

**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 diminta untuk melakukan proses refactoring pada sebuah program yang berfungsi untuk mengelola data buku dan perpustakaan. Program awal memiliki beberapa kelemahan seperti penggunaan variabel yang masih dapat diakses secara langsung (belum menerapkan *encapsulation*), adanya *magic number* tanpa makna yang jelas, serta adanya logika program yang terlalu panjang dalam satu *method*.

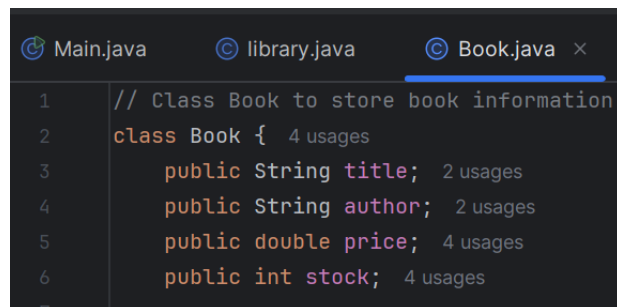
Melalui penerapan beberapa teknik refactoring yaitu Encapsulate Field, Introduce Constant, Extract Method, dan Move Method, kode diperbaiki agar menjadi lebih bersih, terstruktur, dan mudah dipelihara. Hasil akhir dari refactoring ini bukan hanya membuat kode berjalan dengan baik, tetapi juga meningkatkan keterbacaan dan kualitas desain program secara keseluruhan.

### Langkah – Langkah Refactoring:

#### Langkah 1: Encapsulate Field

##### Langkah-langkah:

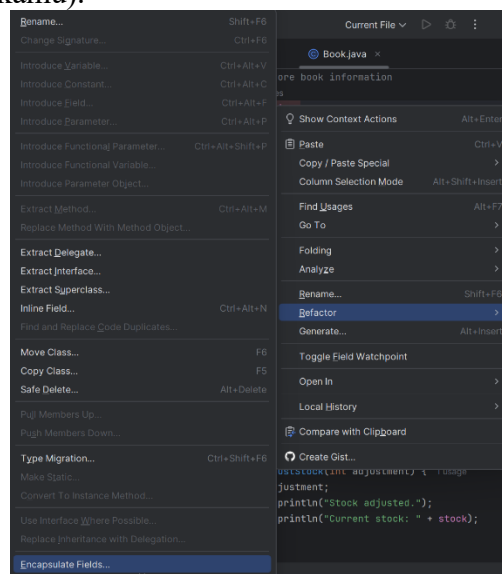
1. Buka proyek Java kamu di IntelliJ IDEA.
2. Buka tab file Book.java.
3. Pastikan isi awalnya masih seperti ini (belum di-refactor):



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

#### Langkah 2 – Pilih Menu Refactor

1. Klik kanan di dalam area kode kelas Book (boleh di atas salah satu field seperti title atau author).
2. Pilih menu **Refactor** → **Encapsulate Fields**.  
(seperti di gambar kedua kamu).



### 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:

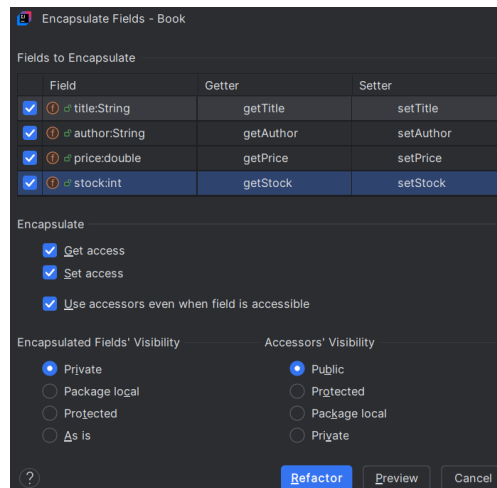
- Getter → getTitle, getAuthor, getPrice, getStock
- Setter → setTitle, setAuthor, setPrice, setStock

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

Klik tombol **Refactor** di bagian bawah jendela.

- Setelah itu IntelliJ otomatis akan: Mengubah field-field public menjadi private
- Membuatkan getter dan setter secara otomatis
- Mengganti semua pemanggilan langsung title, author, price, stock menjadi getTitle(), setTitle(), dan sebagainya.

Kode Sebelum Encapsulate Fields:

```
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 }
```

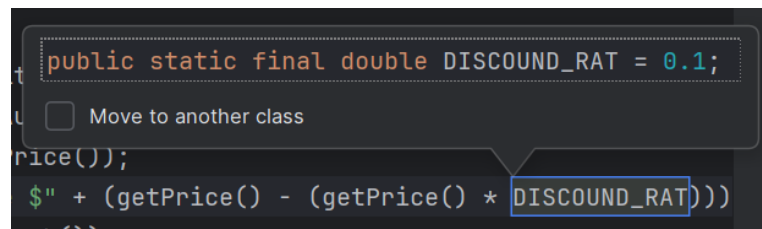
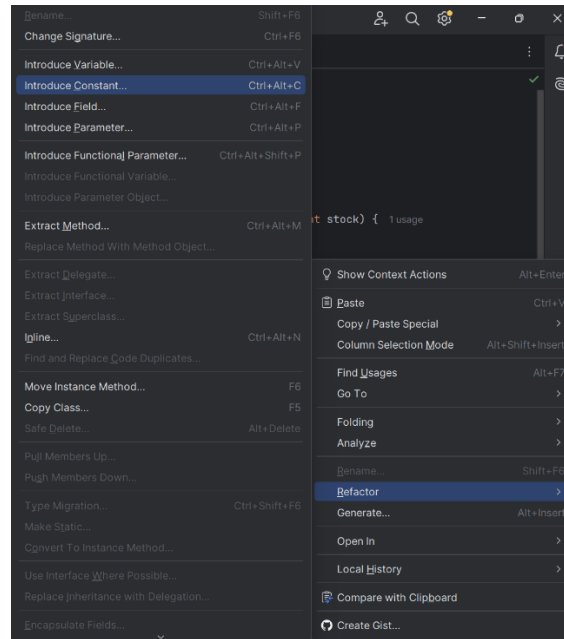
## Kode Sesudah Encapsulate Fields:

```
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
41         return author;
42     }
43
44     public void setAuthor(String author) { 1 usage
45         this.author = author;
46     }
47
48     public double getPrice() { 3 usages
49         return price;
50     }
51
52     public void setPrice(double price) { 1 usage
53         this.price = price;
54     }
55
56     public int getStock() { 3 usages
57         return stock;
58     }
59
60     public void setStock(int stock) { 2 usages
61         this.stock = stock;
62     }
63 }
64
```

## Langkah 2: Introduce Constant

### Langkah – Langkah

1. Blok nilai **0.1** pada kode yang ingin dijadikan konstanta.
2. Klik kanan, pilih **Refactor** → **Introduce Constant...** seperti pada gambar.
3. IntelliJ akan menampilkan pop-up untuk membuat konstanta baru.
4. Ganti nama konstanta menjadi **DISCOUNT\_RATE**.
5. Tekan **Enter**, maka otomatis akan muncul baris:



dan nilai 0.1 pada kode akan diganti menjadi 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
```

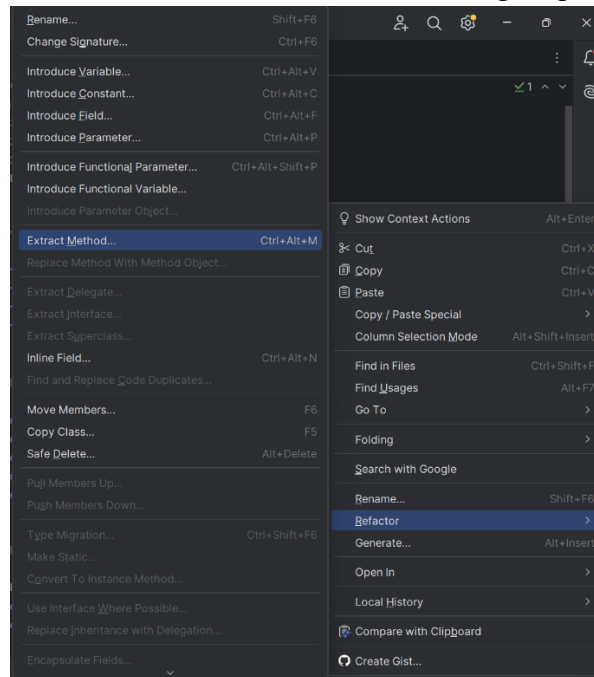
### Perbedaan Sebelum dan Sesudah:

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

1. Blok bagian kode yang berisi perhitungan harga diskon di dalam method `displayInfo()`.
2. Klik kanan pada bagian kode tersebut → pilih **Refactor** → **Extract Method...** seperti terlihat pada gambar.
3. Akan muncul jendela pop-up untuk memberi nama method baru.
4. Ganti namanya menjadi **calculateDiscount** agar sesuai dengan fungsinya.
5. Tekan **Enter**, dan IntelliJ otomatis membuat method baru berisi logika perhitungan tersebut.



Kode Sebelum Extract Method:

```
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 Method:

```
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:

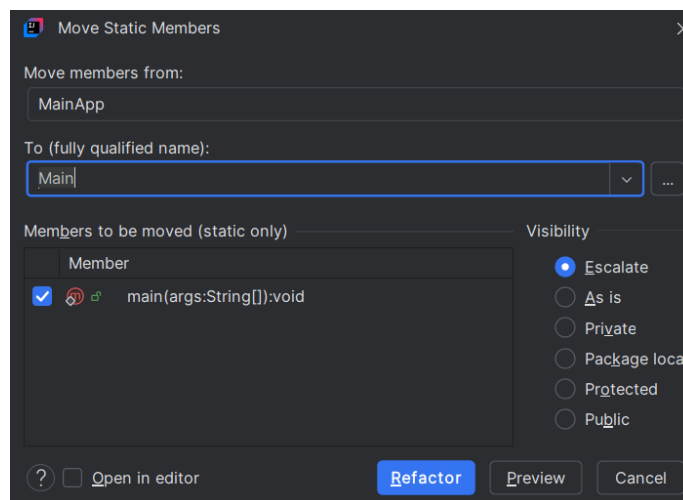
- Struktur kode menjadi lebih **rapi dan terorganisir**,
- Setiap method memiliki **tanggung jawab yang jelas**,
- Program lebih **mudah dipahami dan diperbaiki** jika nanti rumus diskon perlu diubah.

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.