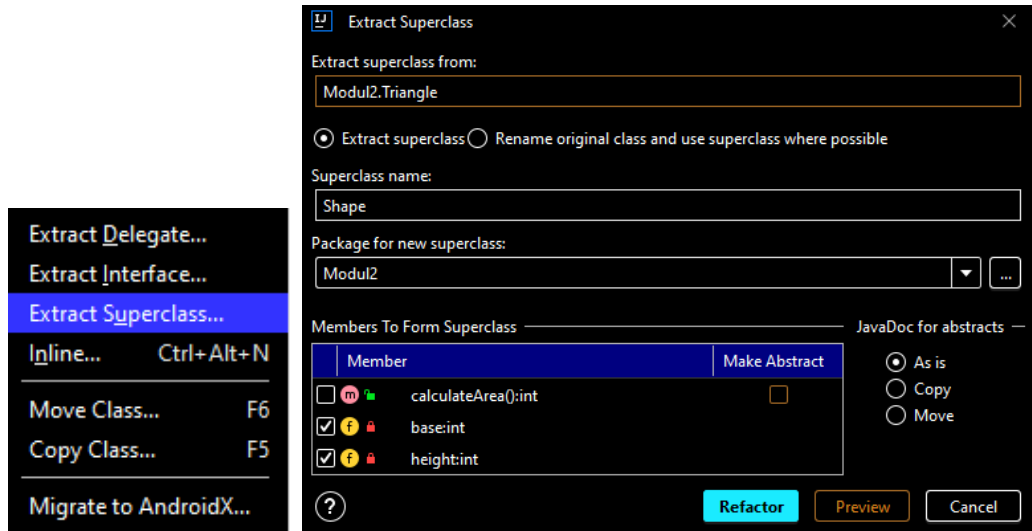
    // Constructors, etc.
    // ...

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

## 9. Extract Superclass

Membuat superclass atau interface dari kelas-kelas yang memiliki kesamaan fungsi atau atribut.



Contoh:

a. Before

```
package Modul2;

public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int calculateArea() {
        return width * height;
    }

    // other methods specific to Rectangle
}
```

b. After

```
package Modul2;

public class Shape {
    protected int width;
    protected int height;

    public Shape(int width, int height) {
        this.width = width;
        this.height = height;
    }
}
```

```
package Modul2;

public class Rectangle extends Shape {

    public Rectangle(int width, int height) {
        super(width, height);
    }

    public int calculateArea() {
        return width * height;
    }

    // other methods specific to Rectangle
}
```

#### E. Bagaimana Cara Melakukan Refactoring?

Berikut adalah contoh refactoring dengan menggunakan IDE IntelliJ IDEA Community Edition:

1. Identifikasi bagian kode yang ingin direfactor. Misalnya, kita ingin mengekstrak perhitungan menjadi method-method baru.
2. Siapkan baris kode yang akan di refactoring (extract method)

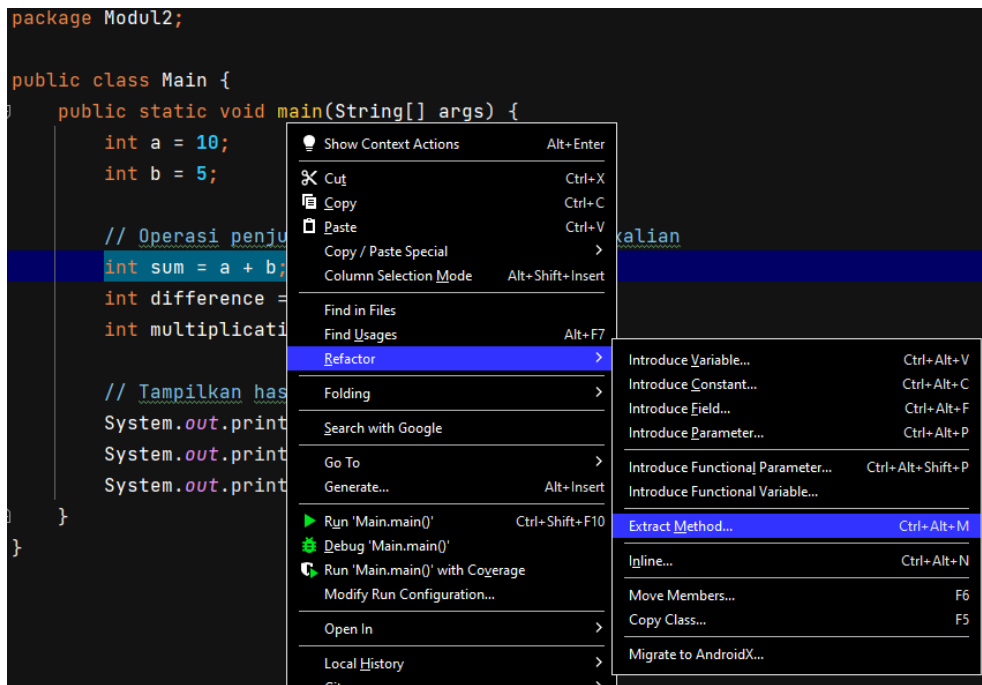
```
package Modul2;

public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;

        // Operasi penjumlahan, pengurangan, dan perkalian
        int sum = a + b;
        int difference = a - b;
        int multiplication = a * b;

        // Tampilkan hasil
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Multiplication: " + multiplication);
    }
}
```

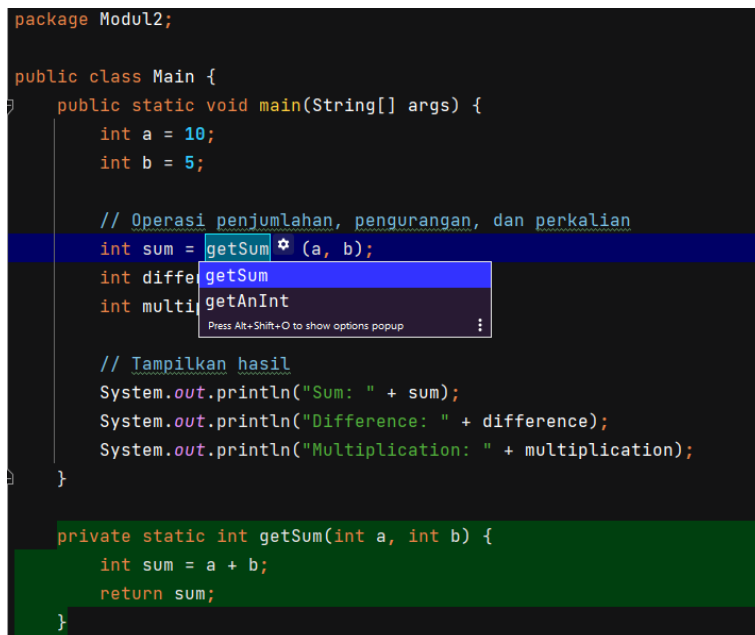
3. Select baris yang akan diekstrak menjadi sebuah method, kemudian klik kanan > Refactor > Extract Method.



Selain itu, kalian juga bisa membuka menu refactoring dengan cara lain. Menu refactoring juga ada di bagian atas (main menu).



4. Selanjutnya akan muncul pop up untuk penamaan method hasil extract. Penamaan method bisa diambil dari nama method yang sudah tersedia di pop up atau bisa juga custom untuk menamai method tersebut. Jika sudah klik enter.



5. IDE akan secara otomatis membuat method baru hasil extract. Berikut adalah hasil extract method dari operasi penjumlahan, pengurangan, dan perkalian serta extract method dari bagian tampilan hasil (display)



```

package Modul2;

public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;

        // Operasi penjumlahan, pengurangan, dan perkalian
        int sum = getSum(a, b);
        int difference = getDifference(a, b);
        int multiplication = getMultiplication(a, b);

        // Tampilkan hasil
        Display(sum, difference, multiplication);
    }

    private static void Display(int sum, int difference, int
multiplication) {
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Multiplication: " + multiplication);
    }

    private static int getMultiplication(int a, int b) {
        int multiplication = a * b;
        return multiplication;
    }

    private static int getDifference(int a, int b) {
        int difference = a - b;
        return difference;
    }

    private static int getSum(int a, int b) {
        int sum = a + b;
        return sum;
    }
}

```

---

## REFERENSI

<https://codegym.cc/groups/posts/196-how-refactoring-works-in-java>

<https://refactoring.guru/refactoring>

<https://code.visualstudio.com/docs/java/java-refactoring>

<https://www.methodsandtools.com/archive/archive.php?id=4>

<https://www.methodsandtools.com/archive/archive.php?id=4>

---

## LATIHAN PRATIUM

### LATIHAN 1

```
package Modul2.Latihan;

public class MessyCode {
    public double r;

    public MessyCode(double radius) {
        this.r = radius;
    }

    // Main method
    public static void main(String[] args) {
        MessyCode circle = new MessyCode(2.5);
        double area;
        area = 3.14 * circle.r * circle.r;
        System.out.println("Radius: " + circle.r);
        System.out.println("Area: " + area);
    }
}
```


Program di atas ini adalah program perhitungan luas lingkaran. Tetapi, program tersebut masih sulit dipahami, sulit diubah, dan tidak terstruktur dengan baik sehingga perlu dilakukan refactoring. Oleh karena itu, lakukanlah refactoring pada program tersebut dengan memperhatikan petunjuk di bawah ini:

- Ubah nama class yang awalnya **MessyCode** menjadi **Circle** (Clue: rename)
- Gantilah Magic Number **3.14** menjadi **Konstanta PHI** yang bernilai 3.14 (Clue: Introduce Constant)
- Lakukanlah refactoring dengan hasil seperti di bawah ini pada perhitungan luas lingkaran sehingga tidak memerlukan lagi variabel area. (Clue: inline variable)

```

area = PHI * circle.r * circle.r;
System.out.println("Radius: " + circle.r);
System.out.println("Area: " + area);

```



```

System.out.println("Radius: " + circle.r);
System.out.println("Area: " + PHI * circle.r * circle.r);


```

- Ekstrak rumus perhitungan luas menjadi sebuah method baru dengan nama **getArea()**. (Clue: Extract Method)

```

public static void main(String[] args) {
    Circle circle = new Circle(radius: 2.5);
    System.out.println("Radius: " + circle.r);
    System.out.println("Area: " + PHI * circle.r * circle.r);
}

```



```

public static void main(String[] args) {
    Circle circle = new Circle(radius: 2.5);
    System.out.println("Radius: " + circle.r);
    System.out.println("Area: " + getArea(circle));
}

1 usage
private static double getArea(Circle circle) {
    return PHI * circle.r * circle.r;
}

```

- Encapsulate field dengan menambahkan getter dan setter untuk variabel **r**. (Clue: Encapsulate Field)

## LATIHAN 2

Sebutkan teknik refactoring apa saja yang digunakan pada soal Latihan 1 !

## TUGAS

### TUGAS 1

Buatlah sebuah class sistem tiket konser dengan nama kelas **ConcertTicketSystem**, lalu tulis ulang kode seperti berikut !

```

package Modul2.Tugas;

import java.util.Scanner;

class ConcertTicketSystem {

    private int[][] seatPrices;
    private boolean[][] seatAvailability;
    private int totalRows;
    private int totalCols;
    private int seatsAvailable;

    public ConcertTicketSystem(int rows, int cols) {
        this.totalRows = rows;
        this.totalCols = cols;
        this.seatsAvailable = rows * cols;
        initializeSeatPricesAndAvailability();
    }
}

```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the number of rows in the concert hall: ");
    int rows = scanner.nextInt();
    System.out.print("Enter the number of columns in the concert hall: ");
    int cols = scanner.nextInt();

    ConcertTicketSystem ticketSystem = new ConcertTicketSystem(rows, cols);

    System.out.println("Welcome to the Concert Ticket System!");
    ticketSystem.showSeatPrices();

    char choice;
    do {
        System.out.println("\nOptions:");
        System.out.println("1. View Available Seats");
        System.out.println("2. Buy Ticket");
        System.out.println("3. View Available Seats Count");
        System.out.println("4. Exit");
        System.out.print("Enter your choice (1/2/3/4): ");
        choice = scanner.next().charAt(0);

        switch (choice) {
            case '1':
                ticketSystem.showAvailableSeats();
                break;
            case '2':
                System.out.print("Enter row number: ");
                int row = scanner.nextInt();
                System.out.print("Enter column number: ");
                int col = scanner.nextInt();
                ticketSystem.buyTicket(row, col);
                break;
            case '3':
                System.out.println("Available Seats Count: " + ticketSystem.seatAvailability);
                break;
            case '4':
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } while (choice != '4');

    scanner.close();
}

private void initializeSeatPricesAndAvailability() {
    seatPrices = new int[totalRows][totalCols];
    seatAvailability = new boolean[totalRows][totalCols];

    for (int i = 0; i < totalRows; i++) {
        for (int j = 0; j < totalCols; j++) {
            seatPrices[i][j] = 50;
            seatAvailability[i][j] = true;
        }
    }
}

public void showSeatPrices() {
    System.out.println("Seat Prices:");
    for (int i = 0; i < totalRows; i++) {
        for (int j = 0; j < totalCols; j++) {
            System.out.println("Row " + (i + 1) + ", Column " + (j + 1) + ": $" + seatPrices[i]
[j]);
        }
    }
}
}

```

```

public void showAvailableSeats() {
    System.out.println("Available Seats:");
    for (int i = 0; i < totalRows; i++) {
        for (int j = 0; j < totalCols; j++) {
            if (seatAvailability[i][j]) {
                System.out.print(" " + (i + 1) + ", " + (j + 1) + " ] ");
            } else {
                System.out.print(" [ X ] ");
            }
        }
        System.out.println();
    }
}

public void buyTicket(int row, int col) {
    boolean validSeat = row >= 1 && row <= totalRows && col >= 1 && col <= totalCols &&
    seatAvailability[row - 1][col - 1];
    if (validSeat) {
        int price = seatPrices[row - 1][col - 1];
        System.out.println("You have purchased a ticket at row " + row + " and column " + col + "
for $" + price);
        seatAvailability[row - 1][col - 1] = false;
        seatsAvailable--;
    } else {
        System.out.println("Invalid seat selection or seat is already taken.");
    }
}
}

```

## TUGAS 2

Penulisan kode pada tugas 1 masih kurang benar, Refactor kode tersebut mengikuti instruksi di bawah ini:

1. Lakukan refactoring **Replace Magic Number with Symbolic Constant** pada harga seat yang konstan bernilai 50. Jadikan harga tersebut sebagai konstanta **DEFAULT\_PRICE** melalui refactoring ! (Clue: Introduce Constant)

```

private void initializeSeatPricesAndAvailability() {
    seatPrices = new int[totalRows][totalCols];
    seatAvailability = new boolean[totalRows][totalCols];

    for (int i = 0; i < totalRows; i++) {
        for (int j = 0; j < totalCols; j++) {
            seatPrices[i][j] = 50;
            seatAvailability[i][j] = true;
        }
    }
}

```

2. Buatlah method getter dari variabel global seatsAvailable dengan menggunakan refactoring **encapsulate field** ! (Clue: Method getter saja tidak perlu method setter)
3. Lakukan refactoring **Inline Variable** pada bagian di bawah ini! (Clue: Inline Variable)

```

public void buyTicket(int row, int col) {
    boolean validSeat = row >= 1 && row <= totalRows && col >= 1 && col <= totalCols && seatAvailability[row - 1][col - 1];
    if (validSeat) {
        int price = seatPrices[row - 1][col - 1];
        System.out.println("You have purchased a ticket at row " + row + " and column " + col + " for $" + price);
        seatAvailability[row - 1][col - 1] = false;
        seatsAvailable--;
    } else {
        System.out.println("Invalid seat selection or seat is already taken.");
    }
}

```

4. **Extract method** bagian yang diselect di bawah ini sehingga menjadi method baru dengan nama `isValidSeat()` ! (Clue: Extract Method)

```

public void buyTicket(int row, int col) {
    if (row >= 1 && row <= totalRows && col >= 1 && col <= totalCols && seatAvailability[row - 1][col - 1]) {
        int price = seatPrices[row - 1][col - 1];
        System.out.println("You have purchased a ticket at row " + row + " and column " + col + " for $" + price);
        seatAvailability[row - 1][col - 1] = false;
        seatsAvailable--;
    } else {
        System.out.println("Invalid seat selection or seat is already taken.");
    }
}

```

5. Buatlah Kelas baru dengan nama **Main** !  
 6. Pindahkan method main yang ada di kelas **ConcertTicketSystem** ke dalam kelas **Main** dengan menggunakan teknik refactoring ! (Clue: Move members)  
 7. Pada bagian method main di kelas Main, ubahlah bagian di bawah ini !

Before

```

        break;
    case '3':
        System.out.println("Available Seats Count: " + ticketSystem.seatAvailability);
        break;
    case '4':
        System.out.println("Exiting...");

```

After

```

    case '3':
        System.out.println("Available Seats Count: " + ticketSystem.getSeatsAvailable());
        break;
    case '4':
        System.out.println("Exiting...");

```

### TUGAS 3

Buatlah sebuah class **Employee**, lalu tulis ulang kode seperti di bawah ini !

```

package Modul2.Tugas.Tugas3.Jawaban;

public class Employee {
    private String en;
    private int empAge;
    private double salary;
    private String jd;

    public Employee(String empName, int age, double empSalary, String jobDescription) {
        en = empName;
        empAge = age;
        salary = empSalary;
        jd = jobDescription;
    }

    public void e() {
        System.out.println("Employee Name: " + en);
        System.out.println("Age: " + empAge);
        System.out.println("Salary: $" + salary);
        System.out.println("Job Description: " + jd);
    }

    public double c() {
        return salary * 12;
    }

    public void s(double raisePercentage) {
        salary += (salary * raisePercentage / 100);
    }
}

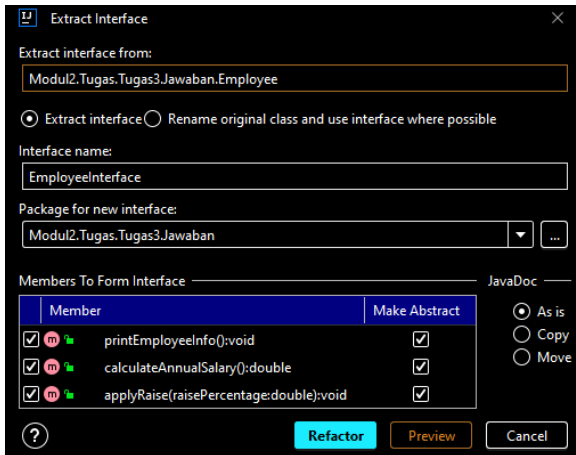
```

Penulisan kode di atas masih sulit untuk dipahami sehingga perlu dilakukan refactoring. Lakukanlah refactoring sesuai dengan petunjuk di bawah ini:

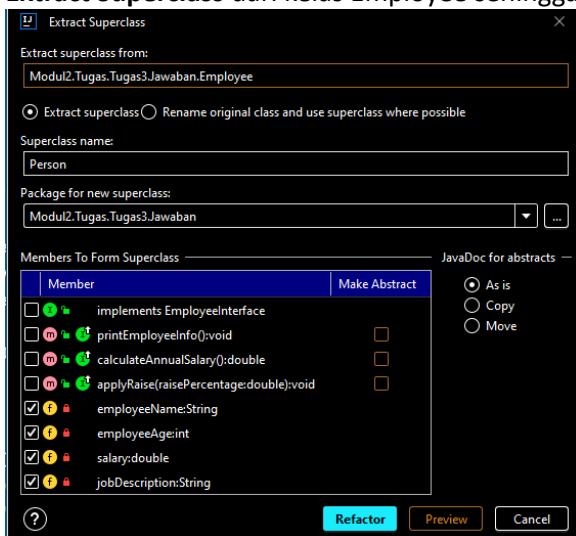
1. Lakukan teknik refactoring **Rename Method/Variable** pada penulisan variabel, parameter, dan method yang sulit dipahami. Untuk itu berikut adalah table penamaan yang benar:

PENAMAAN SEBELUM RAFACTORING	PENAMAAN SETELAH REFACTORING
private String en; private int empAge; private double salary; private String jd;	private String employeeName; private int employeeAge; private double salary; private String jobDescription;
public Employee(String empName, int age, double empSalary, String jobDescription)	public Employee(String employeeName, int employeeAge, double salary, String jobDescription)
e()	printEmployeeInfo()
c()	calculateAnnualSalary()
s(double raisePercentage)	applyRaise(double raisePercentage)

2. **Extract Interface** dari kelas Employee sehingga menjadi interface dengan nama **EmployeeInterface**. Dalam pengestrakan interface pastikan membuat semua method menjadi abstrak.

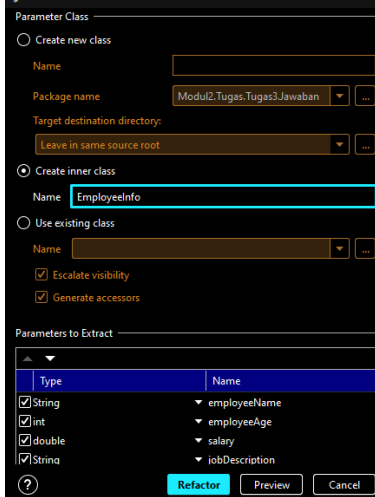


3. **Extract Superclass** dari kelas **Employee** sehingga menjadi super class dengan nama **Person**.



4. Lakukanlah refactoring **Introduce Parameter Object** pada bagian parameter dari Constructor Method **Employee**.

```
public Employee(String employeeName, int employeeAge, double salary, String jobDescription) {
    super(employeeName, employeeAge, salary, jobDescription);
}
```



5. **Note: refactor dilakukan secara live saat demo dengan asisten**



**KRITERIA & DETAIL PENILAIAN**

Kriteria Penilaian		Nilai
Latihan 1		15
Telah Mengimplementasikan Teknik Refactoring Rename Class	3	
Telah Mengimplementasikan Teknik Refactoring Replace Magic Number with Symbolic Constant	3	
Telah Mengimplementasikan Teknik Refactoring Inline Variable	3	
Telah Mengimplementasikan Teknik Refactoring Extract method	3	
Telah Mengimplementasikan Teknik Refactoring Encapsulate Field	3	
Latihan 2		5
Tugas 1		10
Tugas 2		20
Telah Mengimplementasikan Teknik Refactoring Replace Magic Number with Symbolic Constant	3	
Telah Mengimplementasikan Teknik Refactoring Encapsulate Field	3	
Telah Mengimplementasikan Teknik Refactoring Inline Variable	3	
Telah Mengimplementasikan Teknik Refactoring Extract method	4	
Telah Mengimplementasikan Teknik Refactoring Move method	4	
Program Tidak Error	3	20
Tugas 3		
Telah Mengimplementasikan Teknik Refactoring Rename Method/Variable	5	
Telah Mengimplementasikan Teknik Refactoring Extract Interface	5	
Telah Mengimplementasikan Teknik Refactoring Extract Superclass	5	
Telah Mengimplementasikan Teknik Refactoring Introduce Parameter Object	5	
Pemahaman		30
Total		100