LAPORAN CODELAB 1 PEMROGRAMAN LANJUT

Nama : Muhamad Robi Ardita

NIM : 202410370110002

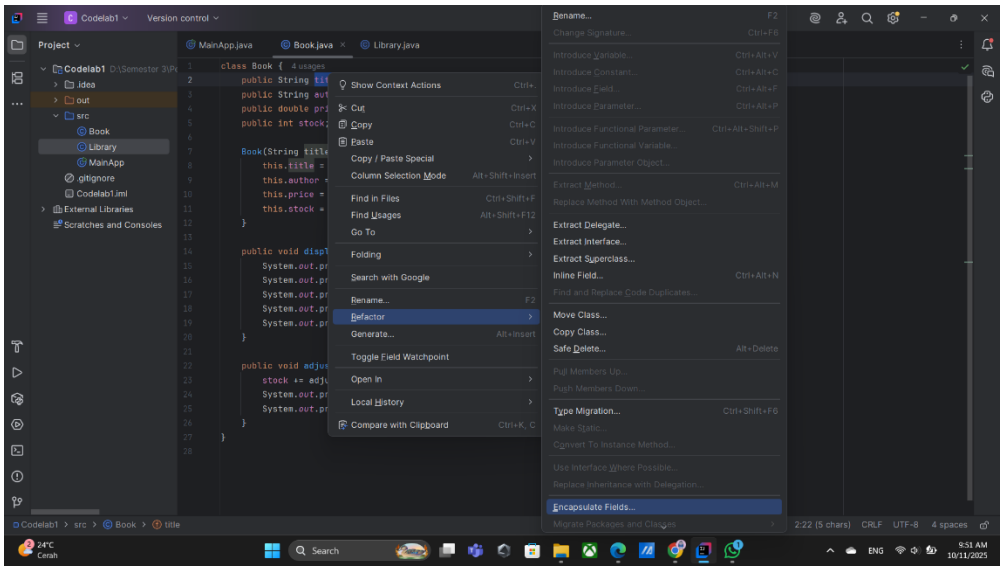
Matkul : Praktikum Pemrograman Lanjut B

Tujuan

Menerapkan konsep *refactoring* dalam pemrograman berorientasi objek menggunakan bahasa Java guna meningkatkan kualitas kode, meliputi aspek keterbacaan, efisiensi, serta kemudahan dalam pemeliharaan program.

Langkah – Langkah

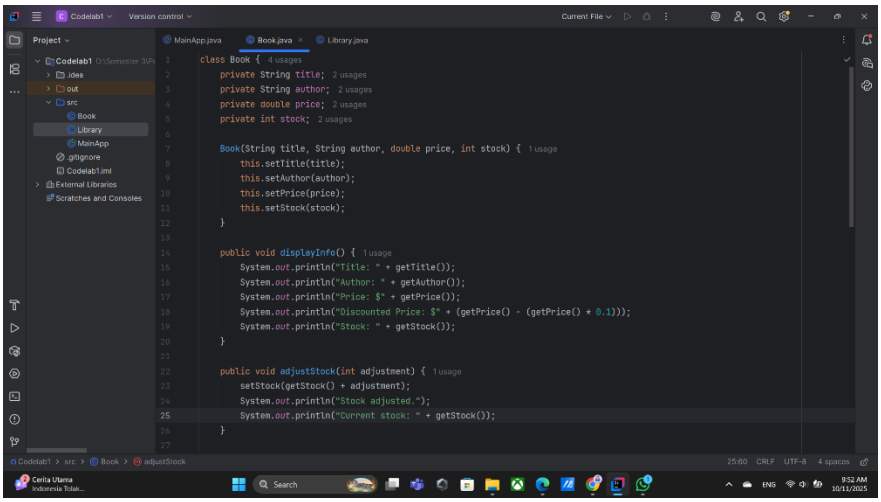
1. Refactor Book
   Pertama, kita melakukan refactor pada class Book untuk menambahkan *setter* dan *getter* di setiap atribut, yaitu title, author, price, dan stock.



   Tujuannya supaya data tidak bisa diubah langsung dari luar kelas, tapi lewat metode khusus agar lebih aman dan rapi.
   Langkah ini dilakukan dengan memilih menu Refactor → Encapsulate Fields di IntelliJ IDEA dan jangan lupa untuk centang atribut yang ingin di Encapsulate Field

   Fitur ini otomatis membuat *getter* dan *setter* untuk setiap atribut tanpa perlu mengetik manual.

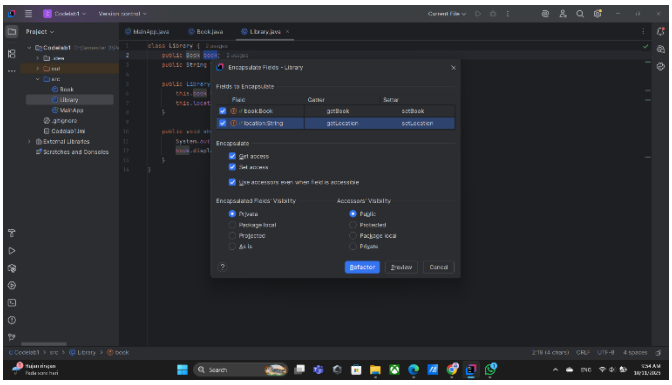
Sebelum:



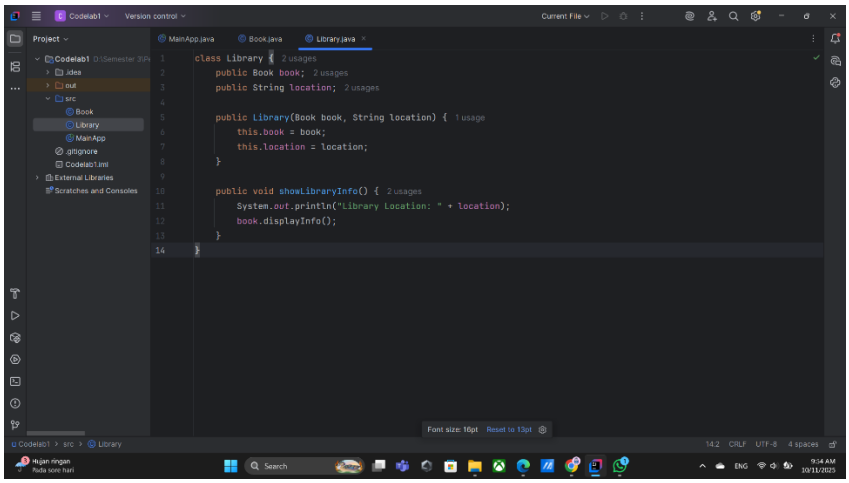
```
class Book {  4 usages
    private String title;  2 usages
    private String author;  2 usages
    private double price;  2 usages
    private int stock;  2 usages

    Book(String title, String author, double price, int stock) {  1 usage
        this.setTitle(title);
        this.setAuthor(author);
        this.setPrice(price);
        this.setStock(stock);
    }

    public void displayInfo() {  1 usage
        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());
    }

    public void adjustStock(int adjustment) {  1 usage
        setStock(getStock() + adjustment);
        System.out.println("Stock adjusted.");
        System.out.println("Current stock: " + getStock());
    }

    public String getTitle() {  1 usage
        return title;
    }

    public void setTitle(String title) {  1 usage
        this.title = title;
    }

    public String getAuthor() {  1 usage
        return author;
    }

    public void setAuthor(String author) {  1 usage
        this.author = author;
    }

    public double getPrice() {  3 usages
        return price;
    }
```

2. Refactor Library

   Langkah berikutnya kita melakukan refactor pada class Library. Di sini kita juga menambahkan *getter* dan *setter* untuk atribut book dan location. Dengan begitu, data di class Library bisa diambil dan diubah dengan cara yang lebih teratur dan sesuai prinsip OOP.



Sebelum:



```
class Library {
    public Book book;
    public String location;

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

    public void showLibraryInfo() {
        System.out.println("Library Location: " + location);
        book.displayInfo();
    }
}
```

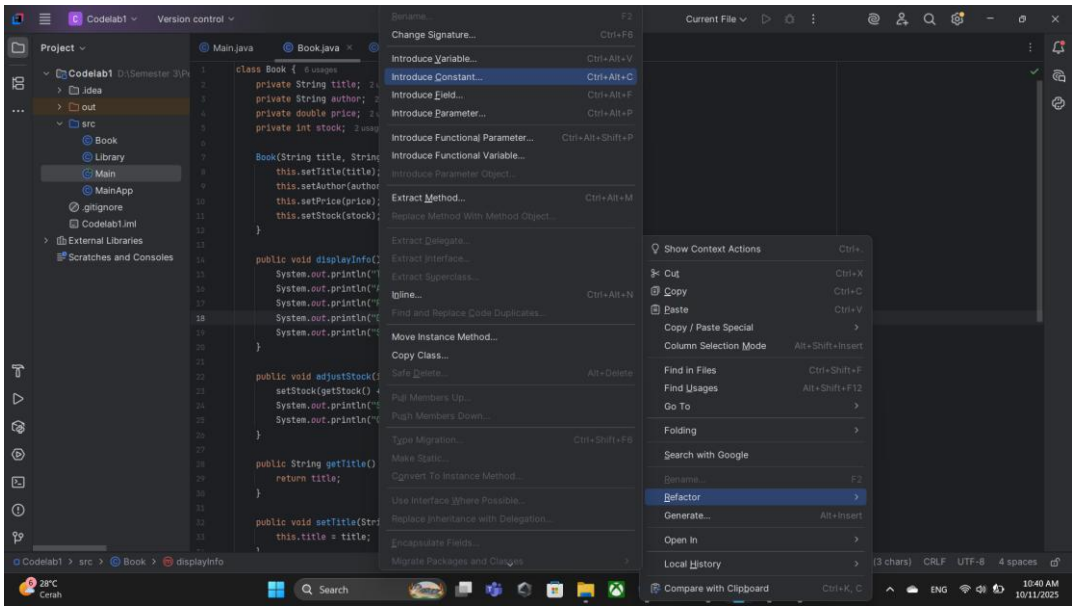
Sesudah:

```
class Library {
    private Book book;
    private String location;

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

    public void showLibraryInfo() {
        System.out.println("Library Location: " + getLocation());
        getBook().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;
    }
}
```

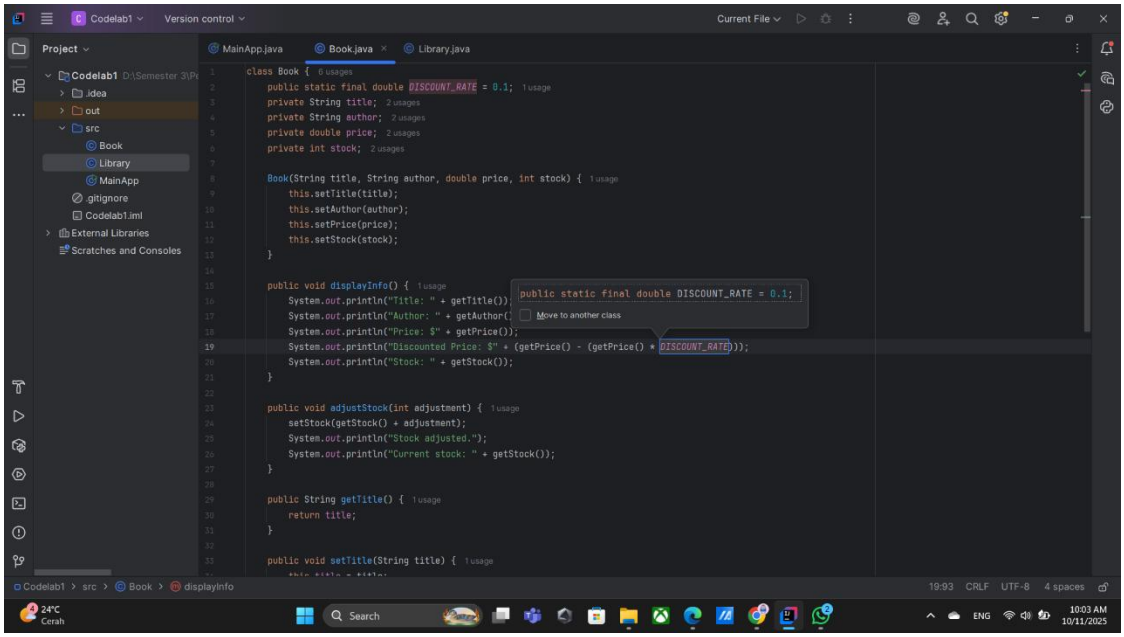
3. Introduce Constant

   Pada langkah ini, kita menambahkan sebuah konstanta di dalam class Book untuk menyimpan nilai diskon. Konstanta ini diberi nama DISCOUNT_RATE dengan nilai 0.1 atau setara dengan 10%.

   Tujuannya supaya kalau nanti ada perubahan nilai diskon, kita cukup ubah di satu tempat saja tanpa harus mencari di semua baris kode.

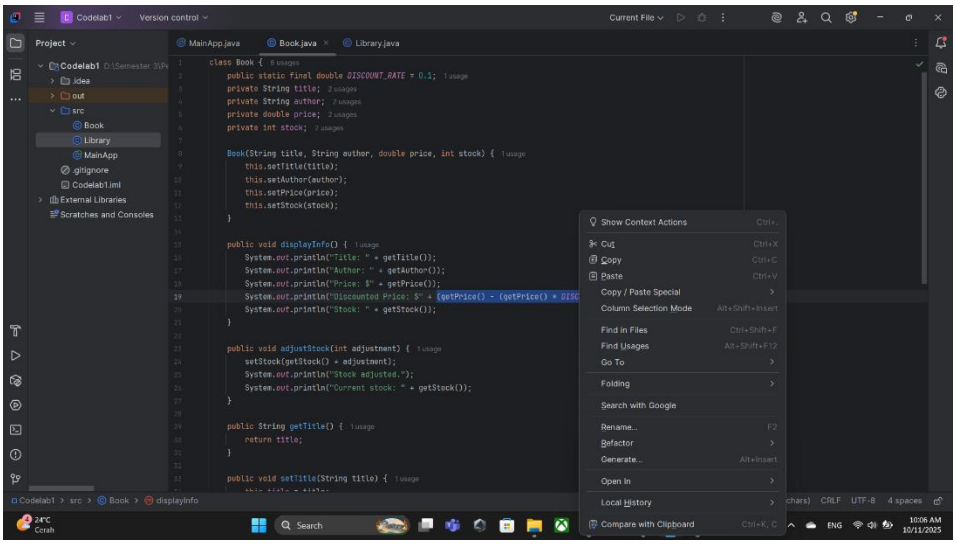


   Gambar di atas menunjukkan langkah saat melakukan refactor menggunakan fitur Introduce Constant di IntelliJ IDEA. Caranya yaitu klik kanan pada nilai yang ingin dijadikan konstanta (misalnya 0.1), lalu pilih Refactor → Introduce Constant. Setelah itu, beri nama konstanta tersebut dengan DISCOUNT_RATE dan IDE akan otomatis menambahkannya ke dalam class Book.



4. Extract Method

Pada langkah ini, kita memisahkan perhitungan diskon dari metode displayInfo() menjadi metode baru bernama calculateDiscount().
Tujuannya supaya kode lebih rapi, mudah dibaca, dan fungsi perhitungannya bisa dipakai kembali tanpa harus menulis ulang di tempat lain.



Langkah-langkah:

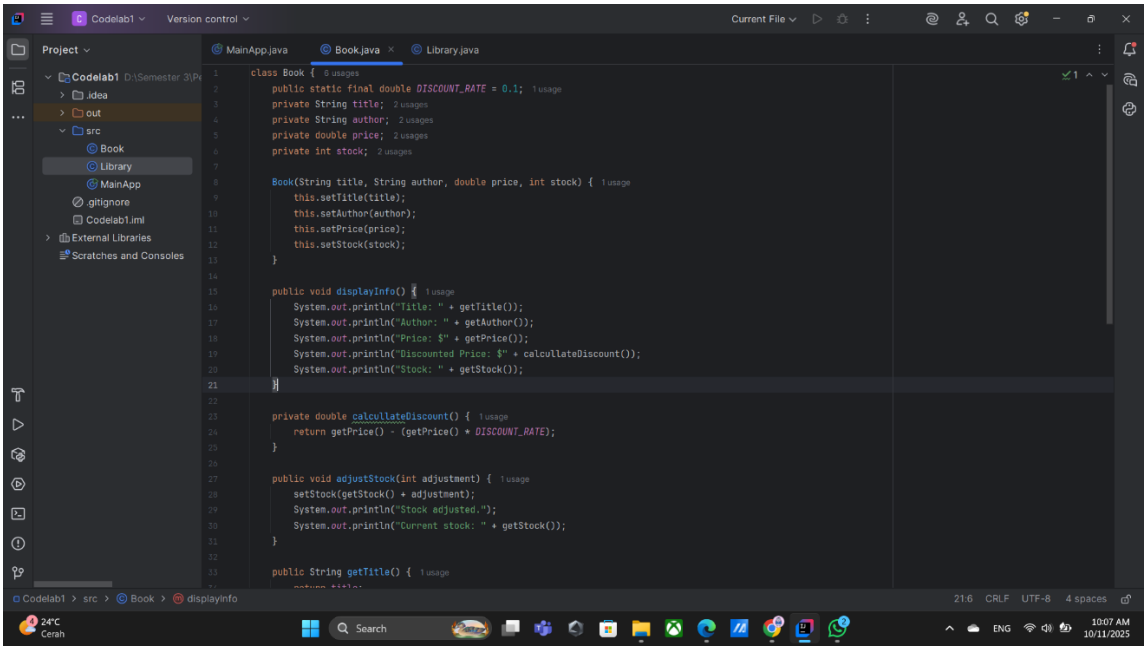
a. Blok bagian kode yang digunakan untuk menghitung diskon di dalam metode displayInfo().

b. Klik kanan pada bagian kode tersebut.

c. Pilih menu Refactor → Extract Method.

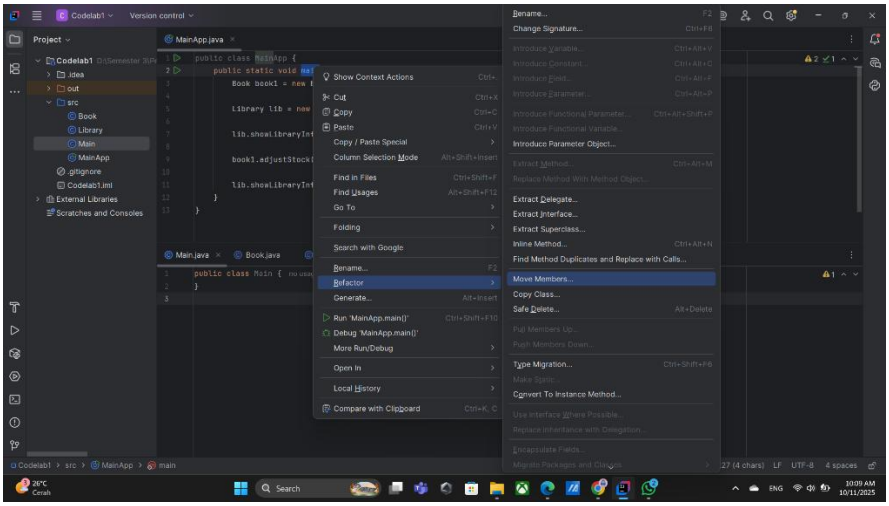
d. Beri nama metode baru dengan calculateDiscount.

e. Klik Refactor, dan IDE akan otomatis membuat metode baru untuk menghitung diskon.



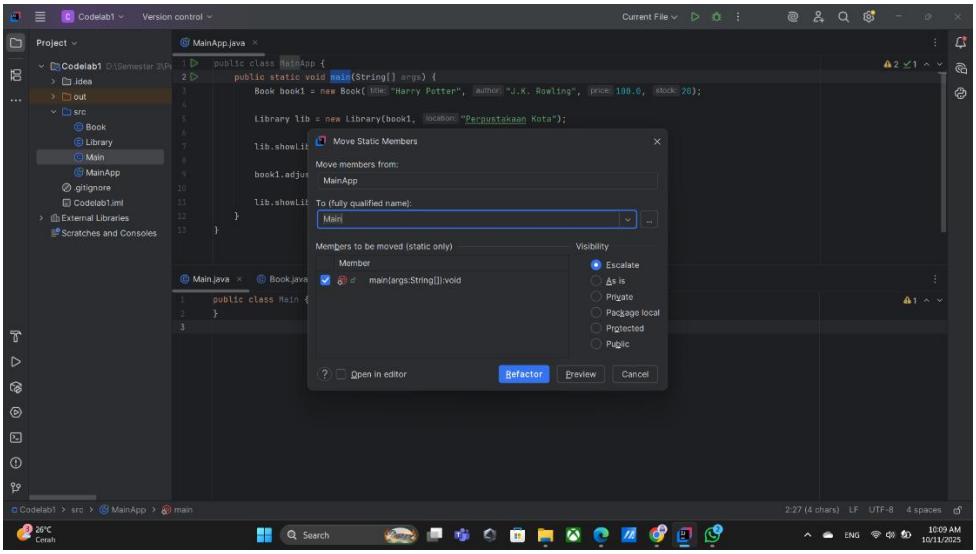
5. Move Method

Pada langkah ini, kita memindahkan method main() dari class MainApp ke class baru bernama Main. Tujuannya supaya struktur program lebih rapi dan sesuai aturan, di mana class Main berfungsi sebagai titik awal program.

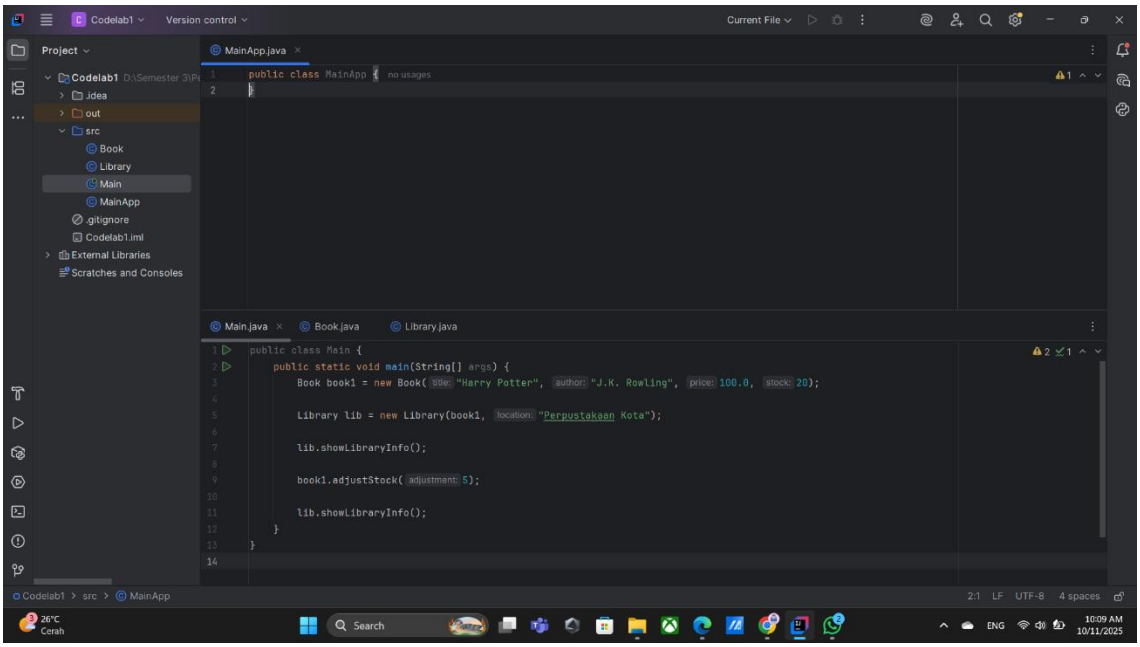


Langkah-langkah:

1. Buat class baru dengan nama Main.
2. Buka class MainApp, lalu blok bagian kode method main().
3. Klik kanan pada bagian yang sudah diblok, lalu pilih Refactor → Move Members.
4. Setelah itu, pilih class Main sebagai tempat tujuan untuk memindahkan method tersebut.



5. Klik Refactor, dan method main() akan otomatis berpindah ke class Main.



Hasil Program

```
Library Location: Perpustakaan Kota
Title: Harry Potter
Author: J.K. Rowling
Price: $100.0
Discounted Price: $90.0
Stock: 20
Stock adjusted.
Current stock: 25
Library Location: Perpustakaan Kota
Title: Harry Potter
Author: J.K. Rowling
Price: $100.0
Discounted Price: $90.0
Stock: 25
```

Kesimpulan

Setelah dilakukan proses refactoring, program menjadi lebih rapi dan mudah dipahami. Kode yang sebelumnya masih sederhana kini sudah mengikuti konsep pemrograman berorientasi objek (OOP), khususnya prinsip Encapsulation dan Abstraction. Selain itu, struktur program juga menjadi lebih modular, sehingga mudah untuk diperbaiki, dikembangkan, atau ditambahkan fitur baru tanpa harus mengubah bagian utama kode.