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

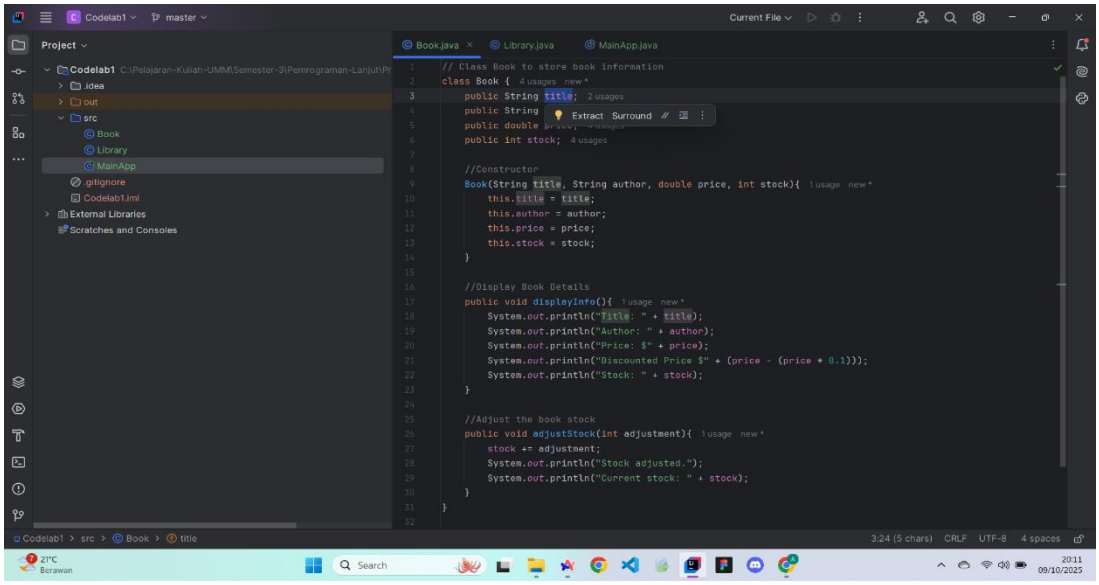
Mempraktikkan penerapan prinsip **refactoring** pada pemrograman berorientasi objek menggunakan Java untuk meningkatkan keterbacaan, efisiensi, dan struktur kode.

3. Dasar Teori

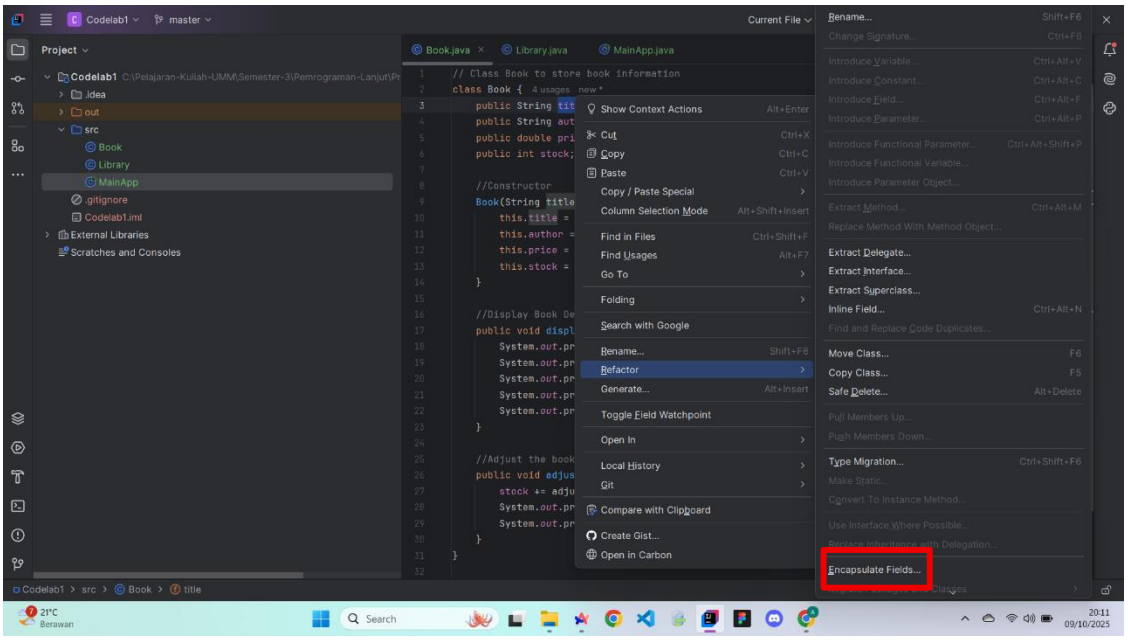
- **Encapsulation**: Melindungi data agar hanya bisa diakses melalui metode khusus (getter dan setter).
- **Constant**: Nilai yang tetap selama program berjalan, dideklarasikan dengan final.
- **Method Extraction**: Pemisahan bagian logika agar lebih modular dan mudah diuji.
- **Refactoring**: Proses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

4. Langkah-Langkah

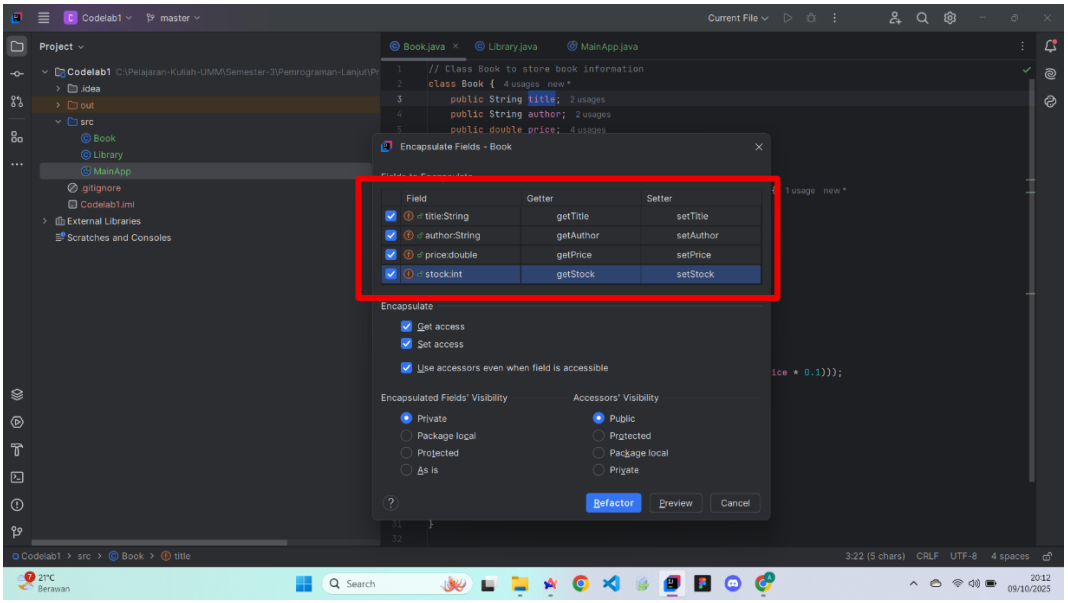
1. **Menambahkan getter dan setter untuk setiap atribut pada class Book dan Library.**
    - Langkah awal block code/atribut yang akan dilakukan refactoring pada class **Book**



- 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:

```java
// 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:

```java
// 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 isa melakukan refactoring Encapsulate Field dengan cara yang sama dengan class **Book**, Saya akan tunjukan code sebelum di refactoring dan sesudah di refactoring:

  Sebelum:

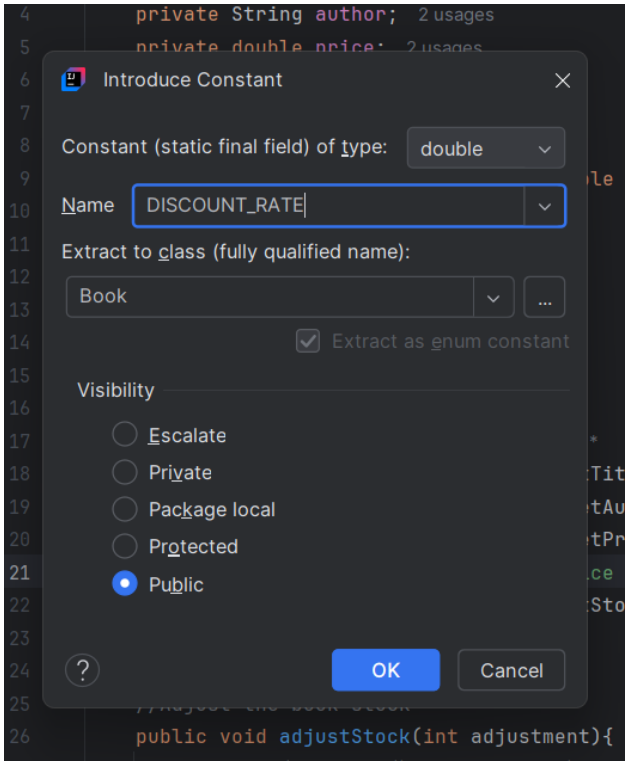
```java
//Class Library to Store Library location and a book
public class Library {
    public Book book;
    public String location;

    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();
    }
}
```

Sesudah:

```java
//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());book().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;
    }
}
```

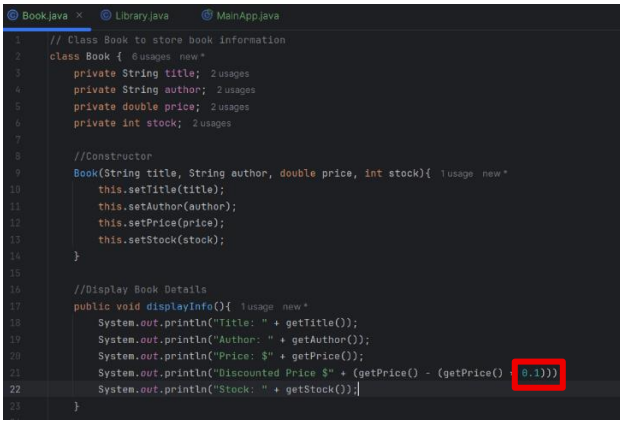
setter untuk field book dan location pada Class **Library** otomatis langsung ditambahkan, saya menggunakan exstension tambahan yaitu **carbon** agar code bisa di upload secara jelas dan lengkap.

2. **Menambahkan konstanta DISCOUNT_RATE = 0.1 di class Book.**
   - Cara melakukan refactor introduce Constant yaitu Pilih salah satu nilai atau value yang ingin dijadian sebuah konstanta, Klik kanan → pilih Refactor → lalu Introduce Constant, Beri nama konstanta.

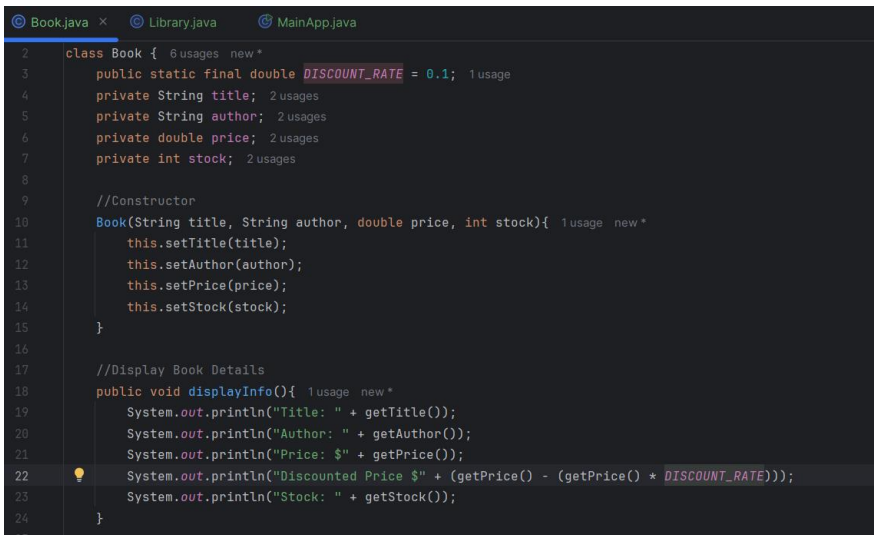


   - Berikut ada code sebelum dan sesudah di refactoring pada class Book:

     Sebelum:

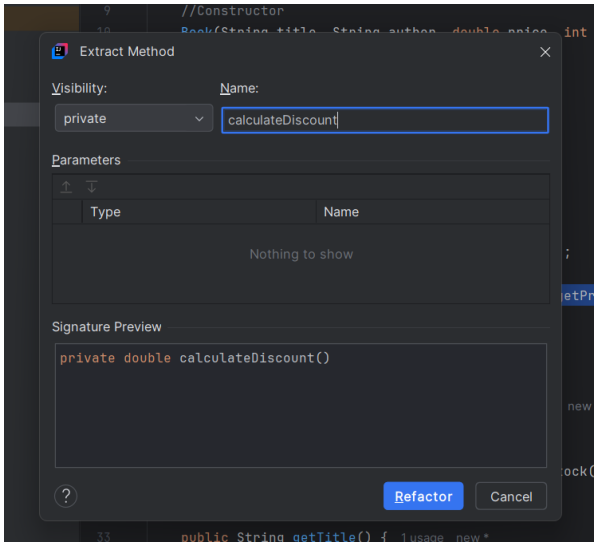


Sesudah:



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(){  1 usage  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())
}
```

Extract Surround // ⋮

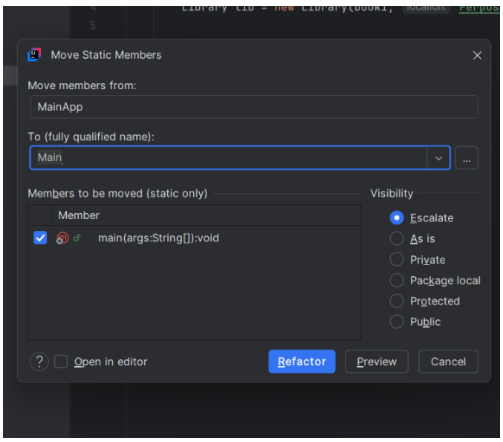
Sesudah:

```
//Display Book Details
public void displayInfo(){  1 usage  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() {  1 usage  new *
    return getPrice() - (getPrice() * DISCOUNT_RATE);
}
```

Extract Surround // ⋮

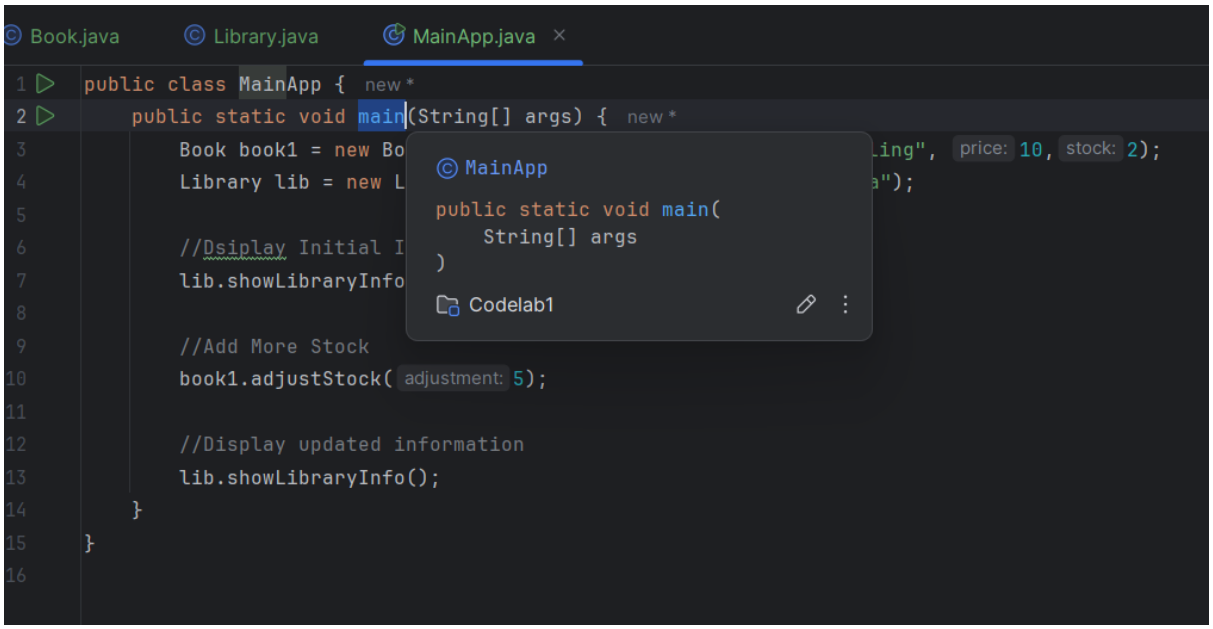
Secara otomatis akan generate sebuah method baru yaitu **calculateDiscount()**.

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 buatkan class Main terlebih dahulu, setelah class main dibuat maka pilihlah class Main sebagai tempat code yang akan dipindahkan.

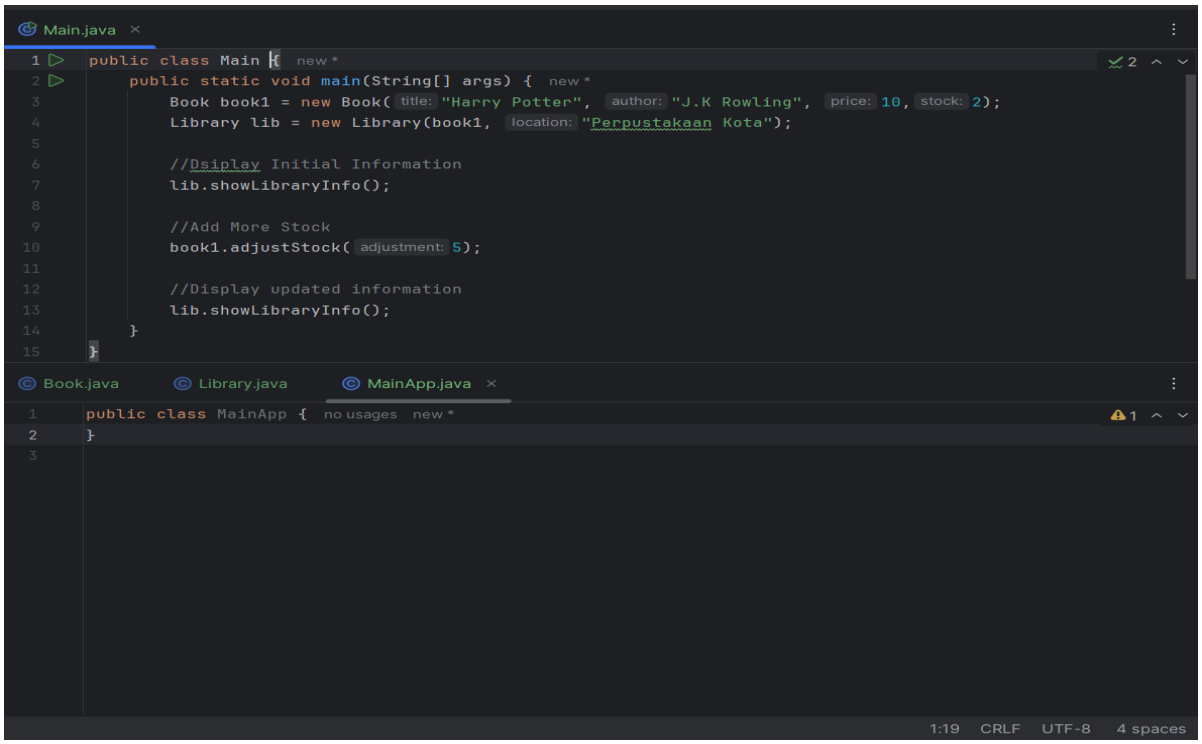


- Berikut saya akan berikan code sebelum dan sesudah di refactor:

  Sebelum:

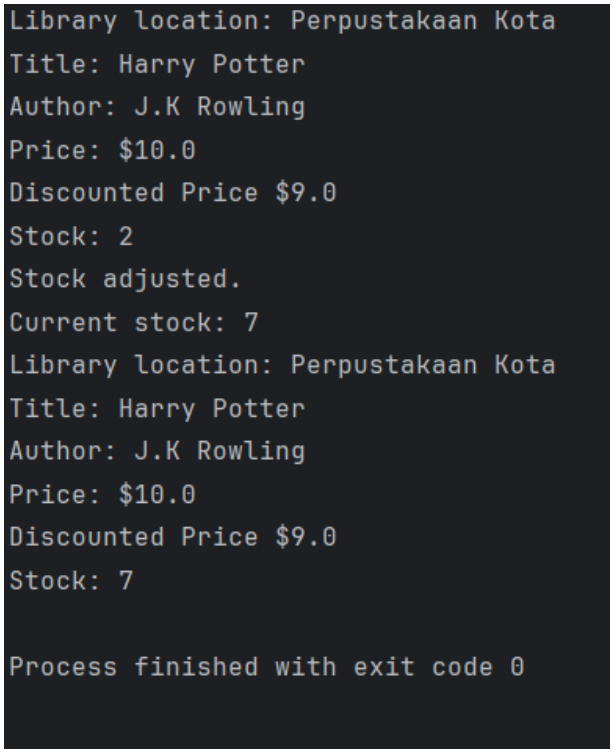


  Sesudah:



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

5. Hasil Program (Output)



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.