

**LAPORAN PRAKTIKUM  
CODELAB PEMROGRAMAN LANJUT  
MODUL 2**



**Nama:** Natania Oktaviani

**NIM:** 202410370110272

**Kelas:** D

**PRODI INFORMATIKA  
FAKULTAS TEKNIK  
UNIVERSITAS MUHAMMADIYAH MALANG  
2025**

## CODELAB :

### Latar Belakang

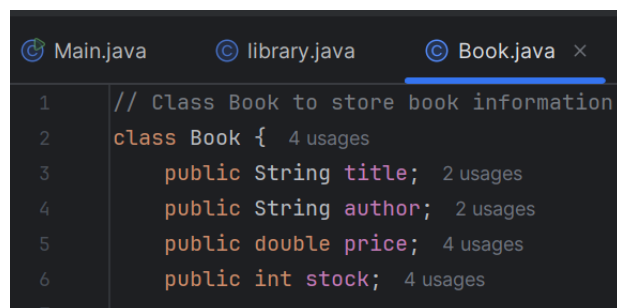
Pada percobaan Codelab Modul 2 ini, saya melakukan proses refactoring terhadap sebuah program yang berfungsi mengelola data buku dan perpustakaan. Program awal masih memiliki beberapa kelemahan, seperti akses langsung terhadap variabel (belum menerapkan prinsip encapsulation), penggunaan magic number tanpa makna yang jelas, serta logika program yang terlalu panjang dalam satu method.

Melalui penerapan teknik refactoring seperti Encapsulate Field, Introduce Constant, Extract Method, dan Move Method, struktur kode diperbaiki sehingga menjadi lebih rapi, terorganisasi, dan mudah dikelola. Hasil akhirnya tidak hanya membuat program berfungsi dengan baik, tetapi juga meningkatkan keterbacaan serta kualitas desain perangkat lunak secara keseluruhan.

### Langkah – Langkah Refactoring:

#### Langkah 1: Encapsulate Field

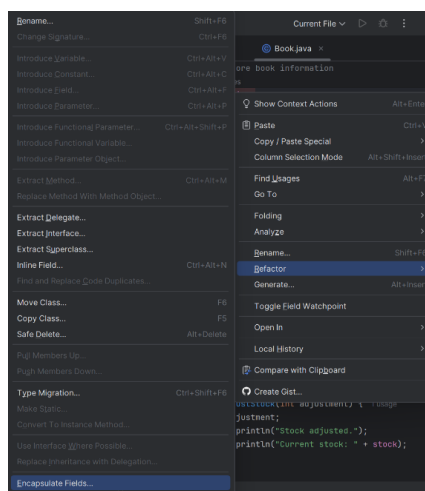
Langkah pertama, buka proyek Java milikmu menggunakan IntelliJ IDEA. Setelah itu, cari file Book.java pada bagian Project Explorer dan buka file tersebut. Sebelum memulai proses refactoring, pastikan kode di dalamnya masih merupakan versi awal tanpa perubahan.



```
1 // Class Book to store book information
2 class Book { 4 usages
3     public String title; 2 usages
4     public String author; 2 usages
5     public double price; 4 usages
6     public int stock; 4 usages
```

#### Langkah 2 – Pilih Menu Refactor

Pada tahap ini, buka kelas Book di IntelliJ IDEA. Arahkan kursor ke salah satu field seperti title atau author, kemudian klik kanan. Setelah itu, pilih opsi Refactor, dan lanjutkan dengan memilih Encapsulate Fields seperti yang ditunjukkan pada gambar kedua.

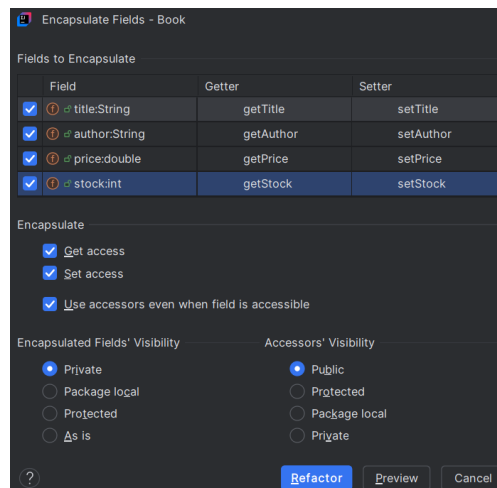


### Langkah 3 – Jendela “Encapsulate Fields – Book” Muncul

Akan muncul tampilan seperti pada gambar ketiga kamu, berjudul **Encapsulate Fields – Book**

Di sini ada beberapa pengaturan penting:

1. **Checklist Field yang mau dienkapsulasi:**  
Centang semuanya:  
title:String ,author:String ,price:double ,stock:int
2. **Getter dan Setter:**  
Pastikan kolomnya otomatis muncul seperti ini:  
-Getter → getTitle, getAuthor, getPrice, getStock  
=Setter → setTitle, setAuthor, setPrice, setStock
3. **Encapsulate Options (bagian bawah):**  
-Centang **Get access**  
-Centang **Set access**  
-Centang **Use accessors even when field is accessible**
4. **Encapsulated Fields' Visibility:** pilih **Private**
5. **Accessors' Visibility:** pilih **Public**



### Langkah 4 – Jalankan Refactor

Setelah jendela pengaturan muncul, klik tombol Refactor di bagian bawah. IntelliJ kemudian akan melakukan proses otomatis, yaitu mengubah semua field dari public menjadi private, membuat getter dan setter, serta memperbarui seluruh bagian kode yang sebelumnya mengakses variabel title, author, price, dan stock agar menggunakan metode get dan set..

## Kode Sebelum Encapsulate Fields:

```
Main.java  library.java  Book.java x
1 // Class Book to store book information
2 class Book { 4 usages
3     public String title; 2 usages
4     public String author; 2 usages
5     public double price; 4 usages
6     public int stock; 4 usages
7
8     // Constructor
9     Book(String title, String author, double price, int stock) { 1 usage
10         this.title = title;
11         this.author = author;
12         this.price = price;
13         this.stock = stock;
14     }
15
16     // Display book details
17     public void displayInfo() { 1 usage
18         System.out.println("Title: " + title);
19         System.out.println("Author: " + author);
20         System.out.println("Price: $" + price);
21         System.out.println("Discounted Price $" + (price - (price * 0.1)));
22         System.out.println("Stock: " + stock);
23     }
24
25     // Adjust the book stock
26     public void adjustStock(int adjustment) { 1 usage
27         stock += adjustment;
28         System.out.println("Stock adjusted.");
29         System.out.println("Current stock: " + stock);
30     }
31 }
```

## Kode Sesudah Encapsulate Fields:

```
Main.java  library.java  Book.java x
1 // Class Book to store book information
2 class Book { 4 usages
3     private String title; 2 usages
4     private String author; 2 usages
5     private double price; 2 usages
6     private int stock; 2 usages
7
8     // Constructor
9     Book(String title, String author, double price, int stock) { 1 usage
10         this.setTitle(title);
11         this.setAuthor(author);
12         this.setPrice(price);
13         this.setStock(stock);
14     }
15
16     // Display book details
17     public void displayInfo() { 1 usage
18         System.out.println("Title: " + getTitle());
19         System.out.println("Author: " + getAuthor());
20         System.out.println("Price: $" + getPrice());
21         System.out.println("Discounted Price $" + (getPrice() - (getPrice() * 0.1)));
22         System.out.println("Stock: " + getStock());
23     }
24
25     // Adjust the book stock
26     public void adjustStock(int adjustment) { 1 usage
27         setStock(getStock() + adjustment);
28         System.out.println("Stock adjusted.");
29         System.out.println("Current stock: " + getStock());
30     }
31
32     public String getTitle() { 1 usage
33         return title;
34     }
35
36     public void setTitle(String title) { 1 usage
37         this.title = title;
38     }
39
40     public String getAuthor() { 1 usage
```

```

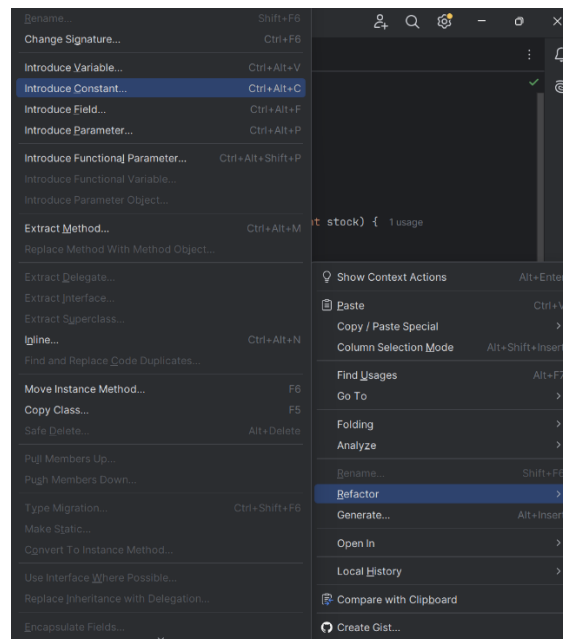
59
60     public void setStock(int stock) { 2 usages
61         this.stock = stock;
62     }
63 }
64

```

## Langkah 2: Introduce Constant

### Langkah – Langkah

Langkah berikutnya, pilih nilai 0.1 di dalam kode yang digunakan untuk menghitung diskon. Klik kanan pada nilai tersebut, lalu pilih Refactor → Introduce Constant.... Setelah pop-up muncul, beri nama konstanta DISCOUNT\_RATE, kemudian tekan Enter. IntelliJ akan otomatis membuat baris deklarasi konstanta seperti:



dan nilai 0.1 pada kode akan diganti menjadi DISCOUNT\_RATE

```

public static final double DISCOUNT_RATE = 0.1;
☐ Move to another class
price();
$" + (getPrice() - (getPrice() * DISCOUNT_RATE)))

```

Kode sebelum Introduce Constant:

```

21     System.out.println("Discounted Price $" + (price - (price * 0.1)));

```

Kode Sesudah Introduce Constant:

```

1 // Class Book to store book information
2 class Book { 4 usages
3     public static final double DISCOUNT_RATE = 0.1; 1 usage

```

### Perbedaan Sebelum dan Sesudah:

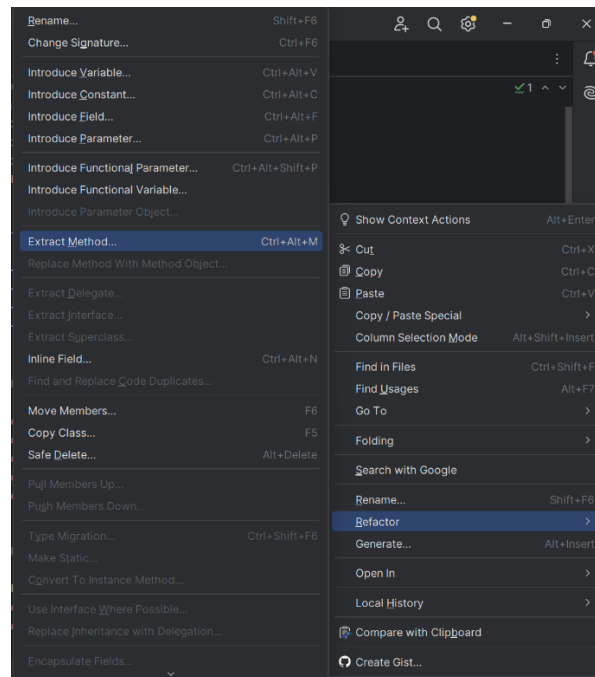
Sebelum dilakukan refactoring, nilai 0.1 ditulis langsung di dalam perhitungan, sehingga sulit dipahami apa maksud angka tersebut. Setelah dilakukan *Introduce Constant*, angka tersebut diganti menjadi DISCOUNT\_RATE, sehingga kode lebih jelas, mudah dibaca, dan mudah diubah jika nanti nilai diskonnya

ingin diganti.

### Langkah 3: Extract Method

Langkah – Langkah

Langkah berikutnya, pilih kode perhitungan diskon di dalam method `displayInfo()`. Setelah itu, klik kanan dan pilih Refactor, kemudian pilih Extract Method.... IntelliJ akan menampilkan jendela untuk menamai method baru. Beri nama `calculateDiscount`, lalu tekan Enter. Secara otomatis, IntelliJ akan membuat method baru yang memisahkan logika perhitungan diskon dari method utama.



Kode Sebelum Extract Me

```
public void displayInfo() { 1 usage new *
    System.out.println("Title: " + title);
    System.out.println("Author: " + author);
    System.out.println("Price: $" + price);
    System.out.println("Discounted Price $" + (price - (price * DISCOUNT_RATE)));
    System.out.println("Stock: " + stock);
}

public void adjustStock(int adjustment) { 1 usage new *
    stock += adjustment;
    System.out.println("Stock adjusted.");
    System.out.println("Current stock: " + stock);
}
```

## Kode Sesudah Extract Me

```
17 // Display book details
18 public void displayInfo() { 1 usage
19     System.out.println("Title: " + getTitle());
20     System.out.println("Author: " + getAuthor());
21     System.out.println("Price: $" + getPrice());
22     System.out.println("Discounted Price $" + (getPrice() - calculateDiscount()));
23     System.out.println("Stock: " + getStock());
24 }
25
26 private double calculateDiscount() { 1 usage
27     return getPrice() * DISCOUNT_RATE;
28 }
```

### Perbedaan Sebelum dan Sesudah (Extract Method)

Sebelum dilakukan *Extract Method*, seluruh logika perhitungan diskon ditulis langsung di dalam method `displayInfo()`. Hal ini membuat kode menjadi panjang dan sulit dibaca karena satu method menampung terlalu banyak tugas sekaligus — baik untuk menampilkan data maupun menghitung diskon.

Setelah dilakukan refactoring dengan teknik *Extract Method*, bagian perhitungan diskon dipisahkan ke dalam method baru bernama `calculateDiscount()`. Dengan pemisahan ini:

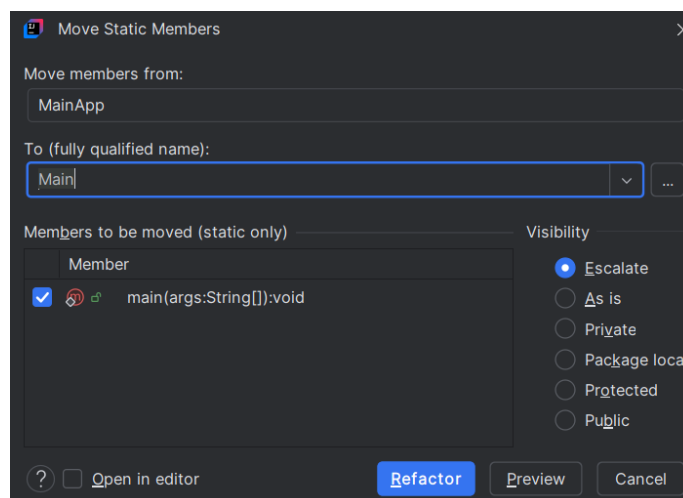
- Struktur kode menjadi lebih **rapi dan terorganisir**,
- Setiap method memiliki **tanggung jawab yang jelas**,
- Program lebih **mudah dipahami dan diperbaiki** jika nanti rumus diskon perlu diubah.

Hasilnya, `displayInfo()` hanya berfungsi untuk menampilkan informasi buku, sementara `calculateDiscount()` khusus menangani perhitungan diskon.

### Langkah 4: Move Method

Langkah – Langkah

1. Buat Java Class baru dengan nama Main.
2. Blok method main dan klik kanan lalu pilih menu Refactor.
3. Pilih menu Move members lalu akan muncul sebagai berikut:



4. Setelah muncul box Move Members, kita hanya harus memasukkan nama Main dan tekan Refactor.

Setelah itu semua yang ada pada MainApp akan berpindah ke Main.