

LAPORAN CODELAB 1 PEMROGRAMAN LANJUT

Nama: Naufal Arkaan

Nim: 202410370110020

Mata Kuliah: Praktikum Pemrograman Lanjut B

1. Judul

Refactoring Program Pengelolaan Data Buku dan Perpustakaan

2. Tujuan

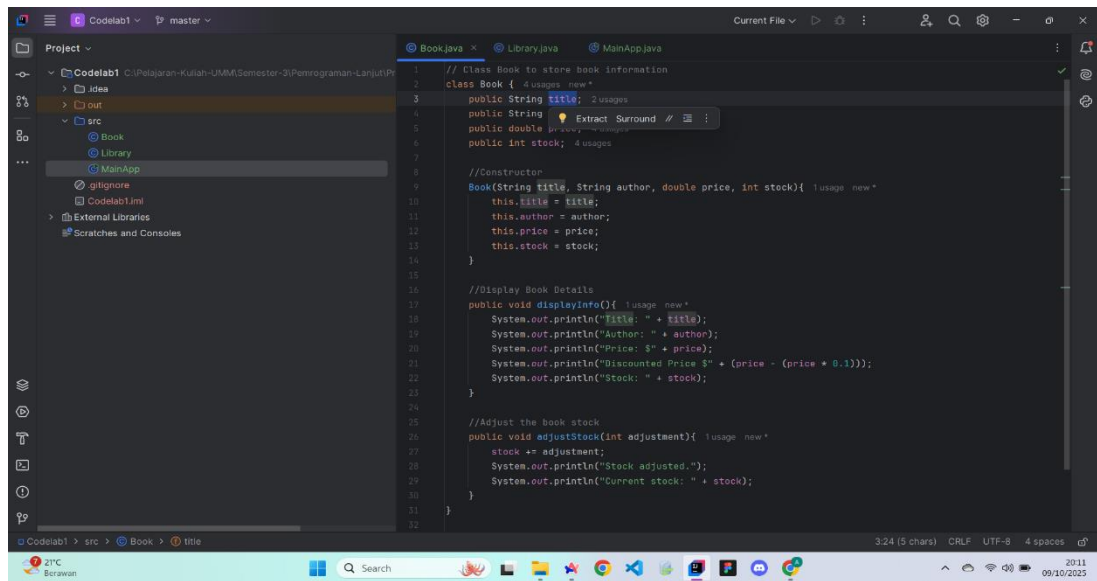
Mempraktikkan penerapan prinsip **refactoring** pada pemrograman berorientasi objek menggunakan Java untuk meningkatkan keterbacaan, efisiensi, dan struktur kode.

3. Dasar Teori

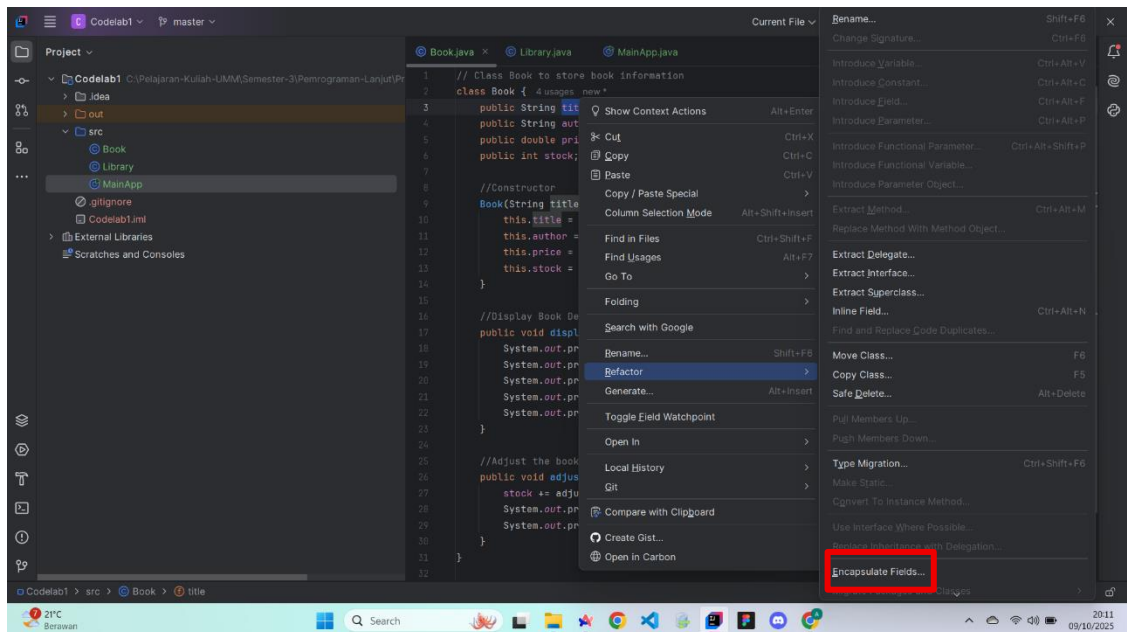
- **Encapsulation:** Melindungi data agar hanya bisa diakses melalui metode khusus (getter dan setter).
- **Constant:** Nilai yang tetap selama program berjalan, dideklarasikan dengan final.
- **Method Extraction:** Pemisahan bagian logika agar lebih modular dan mudah diuji.
- **Refactoring:** Proses memperbaiki struktur internal kode tanpa mengubah perilaku eksternal program.

4. Langkah-Langkah

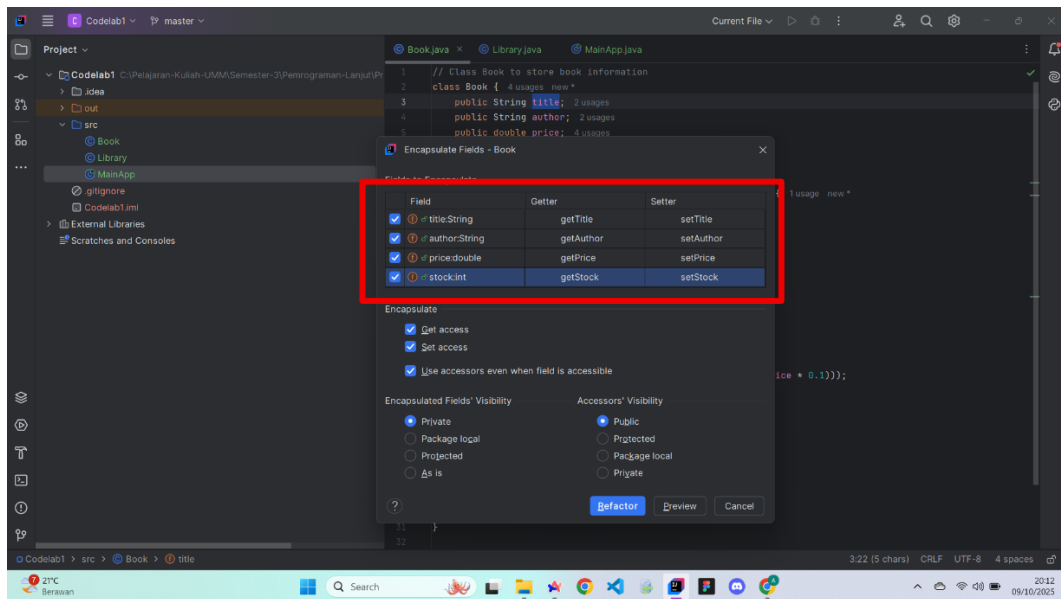
1. Menambahkan getter dan setter untuk setiap atribut pada class **Book** dan **Library**.
 - Langkah awal block code/atribut yang akan dilakukan refactoring pada class **Book**



- Setelah itu klik kanan pada mousepad, pilih refactor dan pilih Encapsulate Field



- Selanjutnya centang atribut yang ingin di Encapsulate Field



- Berikut hasil code setelah di refactoring dan sebelum di refactoring pada class **Book**:
- Sebelum:**

```
// Class Book to store book information
class Book {
    public String title;
    public String author;
    public double price;
    public int stock;

    //Constructor
    Book(String title, String author, double price, int stock){
        this.title = title;
        this.author = author;
        this.price = price;
        this.stock = stock;
    }

    //Display Book Details
    public void displayInfo(){
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
        System.out.println("Discounted Price $" + (price - (price * 0.1)));
        System.out.println("Stock: " + stock);
    }

    //Adjust the book stock
    public void adjustStock(int adjustment){
        stock += adjustment;
        System.out.println("Stock adjusted.");
        System.out.println("Current stock: " + stock);
    }
}
```

Sesudah:

```
// Class Book to store book information
class Book {
    private String title;
    private String author;
    private double price;
    private int stock;

    //Constructor
    Book(String title, String author, double price, int stock){
        this.setTitle(title);
        this.setAuthor(author);
        this.setPrice(price);
        this.setStock(stock);
    }

    //Display Book Details
    public void displayInfo(){
        System.out.println("Title: " + getTitle());
        System.out.println("Author: " + getAuthor());
        System.out.println("Price: $" + getPrice());
        System.out.println("Discounted Price $" + (getPrice() - (getPrice() *
0.1)));
        System.out.println("Stock: " + getStock());
    }

    //Adjust the book stock
    public void adjustStock(int adjustment){
        setStock(getStock() + adjustment);
        System.out.println("Stock adjusted.");
        System.out.println("Current stock: " + getStock());
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }
}
```

- Pada class **Library** kita bisa melakukan refactoring Encapsulate Field dengan cara yang sama dengan class **Book**, Saya akan tunjukkan code sebelum di refactoring dan sesudah di refactoring:

Sebelum:

```

//Class Library to Store Library location and a book
public class Library {
    public Book book;
    public String location;

    public Library(Book book, String location){
        this.book = book;
        this.location = location;
    }

    //Display Library and Book Information
    public void showLibraryInfo(){
        System.out.println("Library location: " + location);
        book.displayInfo();
    }
}

```

carbon

Sesudah:

```

//Class Library to Store Library location and a book
public class Library {
    private Book book;
    private String location;

    public Library(Book book, String location){
        this.setBook(book);
        this.setLocation(location);
    }

    //Display Library and Book Information
    public void showLibraryInfo(){
        System.out.println("Library location: " +
        getLocation().displayInfo());
    }

    public Book getBook() {
        return book;
    }

    public void setBook(Book book) {
        this.book = book;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }
}

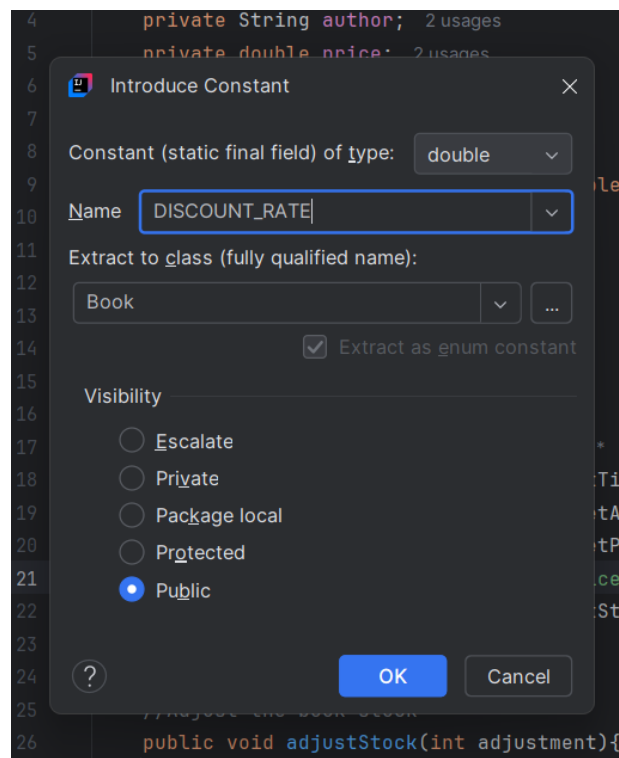
```

carbon

setter untuk field book dan location pada Class **Library** otomatis langsung ditambahkan, saya menggunakan extension tambahan yaitu **carbon** agar code bisa di upload secara jelas dan lengkap.

2. Menambahkan konstanta **DISCOUNT_RATE = 0.1** di class **Book**.

- Cara melakukan refactor introduce Constant yaitu Pilih salah satu nilai atau value yang ingin dijadikan sebuah konstanta, Klik kanan → pilih Refactor → lalu Introduce Constant, Beri nama konstanta.



- Berikut ada code sebelum dan sesudah di refactoring pada class **Book**:

Sebelum:

```
Book.java x Library.java MainApp.java
1 // Class Book to store book information
2 class Book { 4 usages new*
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 new*
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 new*
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 }
```

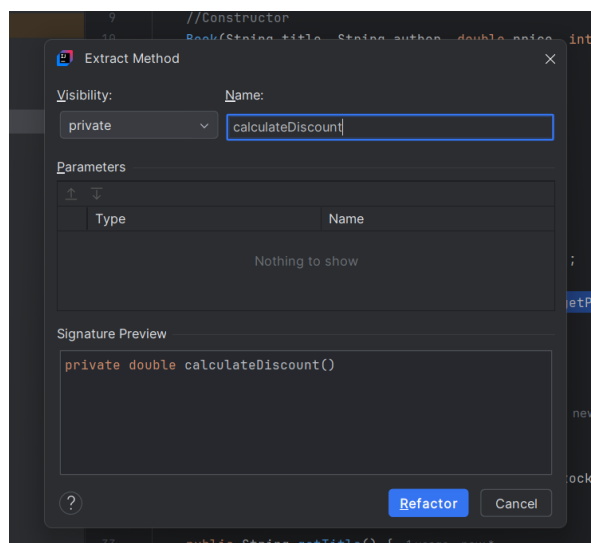
Sesudah:

```
Book.java x Library.java MainApp.java
2 class Book { 6 usages new *
3     public static final double DISCOUNT_RATE = 0.1; 1 usage
4     private String title; 2 usages
5     private String author; 2 usages
6     private double price; 2 usages
7     private int stock; 2 usages
8
9     //Constructor
10    Book(String title, String author, double price, int stock){ 1 usage new *
11        this.setTitle(title);
12        this.setAuthor(author);
13        this.setPrice(price);
14        this.setStock(stock);
15    }
16
17    //Display Book Details
18    public void displayInfo(){ 1 usage new *
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() - (getPrice() * DISCOUNT_RATE)));
23        System.out.println("Stock: " + getStock());
24    }
```

Code sudah otomatis berubah jadi constant/nilai tetap yang ga bisa diubah lagi, sehingga kita lebih mudah menggunakannya berulang kali.

3. Memisahkan perhitungan diskon dari displayInfo() menjadi metode baru calculateDiscount().

- Cara melakukan extract method adalah Blok bagian kode yang ingin dipisahkan, Klik kanan pada kode tersebut, pilih Refactor → Extract Method, Berikan nama metode sesuai dengan fungsi atau tujuan dari kode tersebut.



- Berikut saya akan berikan code sebelum dan sesudah di refactor:

Sebelum:

```
//Display Book Details
public void displayInfo(){ 1usage new *
    System.out.println("Title: " + getTitle());
    System.out.println("Author: " + getAuthor());
    System.out.println("Price: $" + getPrice());
    System.out.println("Discounted Price $" + (getPrice() - (getPrice() * DISCOUNT_RATE)));
    System.out.println("Stock: " + getStock())
}
```

Sesudah:

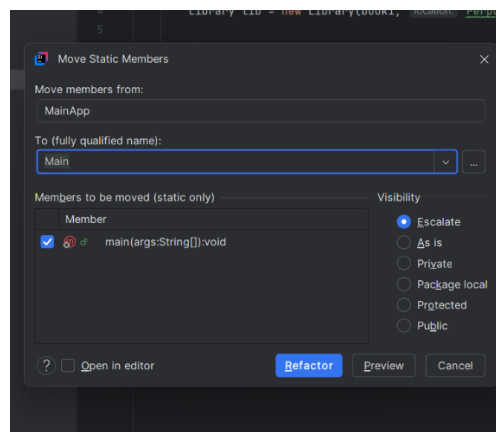
```
//Display Book Details
public void displayInfo(){ 1usage new *
    System.out.println("Title: " + getTitle());
    System.out.println("Author: " + getAuthor());
    System.out.println("Price: $" + getPrice());
    System.out.println("Discounted Price $" + calculateDiscount());
    System.out.println("Stock: " + getStock())
}

private double calculateDiscount() { 1usage new *
    return getPrice() - (getPrice() * DISCOUNT_RATE);
}
```

Secara otomatis akan generate sebuah method baru yaitu **calculateDiscount()**.

4. Memindahkan method main() dari class MainApp ke class baru Main.

- Cara melakukan move method/move members adalah Blok bagian kode/method yang ingin dipindahkan, Klik kanan pada kode tersebut, pilih Refactor → Move Members, Sebelum itu buatlah class Main terlebih dahulu, setelah class main dibuat maka pilihlah class Main sebagai tempat code yang akan dipindahkan.



- Berikut saya akan berikan code sebelum dan sesudah di refactor:

Sebelum:

```

1 public class MainApp { new *
2     public static void main(String[] args) { new *
3         Book book1 = new Book("Harry Potter", "J.K Rowling", price: 10, stock: 2);
4         Library lib = new Library(book1, location: "Perpustakaan Kota");
5
6         //Display Initial Information
7         lib.showLibraryInfo();
8
9         //Add More Stock
10        book1.adjustStock(adjustment: 5);
11
12        //Display updated information
13        lib.showLibraryInfo();
14    }
15 }
16

```

Sesudah:

```

Main.java
1 public class Main { new *
2     public static void main(String[] args) { new *
3         Book book1 = new Book("Harry Potter", "J.K Rowling", price: 10, stock: 2);
4         Library lib = new Library(book1, location: "Perpustakaan Kota");
5
6         //Display Initial Information
7         lib.showLibraryInfo();
8
9         //Add More Stock
10        book1.adjustStock(adjustment: 5);
11
12        //Display updated information
13        lib.showLibraryInfo();
14    }
15 }

MainApp.java
1 public class MainApp { no usages new *
2     }
3

```

Method **main()** yang semula di class **MainApp** pindah ke class **Main**.

5. Hasil Program (Output)

```
Library location: Perpustakaan Kota  
Title: Harry Potter  
Author: J.K Rowling  
Price: $10.0  
Discounted Price $9.0  
Stock: 2  
Stock adjusted.  
Current stock: 7  
Library location: Perpustakaan Kota  
Title: Harry Potter  
Author: J.K Rowling  
Price: $10.0  
Discounted Price $9.0  
Stock: 7  
  
Process finished with exit code 0
```

6. Kesimpulan

Setelah dilakukan refactoring, kode menjadi:

- Lebih mudah dibaca dan dikelola,
- Mengikuti prinsip OOP (Encapsulation, Abstraction),
- Modular dan siap dikembangkan lebih lanjut tanpa mengubah struktur utama.