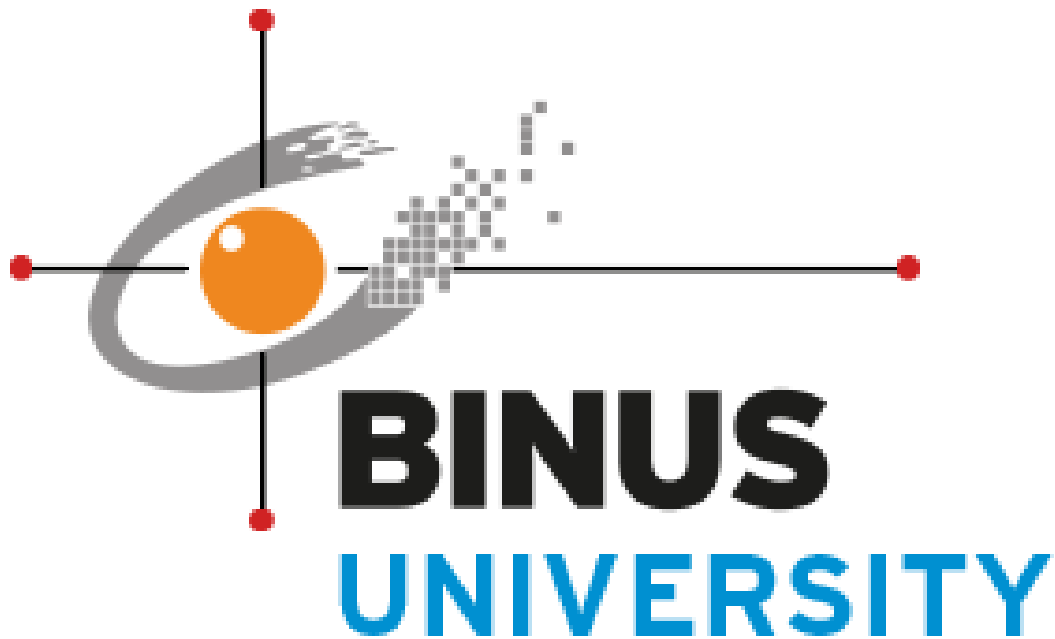


Library Management System Refactored Smells



Group Name

1. Nicholas Brandon Chang 2702288065
2. Raihan Khalis Hartono 2702249341
3. Christopher Imanuel Lesmana 2702220813
4. Felix Ferdinand 2702230221

UNIVERSITAS BINA NUSANTARA

JAKARTA

I. Introduction

Dalam pengembangan software, kualitas code menjadi aspek yang sangat penting agar sistem dapat dikelola, diperluas, dan dipelihara dengan baik. Salah satu indikator rendahnya kualitas code adalah adanya code smell atau design smell, yaitu indikasi adanya masalah dalam desain atau implementasi yang dapat menyebabkan penurunan kualitas software, meskipun code tersebut masih dapat dijalankan.

Pada proyek ini, kami mengambil sebuah sistem dari repositori GitHub berjudul Library Management System untuk dianalisis dan dievaluasi lebih lanjut. Sistem tersebut merupakan aplikasi manajemen perpustakaan yang menangani proses seperti peminjaman dan pengembalian buku. Meskipun fungsionalitas utama sudah berjalan, proyek ini masih mengandung beberapa code smell atau design smell yang dapat mempengaruhi keberlanjutan dan skalabilitas sistem di masa depan.

Tujuan utama dari proyek ini adalah untuk mengidentifikasi code smell dan design smell yang ada dalam sistem tersebut, memberikan penjelasan mengapa hal tersebut termasuk dalam kategori smell, serta merancang dan menerapkan solusi perbaikan (refactoring) untuk meningkatkan kualitas struktur code. Dengan melakukan refactoring berdasarkan prinsip-prinsip desain software yang baik, diharapkan sistem menjadi lebih bersih, efisien, dan mudah dipelihara.

Melalui laporan ini, kami ingin mendemonstrasikan pentingnya praktik code refactoring dalam menjaga kualitas software, khususnya dalam konteks proyek Library Management System. Laporan ini akan mencakup identifikasi masalah, analisis dampaknya, serta solusi yang telah diterapkan untuk mengatasi smells yang ditemukan.

A. Code / Design Smell

1. Code Smell : Long Method

- Location : main method
- Explanation : main method menangani beberapa responsibility: menampilkan menu, menerima input pengguna untuk berbagai aksi (meminjam, mengembalikan, exit), memproses input tersebut, dan memanggil method yang sesuai di class Library. Meskipun hal ini umum dilakukan dalam aplikasi konsol kecil, pada sistem yang lebih besar, method ini bisa menjadi sulit untuk dikelola.

2. Code Smell : Comments

- Location : Banyak comments di dalam main method
- Explanation : Code yang baik seharusnya dapat menjelaskan dirinya sendiri. Comments akan lebih berguna jika menjelaskan alasan mengapa sesuatu dilakukan dengan cara tertentu, atau menjelaskan logika yang kompleks, daripada sekadar menyatakan apa yang sudah jelas dilakukan oleh code tersebut. Comments seperti itu hanya menambah kekacauan tanpa banyak manfaat.

3. Code Smell : Duplicated Code

- Location : Logic di dalam metode borrowBook() dan returnBook()
- Explanation : Kedua method melakukan iterasi melalui daftar buku untuk menemukan objek Book yang cocok. Logika pencarian ini diulang. Jika cara pencarian atau identifikasi buku berubah, maka Anda harus memperbaikinya di beberapa tempat sekaligus.

4. Code Smell : Dead Code

- Location : public void displayDetails()

- Explanation : Method `displayDetails()` didefinisikan di dalam class `Book`, tetapi tidak pernah dipanggil di bagian mana pun dari code yang disediakan. Code yang tidak digunakan hanya menambah kekacauan dan bisa membingungkan bagi pengelola code di masa depan.

B. Object Oriented Smell

1. Encapsulation Smell : Leaky Encapsulation

- Location : `Book.java` (`borrowBook()`, `returnBook()`)
- Explanation : Encapsulation berarti bahwa suatu object harus menjaga keadaan internal dan implementasi perilakunya tetap bersifat private, hanya mengekspos interface yang terdefinisi dengan baik. Meskipun status `isBorrowed` bersifat private dan dimodifikasi melalui method, object `Book` mengambil tanggung jawab untuk mencetak pesan langsung ke konsol dari dalam method `borrowBook()` dan `returnBook()`.

2. Abstraction Smell : Unutilized Abstraction

- Location : `Book.java` (`printDetails()`)
- Explanation : Mengapa ini merupakan sebuah unutilized abstraction?. Method ini tidak pernah dipanggil di mana pun dalam code yang disediakan. Ini adalah sebuah unutilized code, sebuah abstraksi (sebuah perilaku dari `Book`) yang tidak memiliki tujuan atau fungsi dalam sistem saat ini.

3. Modularization Smell : Broken Modularization

- Location : `Book.java`
- Explanation : Meskipun class `Book` sebagian besar sudah kohesif terhadap konsep sebuah buku, menyertakan output ke konsol secara langsung dalam class model ini sedikit merusak modularitasnya. Sebuah modul "`Book`" seharusnya

idealnya bersifat mandiri dalam hal keadaan (state) dan perilaku inti yang terkait dengan buku. Dengan menyertakan I/O konsol secara langsung :

- Ia mengambil tanggung jawab yang berkaitan dengan UI, yang seharusnya ditangani oleh modul lain

II. Methodology

Refactoring Techniques :

1. Long Method : Refactoring technique yang digunakan untuk mengatasi code smell ini adalah “**Extract Method**”, di mana kita akan mengekstrak logika untuk menangani peminjaman dan pengembalian buku ke dalam private static method yang terpisah.
2. Comments : Kita hanya menghapus komentar yang tidak diperlukan.
3. Duplicated Code : Refactoring technique yang akan kita gunakan adalah “**Extract Method**”. Kita akan mengekstrak logika pencarian buku yang umum ke dalam private helper method baru, misalnya findBook(String title, String author). Method ini akan bertanggung jawab untuk mencari dalam daftar buku dan mengembalikan object Book jika ditemukan, atau null jika tidak. Method borrowBook() dan returnBook() kemudian akan menggunakan metode bantu ini.
4. Dead Code : Menghapus code yang tidak digunakan merupakan teknik yang paling sederhana dan umum untuk menangani code smell dead code ini. Jika suatu bagian code (seperti method, variable, field, class, atau block of code yang unreachable) telah dipastikan tidak digunakan dan tidak direncanakan untuk pengembangan dalam waktu dekat maka dapat kita hapus.
5. Leaky Encapsulation : Refactoring teknik yang digunakan untuk refactor code smell leaky encapsulation ini adalah. Modify Method Signature dan Move Method. Dimana kita ubah Book.borrowBook() dan Book.returnBook() agar mengembalikan nilai boolean yang menunjukkan keberhasilan atau kegagalan operasi. Dan kita memindahkan metode

mencetak pesan ke console yang terkait peminjaman/pengembalian dari class Book ke class Library.

6. Unutilized Abstraction : Teknik refactor yang digunakan untuk membenarkan smell tersebut adalah dengan (Delete Unused Code), ini adalah teknik yang paling langsung dan umum. Jika sebuah abstraction tidak digunakan dan tidak ada rencana konkret dalam waktu dekat maka abstraction tersebut sebaiknya dihapus. Menghapus printDetails() method karena tidak digunakan.
7. Broken Modularization : Teknik refactor yang digunakan untuk refactor code smell Broken Modularization adalah Modify Method Signature dan Move Method. Dimana kita memindahkan print ke console dari book.java ke class lain dan kita mengubah return status untuk borrowBook() dan returnBook() menjadi boolean.

III. Result and Analysis

Hasil dari refactoring technique yang saya dapatkan adalah sebagai berikut :

- Long Method : Setelah code smell tersebut di refactor hal ini sangat membantu untuk mengurangi line of code dari method tersebut dan meningkatkan keterbacaan dengan memberikan nama deskriptif pada blok code yang di extract serta mendorong penggunaan ulang code jika logika yang di extract dibutuhkan di tempat lain.

Before :

```
1
2 import java.util.Scanner;
3
4 public class LibraryManagementSystem {
5     public static void main(String[] args){
6         // Create a scanner object for user input.
7         Scanner scanner = new Scanner(System.in);
8
9         // Create a Library object to manage the collection of the books.
10        Library library = new Library();
11
12        // Add some books to the Library's collection.
13        library.addBook(new Book("Verity", "Colleen Hoover"));
14        library.addBook(new Book("The Secret", "Rhonda Byrne"));
15        library.addBook(new Book("Grit", "Angela Duckworth"));
16
17        // Infinite loop to keep the system running until the user chooses to exit.
18        while(true) {
19            // Display the available books in the Library.
20            System.out.println("\nAvailable Books: ");
21            library.displayAvailableBooks();
22
23            // Prompt the user for an action: borrow, return, or exit.
24            System.out.println("\nDo you want to borrow or return a book?");
25            System.out.println("Type 'borrow' to borrow a book, 'return' to return a book, or 'exit' to quit.");
26            String action = scanner.nextLine().toLowerCase(); // Read the user input and convert it into lower case.
27
28            // Check if the user wants to borrow a book.
29            if(action.equalsIgnoreCase("Borrow")){
30                // Get the title and author of the book to be borrowed.
31                System.out.print("Enter the title of the book you want to borrow: ");
32                String borrowTitle = scanner.nextLine();
33                System.out.print("Enter the author of the book you want to borrow: ");
34                String borrowAuthor = scanner.nextLine();
35                // Borrow the specified book from the Library.
36                library.borrowBook(borrowTitle, borrowAuthor);
37
38            } // Check if the user wants to return a book.
39            else if(action.equalsIgnoreCase("Return")){
40                // Get the title and author of the book to return.
41                System.out.print("Enter the title of the book you want to return: ");
42                String returnTitle = scanner.nextLine();
43                System.out.print("Enter the author of the book you want to return: ");
44                String returnAuthor = scanner.nextLine();
45                // Return the specified book to the Library.
46                library.returnBook(returnTitle, returnAuthor);
47
48            } // Check if the user wants to exit the system.
49            else if (action.equalsIgnoreCase("exit")) {
50                // Exit condition
51                System.out.println("Exiting the system. Goodbye!");
52                break;
53            } // Handle invalid input from the user.
54            else {
55                System.out.println("Invalid option. Please enter 'borrow', 'return', or 'exit'.");
56            }
57        }
58        // Close the scanner to prevent resource leaks.
59        scanner.close();
60    }
61 }
62
63 }
64
65
66
```

After :

```
1 package refactored;
2 import java.util.Scanner;
3
4 public class LibraryManagementSystem {
5
6     public static void main(String[] args) {
7         try (Scanner scanner = new Scanner(System.in)) {
8             Library library = new Library();
9
10            library.addBook(new Book("Verity", "Colleen Hoover"));
11            library.addBook(new Book("The Secret", "Rhonda Byrne"));
12            library.addBook(new Book("Grit", "Angela Duckworth"));
13
14            boolean running = true;
15            while (running) {
16                System.out.println("\nAvailable Books: ");
17                library.displayAvailableBooks();
18
19                System.out.println("\nMenu:");
20                System.out.println("Type 'borrow' to borrow a book.");
21                System.out.println("Type 'return' to return a book.");
22                System.out.println("Type 'exit' to quit.");
23                System.out.print("Enter your choice: ");
24                String action = scanner.nextLine().trim().toLowerCase();
25                switch (action) {
26                    case "borrow":
27                        handleBorrowAction(scanner, library);
28                        break;
29                    case "return":
30                        handleReturnAction(scanner, library);
31                        break;
32                    case "exit":
33                        System.out.println("Exiting the system. Goodbye!");
34                        running = false;
35                        break;
36                    default:
37                        System.out.println("Invalid option. Please enter 'borrow', 'return', or 'exit'.");
38                        break;
39                }
40            }
41        }
42    }
43
44    private static void handleBorrowAction(Scanner scanner, Library library) {
45        System.out.print("Enter the title of the book you want to borrow: ");
46        String borrowTitle = scanner.nextLine().trim();
47        System.out.print("Enter the author of the book you want to borrow: ");
48        String borrowAuthor = scanner.nextLine().trim();
49        library.borrowBook(borrowTitle, borrowAuthor);
50    }
51
52    private static void handleReturnAction(Scanner scanner, Library library) {
53        System.out.print("Enter the title of the book you want to return: ");
54        String returnTitle = scanner.nextLine().trim();
55        System.out.print("Enter the author of the book you want to return: ");
56        String returnAuthor = scanner.nextLine().trim();
57        library.returnBook(returnTitle, returnAuthor);
58    }
59 }
```


- Duplicate Code : Duplicate code pada logic borrowBook dan returnBook dalam class Library telat diatasi dengan extract method. Hal ini menghilangkan redundansi, sehingga jika ada perubahan pada cara pencarian buku, modifikasi hanya perlu dilakukan pada satu method.

Before :

```

1
2 import java.util.ArrayList;
3
4 public class Library{
5     // Private member variable to store a list of the books in the Library.
6     private ArrayList<Book> books;
7
8     // Constructor to initialize the ArrayList of books.
9     public Library() {
10         this.books = new ArrayList<Book>();
11     }
12
13     // Method to add a book in the Library's collection.
14     public void addBook(Book book) {
15         books.add(book);
16     }
17
18
19     // Method to display all the available (not borrowed) books in the library.
20     public void displayAvailableBooks(){
21         // Iterate through the list of books.
22         for(Book book : books){
23             // Checking the availability of the book.
24             if(book.isAvailable()){
25                 // Display the title and author of the available books.
26                 System.out.println(book.getTitle() + " by " + book.getAuthor());
27             }
28         }
29     }
30
31     // Method to borrow a book.
32     public void borrowBook(String title, String author){
33         // Iterate through the list of books.
34         for(Book book : books){
35             // To check if the book's title and author matches the input and if its available.
36             if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isAvailable()){
37                 if(book.isAvailable()){
38                     // If the book is available, borrow it.
39                     book.borrowBook();
40                     System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
41                 } else {
42                     System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
43                 }
44                 return; // Exit the method after borrowing.
45             }
46         }
47         // If the book is not found or available display a message.
48         System.out.println("Sorry, the book is not available!");
49     }
50
51     // Method to return a book.
52     public void returnBook(String title, String author){
53         // Iterate through the list of books.
54         for(Book book : books){
55             // Check if the book's title and author matches to the input and if its currently borrowed.
56             if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isBorrowed()){
57                 // If the book is borrowed, return it.
58                 if (!book.isAvailable()) {
59                     book.returnBook(); // Mark the book as returned
60                     System.out.println("You have successfully returned '" + title + "' by " + author + ".");
61                 }
62                 return; // Exit the method after returning.
63             }
64         }
65         // If the book was not found or wasn't borrowed, display a message.
66         System.out.println("This book wasn't borrowed from the Library!");
67     }
68 }
69
70
71
72

```

After :

```
3 import java.util.ArrayList;
4
5 public class Library {
6     private ArrayList<Book> books;
7
8     public Library() {
9         this.books = new ArrayList<Book>();
10    }
11
12    public void addBook(Book book) {
13        books.add(book);
14    }
15
16    public void displayAvailableBooks() {
17        boolean found = false;
18        System.out.println("Available Books:"); // Added a header for clarity
19        for (Book book : books) {
20            if (book.isAvailable()) {
21                System.out.println(book.getTitle() + " by " + book.getAuthor());
22                found = true;
23            }
24        }
25        if (!found) {
26            System.out.println("No books are currently available in the library.");
27        }
28    }
29
30    private Book findBook(String title, String author) {
31        for (Book book : books) {
32            if (book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author)) {
33                return book;
34            }
35        }
36        return null;
37    }
38
39    public void borrowBook(String title, String author) {
40        Book bookToBorrow = findBook(title, author);
41
42        if (bookToBorrow != null) {
43            if (bookToBorrow.isAvailable()) {
44                bookToBorrow.borrowBook();
45            }
46            else {
47                System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
48            }
49        }
50        else {
51            System.out.println("Sorry, the book '" + title + "' by " + author + " was not found in the library.");
52        }
53    }
54
55    public void returnBook(String title, String author) {
56        Book bookToReturn = findBook(title, author);
57
58        if (bookToReturn != null) {
59            if (bookToReturn.isBorrowed()) {
60                bookToReturn.returnBook();
61            }
62            else {
63                System.out.println("This book, '" + title + "' by " + author + ", was not borrowed or is already available.");
64            }
65        }
66        else {
67            System.out.println("Sorry, the book '" + title + "' by " + author + " was not found in the library (cannot return).");
68        }
69    }
70
71 }
```

- Comments : Hasil dari delete comment yang tidak berguna di dalam code ini adalah hasil code yang lebih bersih dan tidak bingung untuk dibaca. Keterbacaan pengembang tidak terganggu oleh comments yang tidak memberikan nilai tambah.

Before :

```

1
2 import java.util.ArrayList;
3
4 public class Library{
5     // Private member variable to store a list of the books in the Library.
6     private ArrayList<Book> books;
7
8     // Constructor to initialize the ArrayList of books.
9     public Library() {
10         this.books = new ArrayList<Book>();
11     }
12
13     // Method to add a book in the Library's collection.
14     public void addBook(Book book) {
15         books.add(book);
16     }
17
18
19     // Method to display all the available (not borrowed) books in the library.
20     public void displayAvailableBooks(){
21         // Iterate through the list of books.
22         for(Book book : books){
23             // Checking the availability of the book.
24             if(book.isAvailable()){
25                 // Display the title and author of the available books.
26                 System.out.println(book.getTitle() + " by " + book.getAuthor());
27             }
28         }
29     }
30
31     // Method to borrow a book.
32     public void borrowBook(String title, String author){
33         // Iterate through the list of books.
34         for(Book book : books){
35             // To check if the book's title and author matches the input and if its available.
36             if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isAvailable()){
37                 if(book.isAvailable()){
38                     // If the book is available, borrow it.
39                     book.borrowBook();
40                     System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
41                 } else {
42                     System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
43                 }
44                 return; // Exit the method after borrowing.
45             }
46         }
47         // If the book is not found or available display a message.
48         System.out.println("Sorry, the book is not available!");
49     }
50
51     // Method to return a book.
52     public void returnBook(String title, String author){
53         // Iterate through the list of books.
54         for(Book book : books){
55             // Check if the book's title and author matches to the input and if its currently borrowed.
56             if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isBorrowed()){
57                 // If the book is borrowed, return it.
58                 if (!book.isAvailable()) {
59                     book.returnBook(); // Mark the book as returned
60                     System.out.println("You have successfully returned '" + title + "' by " + author + ".");
61                 }
62                 return; // Exit the method after returning.
63             }
64         }
65         // If the book was not found or wasn't borrowed, display a message.
66         System.out.println("This book wasn't borrowed from the Library!");
67     }
68 }
69 }
70
71
72

```

After :

```
1 package refactored;
2
3 import java.util.ArrayList;
4
5 public class Library{
6     private ArrayList<Book> books;
7
8     public Library() {
9         this.books = new ArrayList<Book>();
10    }
11
12    public void addBook(Book book) {
13        books.add(book);
14    }
15
16
17    public void displayAvailableBooks(){
18        for(Book book : books){
19            if(book.isAvailable()){
20                System.out.println(book.getTitle() + " by " + book.getAuthor());
21            }
22        }
23    }
24
25    public void borrowBook(String title, String author){
26        for(Book book : books){
27            if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isAvailable()){
28                if(book.isAvailable()){
29                    book.borrowBook();
30                    System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
31                } else {
32                    System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
33                }
34                return;
35            }
36        }
37        System.out.println("Sorry, the book is not available!");
38    }
39
40    public void returnBook(String title, String author){
41        for(Book book : books){
42            if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isBorrowed()){
43                if (!book.isAvailable()) {
44                    book.returnBook();
45                    System.out.println("You have successfully returned '" + title + "' by " + author + ".");
46                }
47                return;
48            }
49        }
50        System.out.println("This book wasn't borrowed from the Library!");
51    }
52
53 }
54
55
56
```

- Dead Code : Metode displayDetails() pada class book telah dihapus. Hal tersebut membuat code menjadi lebih bersih dengan menghilangkan code yang tidak terpakai, kompleksitas code berkurang, dan pengembang di masa depan tidak akan bingung atau membuang waktu untuk memahami code yang sebenarnya tidak memiliki fungsi.

Before :

```
1  public class Book{
2
3      private String title;
4      private String author;
5      private boolean isBorrowed;
6
7      public Book(String title, String author){
8          this.title = title;
9          this.author = author;
10         this.isBorrowed = false;
11     }
12
13     public void borrowBook() {
14         if(!isBorrowed){
15             isBorrowed = true;
16             System.out.println("Book is successfully borrowed: " + title);
17         } else{
18             System.out.println("Sorry, this book is already borrowed");
19         }
20     }
21
22     public void returnBook() {
23         if(isBorrowed){
24             isBorrowed = false;
25             System.out.println("You've successfully returned the book: " + title);
26         } else {
27             System.out.println("This book wasn't borrowed");
28         }
29     }
30
31     public boolean isAvailable() {
32         return !isBorrowed;
33     }
34     public String getTitle(){
35         return title;
36     }
37     public String getAuthor(){
38         return author;
39     }
40     public boolean isBorrowed(){
41         return isBorrowed;
42     }
43     public void displayDetails() {
44         System.out.println("Title: " + title + ", Author: " + author);
45     }
46 }
47
48
49
```

After :

```
1 package refactored;
2 public class Book {
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author) {
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13    public void borrowBook() {
14        if (!isBorrowed) {
15            isBorrowed = true;
16            System.out.println("Book is successfully borrowed: " + title);
17        } else {
18            System.out.println("Sorry, this book is already borrowed");
19        }
20    }
21
22    public void returnBook() {
23        if (isBorrowed) {
24            isBorrowed = false;
25            System.out.println("You've successfully returned the book: " + title);
26        } else {
27            System.out.println("This book wasn't borrowed");
28        }
29    }
30
31    public boolean isAvailable() {
32        return !isBorrowed;
33    }
34
35    public String getTitle() {
36        return title;
37    }
38
39    public String getAuthor() {
40        return author;
41    }
42
43    public boolean isBorrowed() {
44        return isBorrowed;
45    }
46 }
47
48
```

- Leaky Encapsulation : Smell ini terjadi pada class book dimana method borrowBook() dan returnBook() langsung mencetak pesan ke console, hasil dari refactor ini adalah enkapsulasi class book menjadi lebih kuat karena ia hanya fokus pada pengelolaan status internalnya.

Before :

```
1 public class Book{
2
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author){
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13    public void borrowBook() {
14        if(!isBorrowed){
15            isBorrowed = true;
16            System.out.println("Book is successfully borrowed: " + title);
17        } else{
18            System.out.println("Sorry, this book is already borrowed");
19        }
20    }
21
22    public void returnBook() {
23        if(isBorrowed){
24            isBorrowed = false;
25            System.out.println("You've successfully returned the book: " + title);
26        } else {
27            System.out.println("This book wasn't borrowed");
28        }
29    }
30
31    public boolean isAvailable() {
32        return !isBorrowed;
33    }
34    public String getTitle(){
35        return title;
36    }
37    public String getAuthor(){
38        return author;
39    }
40    public boolean isBorrowed(){
41        return isBorrowed;
42    }
43    public void displayDetails() {
44        System.out.println("Title: " + title + ", Author: " + author);
45    }
46 }
47
48
49
50
51
52
```

```

1 |
2 import java.util.ArrayList;
3
4 public class Library{
5     private ArrayList<Book> books;
6
7     // Constructor to initialize the ArrayList of books.
8     public Library() {
9         this.books = new ArrayList<Book>();
10    }
11
12    // Method to add a book in the Library's collection.
13    public void addBook(Book book) {
14        books.add(book);
15    }
16
17
18    // Method to display all the available (not borrowed) books in the library.
19    public void displayAvailableBooks(){
20        // Iterate through the list of books.
21        for(Book book : books){
22            // Checking the availability of the book.
23            if(book.isAvailable()){
24                // Display the title and author of the available books.
25                System.out.println(book.getTitle() + " by " + book.getAuthor());
26            }
27        }
28    }
29
30    // Method to borrow a book.
31    public void borrowBook(String title, String author){
32        // Iterate through the list of books.
33        for(Book book : books){
34            // To check if the book's title and author matches the input and if its available.
35            if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isAvailable()){
36                if(book.isAvailable()){
37                    // If the book is available, borrow it.
38                    book.borrowBook();
39                    System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
40                } else {
41                    System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
42                }
43                return; // Exit the method after borrowing.
44            }
45        }
46        // If the book is not found or available display a message.
47        System.out.println("Sorry, the book is not available!");
48    }
49
50    // Method to return a book.
51    public void returnBook(String title, String author){
52        // Iterate through the list of books.
53        for(Book book : books){
54            // Check if the book's title and author matches the input and if its currently borrowed.
55            if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isBorrowed()){
56                // If the book is borrowed, return it.
57                if (!book.isAvailable()) {
58                    book.returnBook(); // Mark the book as returned
59                    System.out.println("You have successfully returned '" + title + "' by " + author + ".");
60                }
61                return; // Exit the method after returning.
62            }
63        }
64        // If the book was not found or wasn't borrowed, display a message.
65        System.out.println("This book wasn't borrowed from the Library!");
66    }
67
68 }
69

```

After :


```
1 package refactored;
2 public class Book {
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author) {
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13
14    public boolean borrowBook() {
15        if (!isBorrowed) {
16            isBorrowed = true;
17            return true;
18        }
19        return false;
20    }
21
22    public boolean returnBook() {
23        if (isBorrowed) {
24            isBorrowed = false;
25            return true;
26        }
27        return false;
28    }
29
30    public boolean isAvailable() {
31        return !isBorrowed;
32    }
33
34    public String getTitle() {
35        return title;
36    }
37
38    public String getAuthor() {
39        return author;
40    }
41
42    public boolean isBorrowed() {
43        return isBorrowed;
44    }
45 }
46
47
```

```

5 public class Library {
6     private ArrayList<Book> books;
7
8     public Library() {
9         this.books = new ArrayList<>();
10    }
11
12    public void addBook(Book book) {
13        books.add(book);
14    }
15
16    private Book findBook(String title, String author) {
17        for (Book book : books) {
18            if (book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author)) {
19                return book;
20            }
21        }
22        return null;
23    }
24
25    public void displayAvailableBooks() {
26        boolean found = false;
27        System.out.println("Available Books:");
28        for (Book book : books) {
29            if (book.isAvailable()) {
30                System.out.println(book.getTitle() + " by " + book.getAuthor());
31                found = true;
32            }
33        }
34        if (!found) {
35            System.out.println("No books are currently available in the library.");
36        }
37    }
38
39    public void borrowBook(String title, String author) {
40        Book bookToBorrow = findBook(title, author);
41
42        if (bookToBorrow != null) {
43            if (bookToBorrow.isAvailable()) {
44                if (bookToBorrow.borrowBook()) {
45                    System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
46                } else {
47                    System.out.println("Failed to borrow '" + title + "' by " + author + ". An unexpected issue occurred.");
48                }
49            } else {
50                System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
51            }
52        } else {
53            System.out.println("Sorry, the book '" + title + "' by " + author + " was not found in the library.");
54        }
55    }
56
57    public void returnBook(String title, String author) {
58        Book bookToReturn = findBook(title, author);
59
60        if (bookToReturn != null) {
61            if (bookToReturn.isBorrowed()) {
62                if (bookToReturn.returnBook()) {
63                    System.out.println("You have successfully returned '" + title + "' by " + author + ".");
64                } else {
65                    System.out.println("Failed to return '" + title + "' by " + author + ". An unexpected issue occurred.");
66                }
67            } else {
68                System.out.println("This book, '" + title + "' by " + author + ", was not borrowed or is already available.");
69            }
70        } else {
71            System.out.println("Sorry, the book '" + title + "' by " + author + " was not found in the library.");
72        }
73    }

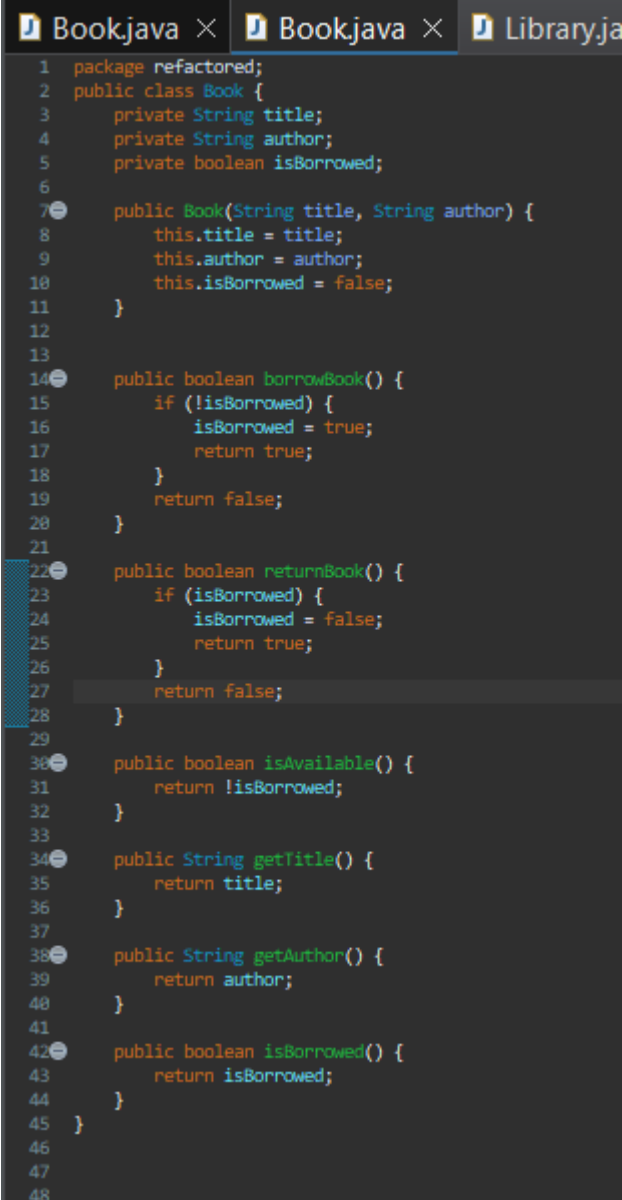
```

- Unutilized Abstraction : Method displayDetails() di class book merupakan contoh abstraction yang tidak dimanfaatkan. Sama seperti dead code sehingga method tersebut telah dihapus. Maka hal tersebut akan mengurangi beban kognitif pada pengembang dan potensi kebingungan mengenai fungsionalitas sebenarnya disediakan.

Before :

```
1 public class Book{
2
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author){
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13    public void borrowBook() {
14        if(!isBorrowed){
15            isBorrowed = true;
16            System.out.println("Book is successfully borrowed: " + title);
17        } else{
18            System.out.println("Sorry, this book is already borrowed");
19        }
20    }
21
22    public void returnBook() {
23        if(isBorrowed){
24            isBorrowed = false;
25            System.out.println("You've successfully returned the book: " + title);
26        } else {
27            System.out.println("This book wasn't borrowed");
28        }
29    }
30
31    public boolean isAvailable() {
32        return !isBorrowed;
33    }
34    public String getTitle(){
35        return title;
36    }
37    public String getAuthor(){
38        return author;
39    }
40    public boolean isBorrowed(){
41        return isBorrowed;
42    }
43    public void displayDetails() {
44        System.out.println("Title: " + title + ", Author: " + author);
45    }
46 }
47
48
49
```

After :



The screenshot shows an IDE with three tabs: 'Book.java', 'Book.java', and 'Library.java'. The first 'Book.java' tab is active and displays the following Java code:

```
1 package refactored;
2 public class Book {
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author) {
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13
14    public boolean borrowBook() {
15        if (!isBorrowed) {
16            isBorrowed = true;
17            return true;
18        }
19        return false;
20    }
21
22    public boolean returnBook() {
23        if (isBorrowed) {
24            isBorrowed = false;
25            return true;
26        }
27        return false;
28    }
29
30    public boolean isAvailable() {
31        return !isBorrowed;
32    }
33
34    public String getTitle() {
35        return title;
36    }
37
38    public String getAuthor() {
39        return author;
40    }
41
42    public boolean isBorrowed() {
43        return isBorrowed;
44    }
45 }
46
47
48
```

- Broken Modularization : Smell ini terkait erat dengan leaky encapsulation pada class book. Setelah refactor, modul book menjadi lebih kohesif dan fokus pada tanggung jawab intinya. Batas antar modul menjadi lebih jelas, meningkatkan independensi modul book.

Before :

```
1 public class Book{
2
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author){
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13    public void borrowBook() {
14        if(!isBorrowed){
15            isBorrowed = true;
16            System.out.println("Book is successfully borrowed: " + title);
17        } else{
18            System.out.println("Sorry, this book is already borrowed");
19        }
20    }
21
22    public void returnBook() {
23        if(isBorrowed){
24            isBorrowed = false;
25            System.out.println("You've successfully returned the book: " + title);
26        } else {
27            System.out.println("This book wasn't borrowed");
28        }
29    }
30
31    public boolean isAvailable() {
32        return !isBorrowed;
33    }
34    public String getTitle(){
35        return title;
36    }
37    public String getAuthor(){
38        return author;
39    }
40    public boolean isBorrowed(){
41        return isBorrowed;
42    }
43    public void displayDetails() {
44        System.out.println("Title: " + title + ", Author: " + author);
45    }
46 }
47
48
49
50
51
52
```

```

1 |
2 import java.util.ArrayList;
3
4 public class Library{
5     private ArrayList<Book> books;
6
7     // Constructor to initialize the ArrayList of books.
8     public Library() {
9         this.books = new ArrayList<Book>();
10    }
11
12    // Method to add a book in the Library's collection.
13    public void addBook(Book book) {
14        books.add(book);
15    }
16
17
18    // Method to display all the available (not borrowed) books in the library.
19    public void displayAvailableBooks(){
20        // Iterate through the list of books.
21        for(Book book : books){
22            // Checking the availability of the book.
23            if(book.isAvailable()){
24                // Display the title and author of the available books.
25                System.out.println(book.getTitle() + " by " + book.getAuthor());
26            }
27        }
28    }
29
30    // Method to borrow a book.
31    public void borrowBook(String title, String author){
32        // Iterate through the list of books.
33        for(Book book : books){
34            // To check if the book's title and author matches the input and if its available.
35            if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isAvailable()){
36                if(book.isAvailable()){
37                    // If the book is available, borrow it.
38                    book.borrowBook();
39                    System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
40                } else {
41                    System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
42                }
43                return; // Exit the method after borrowing.
44            }
45        }
46        // If the book is not found or available display a message.
47        System.out.println("Sorry, the book is not available!");
48    }
49
50    // Method to return a book.
51    public void returnBook(String title, String author){
52        // Iterate through the list of books.
53        for(Book book : books){
54            // Check if the book's title and author matches the input and if its currently borrowed.
55            if(book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author) && book.isBorrowed()){
56                // If the book is borrowed, return it.
57                if (!book.isAvailable()) {
58                    book.returnBook(); // Mark the book as returned
59                    System.out.println("You have successfully returned '" + title + "' by " + author + ".");
60                }
61                return; // Exit the method after returning.
62            }
63        }
64        // If the book was not found or wasn't borrowed, display a message.
65        System.out.println("This book wasn't borrowed from the Library!");
66    }
67
68 }
69

```

After :

```
1 package refactored;
2 public class Book {
3     private String title;
4     private String author;
5     private boolean isBorrowed;
6
7     public Book(String title, String author) {
8         this.title = title;
9         this.author = author;
10        this.isBorrowed = false;
11    }
12
13
14    public boolean borrowBook() {
15        if (!isBorrowed) {
16            isBorrowed = true;
17            return true;
18        }
19        return false;
20    }
21
22    public boolean returnBook() {
23        if (isBorrowed) {
24            isBorrowed = false;
25            return true;
26        }
27        return false;
28    }
29
30    public boolean isAvailable() {
31        return !isBorrowed;
32    }
33
34    public String getTitle() {
35        return title;
36    }
37
38    public String getAuthor() {
39        return author;
40    }
41
42    public boolean isBorrowed() {
43        return isBorrowed;
44    }
45 }
46
47
```

```

5 public class Library {
6     private ArrayList<Book> books;
7
8     public Library() {
9         this.books = new ArrayList<>();
10    }
11
12    public void addBook(Book book) {
13        books.add(book);
14    }
15
16    private Book findBook(String title, String author) {
17        for (Book book : books) {
18            if (book.getTitle().equalsIgnoreCase(title) && book.getAuthor().equalsIgnoreCase(author)) {
19                return book;
20            }
21        }
22        return null;
23    }
24
25    public void displayAvailableBooks() {
26        boolean found = false;
27        System.out.println("Available Books:");
28        for (Book book : books) {
29            if (book.isAvailable()) {
30                System.out.println(book.getTitle() + " by " + book.getAuthor());
31                found = true;
32            }
33        }
34        if (!found) {
35            System.out.println("No books are currently available in the library.");
36        }
37    }
38
39    public void borrowBook(String title, String author) {
40        Book bookToBorrow = findBook(title, author);
41
42        if (bookToBorrow != null) {
43            if (bookToBorrow.isAvailable()) {
44                if (bookToBorrow.borrowBook()) {
45                    System.out.println("You have successfully borrowed '" + title + "' by " + author + ".");
46                } else {
47                    System.out.println("Failed to borrow '" + title + "' by " + author + ". An unexpected issue occurred.");
48                }
49            } else {
50                System.out.println("Sorry, '" + title + "' by " + author + " is already borrowed.");
51            }
52        } else {
53            System.out.println("Sorry, the book '" + title + "' by " + author + " was not found in the library.");
54        }
55    }
56
57    public void returnBook(String title, String author) {
58        Book bookToReturn = findBook(title, author);
59
60        if (bookToReturn != null) {
61            if (bookToReturn.isBorrowed()) {
62                if (bookToReturn.returnBook()) {
63                    System.out.println("You have successfully returned '" + title + "' by " + author + ".");
64                } else {
65                    System.out.println("Failed to return '" + title + "' by " + author + ". An unexpected issue occurred.");
66                }
67            } else {
68                System.out.println("This book, '" + title + "' by " + author + ", was not borrowed or is already available.");
69            }
70        } else {
71            System.out.println("Sorry, the book '" + title + "' by " + author + " was not found in the library.");
72        }
73    }

```


IV. Conclusion

Proyek analisis dan refactor code smell pada sistem Library Management System ini telah berhasil mengidentifikasi dan menangani berbagai jenis code smell yang berpotensi menurunkan kualitas software. Melalui proses refactor yang sistematis, kualitas code telah ditingkatkan secara signifikan, yang berdampak positif pada berbagai aspek pengembangan dan pemeliharaan software.

Berdasarkan analisis yang telah dilakukan, beberapa code smell utama yang ditemukan dan berhasil direfaktor meliputi:

1. **Long Method (Metode Panjang):** Menyebabkan metode sulit dipahami dan dipelihara.
2. **Duplicate Code (code Duplikat):** Meningkatkan risiko inkonsistensi dan menambah beban kerja saat ada perubahan.
3. **Dead Code (code Mati):** Menambah kompleksitas yang tidak perlu pada codebase.
4. **Comments (Komentar yang Menjelaskan Hal Jelas):** Mengurangi kejelasan code karena informasi redundan.
5. **Leaky Encapsulation (Enkapsulasi yang Bocor):** Mencampurkan tanggung jawab antar class dan mengurangi reusabilitas.
6. **Unutilized Abstraction (Abstraksi yang Tidak Digunakan):** Menyederhanakan antarmuka class dan mengurangi kebingungan.
7. **Broken Modularization (Modularisasi yang Rusak):** Mengganggu kohesi modul dan meningkatkan ketergantungan yang tidak perlu.

Setelah melakukan refactor terhadap code smell tersebut, dapat disimpulkan bahwa:

- **Keterbacaan code (Readability):** code menjadi jauh lebih mudah dibaca dan dipahami. Dengan metode yang lebih pendek, penghapusan code duplikat, dan komentar yang lebih relevan, alur logika program menjadi lebih jelas.
- **Kemudahan Pemeliharaan (Maintainability):** Perubahan atau penambahan fitur di masa depan akan lebih mudah dilakukan. Dengan code yang lebih terstruktur, terpusatnya logika yang sama, dan hilangnya code mati, upaya untuk memodifikasi sistem menjadi lebih efisien dan risiko kesalahan berkurang.

- **Modularitas dan Kohesi:** Setiap modul atau class kini memiliki tanggung jawab yang lebih fokus dan jelas. Perbaikan pada Leaky Encapsulation dan Broken Modularization memastikan bahwa class-class seperti book hanya menangani logika inti mereka, meningkatkan kohesi internal dan mengurangi ketergantungan antar modul.
- **Reusabilitas (Reusability):** Komponen code, terutama class book setelah perbaikan enkapsulasi, menjadi lebih mudah untuk digunakan kembali dalam konteks atau sistem lain karena tidak lagi terikat pada implementasi UI spesifik (seperti output konsol).
- **Pengurangan Kompleksitas:** Secara keseluruhan, kompleksitas codebase berkurang. Penghapusan code yang tidak perlu (Dead Code, Unutilized Abstraction) dan penyederhanaan struktur (Long Method, Duplicate Code) membuat sistem lebih ramping dan mudah dikelola.

Proses identifikasi dan refactor code smell merupakan praktik penting dalam rekayasa software. Meskipun pada awalnya memerlukan upaya tambahan, manfaat jangka panjang dalam hal kualitas, stabilitas, dan efisiensi pengembangan sangatlah signifikan. Proyek ini menunjukkan bahwa dengan kesadaran akan code smell dan penerapan teknik refactor yang tepat, kualitas software dapat ditingkatkan secara berkelanjutan, menghasilkan sistem yang lebih baik dan lebih mudah untuk dikembangkan lebih lanjut.

V. Reference

1. <https://martinfowler.com/books/refactoring.html>
2. <https://sourcemaking.com/refactoring>
3. <https://sourcemaking.com/refactoring/smells>
4. <https://refactoring.guru/refactoring/smells>
5. <https://refactoring.guru/refactoring/techniques>