COM102 (Object Oriented Programming) - Practical Skills Assessment 2

Design and Development Document

Introduction

This design and development document will provide a comprehensive overview of the design, implementation, and functionality of the library management system that we have created, and we will focus on why we decided to pick some of the specific implementation decisions that were made during the development phase for our system. First of all, according to the given brief that we were given we had to create a system designed to handle the storage and management of library items, including books and multimedia, and the loans associated with these items. The primary goal of the system is to provide a user-friendly that is clear, intuitive, and most importantly accessible to all users so when they do operate the system they should do so with minimal effort, as we didn't want to have a system which was very disorganized as this would mostly likely led users to end up feeling very frustrated and ultimately decide to not use our system at all. The console-based interface provides a main menu for librarians to perform various operations, such as issuing loans, renewing loans, and recording the return of items and also includes reporting capabilities to provide information about the current state of the library's loans.

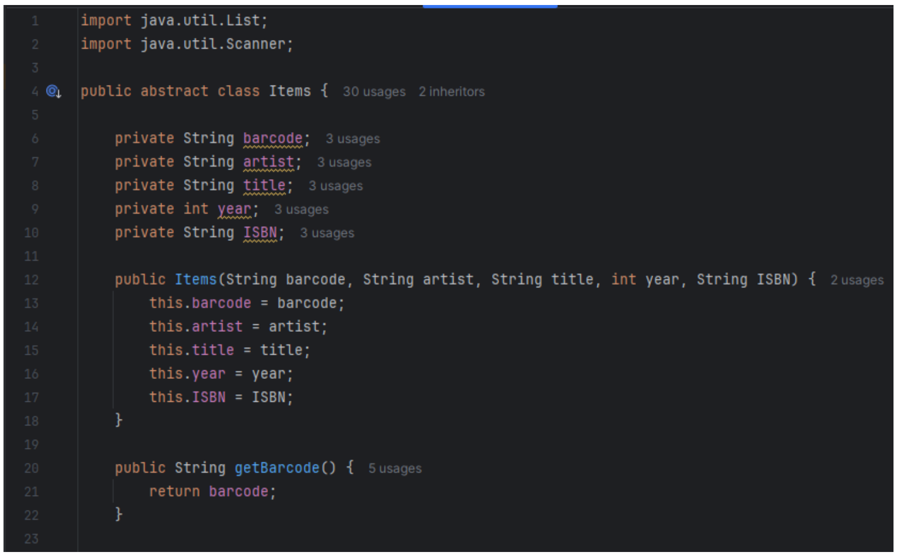
The system aims to make sure that the library operations runs smoothly with a robust platform for effectively managing loans and items, ensuring that accurate record-keeping is followed and making sure that it follows an object-oriented approach, leveraging the principles of inheritance, polymorphism, and encapsulation to create a modular and extensible codebase. We will also discuss about class hierarchy within this document such as the data structures used, and the program control structures implemented to achieve the required functionality, also the key classes that we have implemented will be important such as Items, Books, Multimedia, User, Loans, and the Library Management which are each responsible for representing essential entities and managing the interactions that we have throughout our system. Additionally, we will also discuss the handling of console input and the integration with CSV files for persistent storage of user and item data, as well as the current loan information into our system.

Key Functional that were Required:

- Recording details of each loan, including Barcode, User ID, Issue Date, Due Date, and Number of Renews.
- Predefined set of library users and lendable items.
- Ability to issue, renew, and return library items.
- Viewing all items currently on loan.
- Generating reports on loans, including library name, total number of each loan type, and percentage of items renewed more than once.
- File handling to store loan data accurately and persistently.

Object-Oriented Principles:

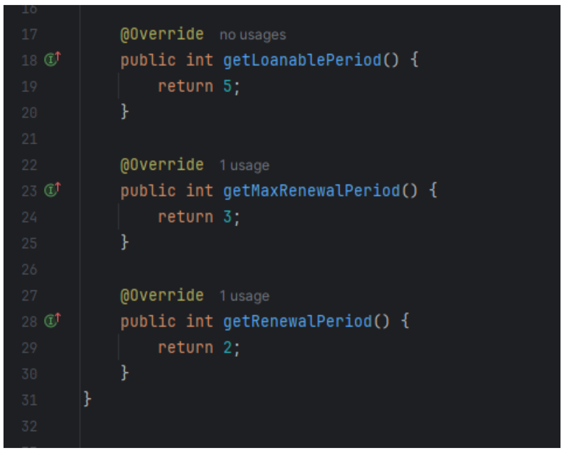
The fundamental element and the backbone of our system has to be the Item abstract class as this serves as the base class for the different types of Items, which represents a loanable item in the library. This class has two concrete subclasses which are book and multimedia, the Item class defines the common properties and behaviours shared by all library items, such as the barcode, artist/author, title, year, and ISBN. While the loan and user class is different, first of all the Loan class represents a loan of a library item to a user, storing the relevant information like the barcode, user ID, issue date, due date, and number of renewals. While the User class encapsulates the details of a library member, including the user ID and name:



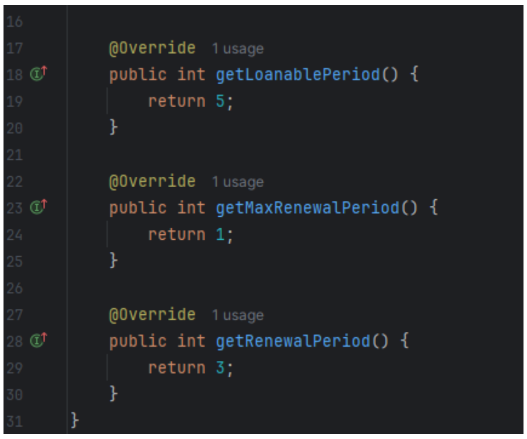
```java
import java.util.List;
import java.util.Scanner;

public abstract class Items {  30 usages  2 inheritors

    private String barcode;  3 usages
    private String artist;  3 usages
    private String title;  3 usages
    private int year;  3 usages
    private String ISBN;  3 usages

    public Items(String barcode, String artist, String title, int year, String ISBN) {  2 usages
        this.barcode = barcode;
        this.artist = artist;
        this.title = title;
        this.year = year;
        this.ISBN = ISBN;
    }

    public String getBarcode() {  5 usages
        return barcode;
    }
```

Polymorphism

Polymorphism is utilized in the Item class hierarchy through the implementation of the getLoanDuration(), getMaxRenewalPeriod(), and getRenewalPeriod() abstract methods. These methods are overridden in the Book and Multimedia subclasses to provide the specific implementation details for each type of item such as the appropriate loan and renewal durations for each item type. For example, the book class overrides these methods to return the loan duration of 5 weeks, a maximum of 3 renewals, and a renewal period of 2 weeks, similarly, we can see that the multimedia class follows a similar format which overrides the abstract methods to return a loan duration of 5 weeks, a maximum of 1 renewal, and a renewal period of 3 weeks



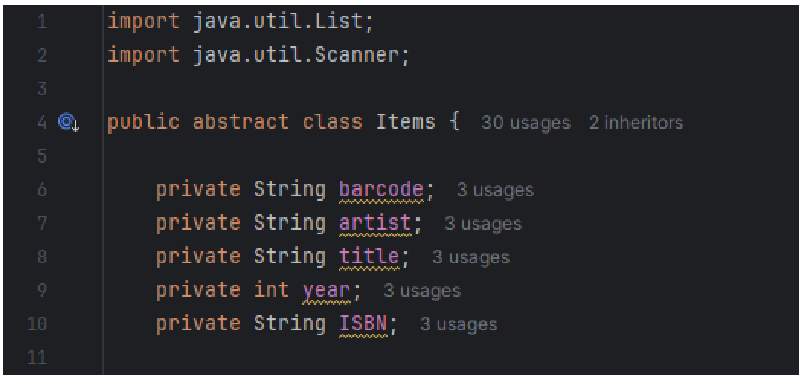
```java
    @Override  no usages
    public int getLoanablePeriod() {
        return 5;
    }

    @Override  1 usage
    public int getMaxRenewalPeriod() {
        return 3;
    }

    @Override  1 usage
    public int getRenewalPeriod() {
        return 2;
    }
}
```



```java
    @Override  1 usage
    public int getLoanablePeriod() {
        return 5;
    }

    @Override  1 usage
    public int getMaxRenewalPeriod() {
        return 1;
    }

    @Override  1 usage
    public int getRenewalPeriod() {
        return 3;
    }
}
```

Encapsulation

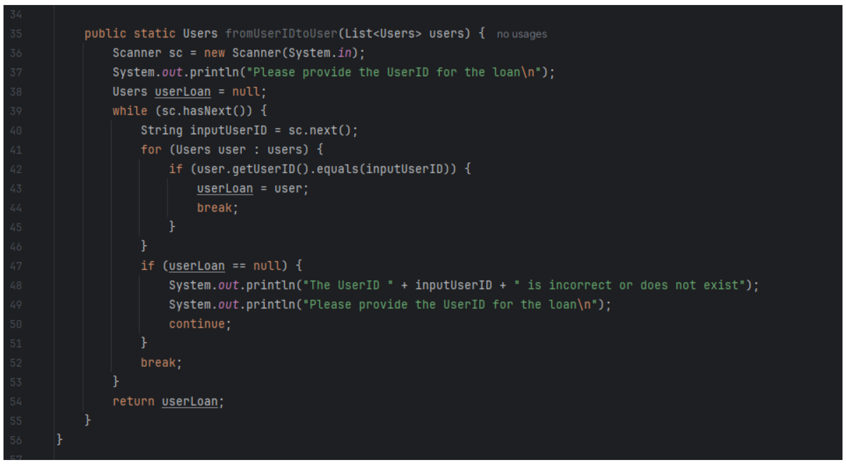
Within the system most of these classes have private instance variables and provide public methods to access and modify the data, ensuring proper encapsulation and data hiding. For instance, the item class has private instance variables for the barcode, artist, title, year, and ISBN, and provides public getter and setter methods to interact with these properties, the reason for this is that it helps to ensure the integrity of the system's data and gives a clear separation between the internal implementation and the given external interface. The library management class does not any private instance variables, instead it actually uses static variables and methods to store and handle the core functionality of the library system. which means they can be accessed directly without creating an instance of the class, which further illustrates and demonstrates a strong emphasis on data security and encapsulation as the client code can directly access these methods. Fundamentally this class helps to organise and arrange the overall functions of the system such as loading and saving data, displaying the main menu, and handling user options and much more.



```java
import java.util.List;
import java.util.Scanner;

public abstract class Items {  30 usages  2 inheritors

    private String barcode;  3 usages
    private String artist;  3 usages
    private String title;  3 usages
    private int year;  3 usages
    private String ISBN;  3 usages

```

Data Structures

An Array List is another important feature used to store the list of current loans within our system. There are many benefits for using this, it helps to provides efficient insertion, deletion, and retrieval of loan objects, which are the primary entities within our system. The Array List allows for easy management of the loan records and supports the various loan-related operations, such as issuing, renewing, and returning loans. For example in the user class, this method takes a list of users as input and returns a specific user based on the user ID provided. It also utilizes an Array List of users passed as a parameter to search for the desired user, the same goes for the loan class where it takes a list of loans and returns based on the barcode that has been provided. Then for the library it stores all of the instances of items such as books and multimedia etc.

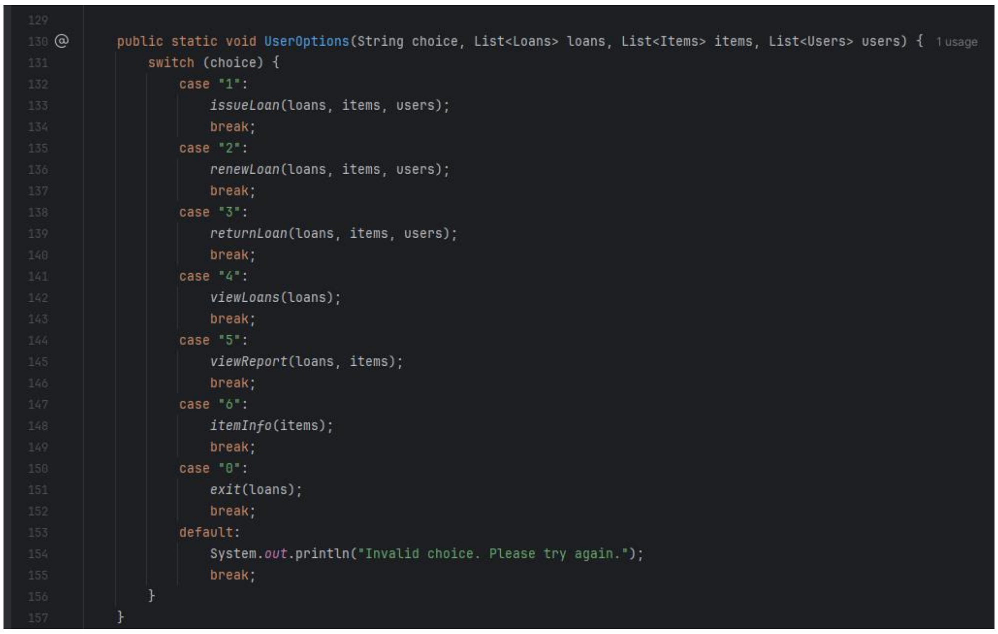


```java

    public static Users fromUserIDtoUser(List<Users> users) {  no usages
        Scanner sc = new Scanner(System.in);
        System.out.println("Please provide the UserID for the loan\n");
        Users userLoan = null;
        while (sc.hasNext()) {
            String inputUserID = sc.next();
            for (Users user : users) {
                if (user.getUserID().equals(inputUserID)) {
                    userLoan = user;
                    break;
                }
            }
            if (userLoan == null) {
                System.out.println("The UserID " + inputUserID + " is incorrect or does not exist");
                System.out.println("Please provide the UserID for the loan\n");
                continue;
            }
            break;
        }
        return userLoan;
    }
}
```

Program Control Structures

Switch Statements

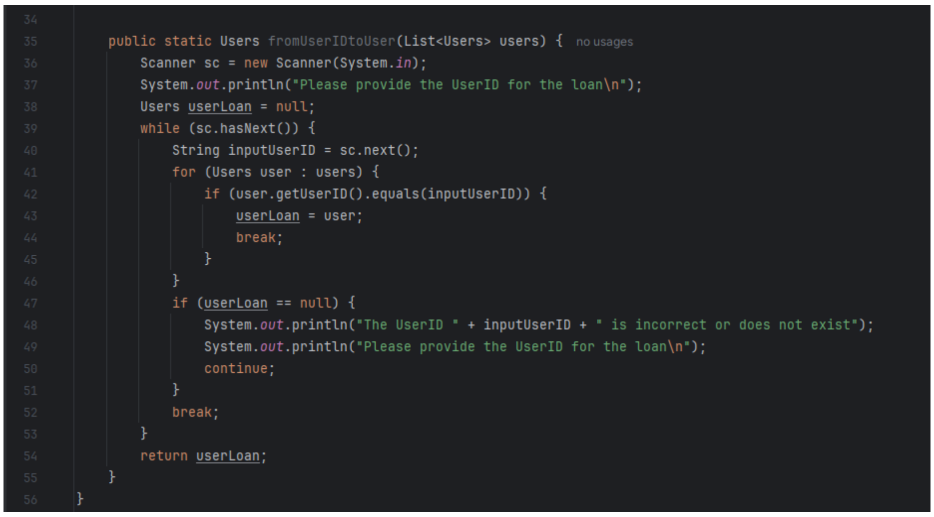
In the library management class Switch statements are used to handle the different user actions and options. The User Options () method handles the different user actions, such as issuing a loan, renewing a loan, and returning an item. In the User Options a switch statement is used to dispatch the user's choice to the corresponding functionality, for example the switch statement checks the value of the choice parameter and executes the corresponding method based on the user's selection. and also prompts a default case to handle invalid choices. This control structure provides a clear and organized way to manage the functionalities of the system especially as this is intended to work with the main menu as this clearly separates the different user actions, and it is more straightforward to add new cases to the switch statement if it is needed. Overall, this helps to make the code easier to understand and read as a result.



```java
    public static void UserOptions(String choice, List<Loans> loans, List<Items> items, List<Users> users) {  1 usage
        switch (choice) {
            case "1":
                issueLoan(loans, items, users);
                break;
            case "2":
                renewLoan(loans, items, users);
                break;
            case "3":
                returnLoan(loans, items, users);
                break;
            case "4":
                viewLoans(loans);
                break;
            case "5":
                viewReport(loans, items);
                break;
            case "6":
                itemInfo(items);
                break;
            case "0":
                exit(loans);
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }
    }
```

Conditional Statements

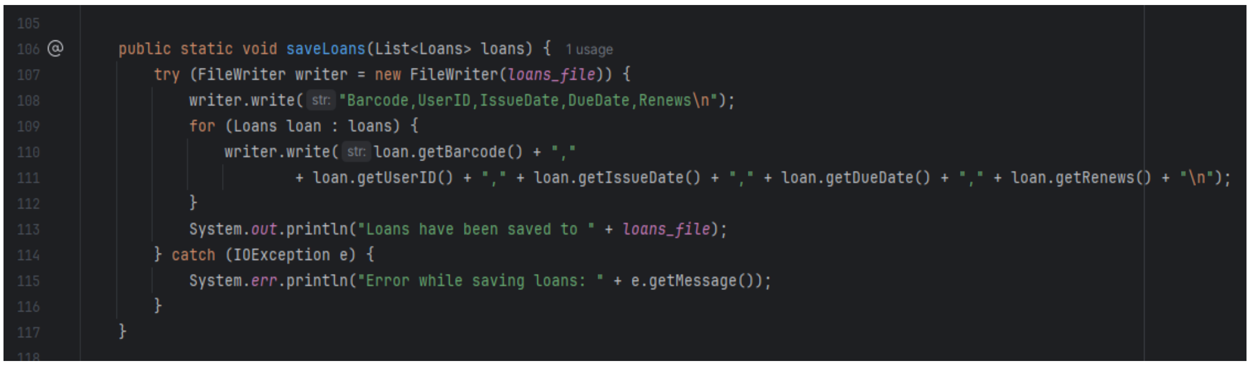
Also in the Library management class, Conditional statements which are (if-else) are used extensively throughout the system and is intended to handle various scenarios such as in the issue Loan(), renew Loan(), and return Loan() methods. Let’s talk about the first one which is In the issue Loan() method, this aims to the check if the provided user and item exist before creating a new loan, ensuring that it exists in the library of our system. For the second one is the renew Loan() method, similar to the one we talked above this checks if the loan does indeed exist and if the renewal is allowed before updating the due date and number of renewals. And lastly the return Loan() method checks to see if the loan does indeed exist before attempting to return the item and remove the loan from the list, this helps to enforce the loan duration and renewal rules. Let’s take the users class as an example in the method from user ID to user, the if statement is used to check if a user with the provided user ID exists in the list of users it will say a match is found, it assigns the current user to the user loan, however if it's not then it will say invalid, or doesn't exist. Overall, this demonstrates the integrity of the system's data to validate user input, check the availability of items and users.



```java
    public static Users fromUserIDtoUser(List<Users> users) {  no usages
        Scanner sc = new Scanner(System.in);
        System.out.println("Please provide the UserID for the loan\n");
        Users userLoan = null;
        while (sc.hasNext()) {
            String inputUserID = sc.next();
            for (Users user : users) {
                if (user.getUserID().equals(inputUserID)) {
                    userLoan = user;
                    break;
                }
            }
            if (userLoan == null) {
                System.out.println("The UserID " + inputUserID + " is incorrect or does not exist");
                System.out.println("Please provide the UserID for the loan\n");
                continue;
            }
            break;
        }
        return userLoan;
    }
}
```

Loops

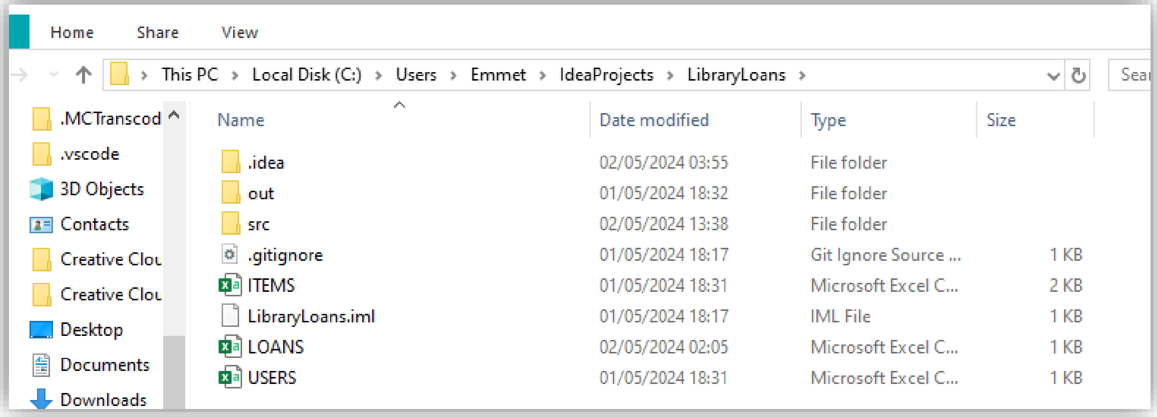
Loops (for and while) are heavily used to iterate through the lists of loans, users, and Items during various operations within our system, to ensure that efficient processing and management of the library's is done, it uses several loops to handle the loading, processing, and saving of the library's data, specifically the loans, items, and users, such as generating the loan report in the view Report() method and writing the loan data to the CSV file in the save Loans() method. These control structures allow the system to efficiently process collections of data and perform the necessary operations in a structured and well-organized manner.



```java
    public static void saveLoans(List<Loans> loans) {  1 usage
        try (FileWriter writer = new FileWriter(loans_file)) {
            writer.write( str: "Barcode,UserID,IssueDate,DueDate,Renews\n");
            for (Loans loan : loans) {
                writer.write( str: loan.getBarcode() + ","
                        + loan.getUserID() + "," + loan.getIssueDate() + "," + loan.getDueDate() + "," + loan.getRenews() + "\n");
            }
            System.out.println("Loans have been saved to " + loans_file);
        } catch (IOException e) {
            System.err.println("Error while saving loans: " + e.getMessage());
        }
    }
```

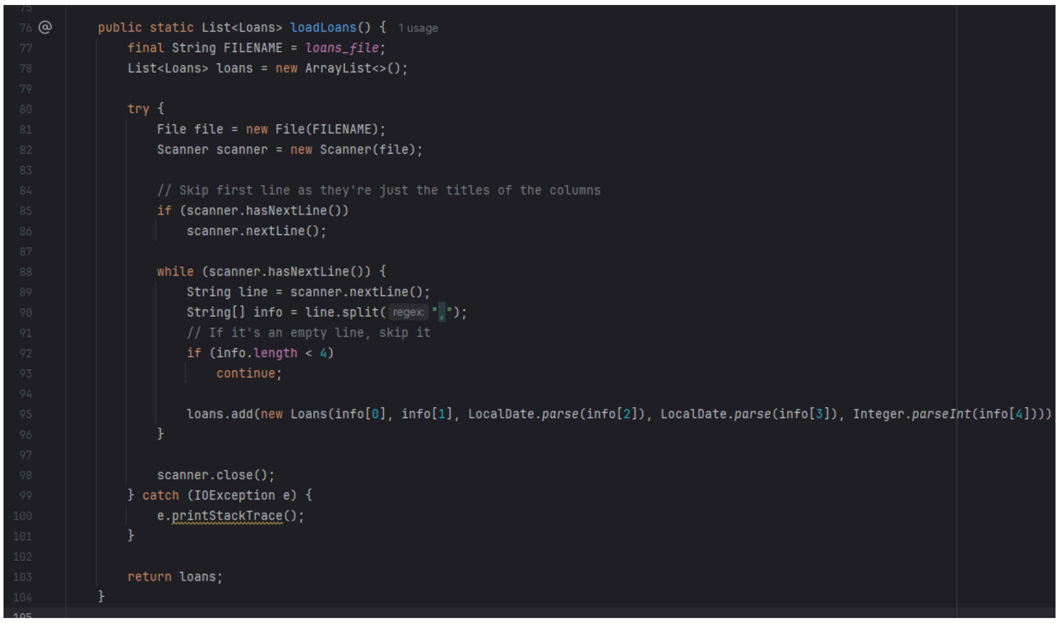
CSV File Input/Output

In our program the Scanner class is used in a lot of our classes such as user, items and mostly our library management, this is used to read user input from the console, such as barcodes, user IDs, and user actions, etc. This approach provides a simple and intuitive interface for the librarian to interact with the system. Given our assignment we had to read in the USERS.csv and ITEMS.csv files we did this by using the load Items(), load Users(), to get the initial data for users and items for our program. They both already contained input fields therefore all we had to do was to import this into our program the way we did this was to place the files located in the same directory as the Java program.



Name	Date modified	Type	Size
.idea	02/05/2024 03:55	File folder	
out	01/05/2024 18:32	File folder	
src	02/05/2024 13:38	File folder	
.gitignore	01/05/2024 18:17	Git Ignore Source ...	1 KB
ITEMS	01/05/2024 18:31	Microsoft Excel C...	2 KB
LibraryLoans.iml	01/05/2024 18:17	IML File	1 KB
LOANS	02/05/2024 02:05	Microsoft Excel C...	1 KB
USERS	01/05/2024 18:31	Microsoft Excel C...	1 KB

The program's ability to read data from CSV files and write the current loan data back to the LOANS.csv file CSV when exiting demonstrates a robust approach to data persistence. The way we did this was using the load Loans() method to read the data from the "ITEMS.csv", "USERS.csv", "LOANS.csv" files.



```java
public static List<Loans> loadLoans() {
    final String FILENAME = loans_file;
    List<Loans> loans = new ArrayList<>();

    try {
        File file = new File(FILENAME);
        Scanner scanner = new Scanner(file);

        // Skip first line as they're just the titles of the columns
        if (scanner.hasNextLine())
            scanner.nextLine();

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] info = line.split(",");
            // If it's an empty line, skip it
            if (info.length < 4)
                continue;

            loans.add(new Loans(info[0], info[1], LocalDate.parse(info[2]), LocalDate.parse(info[3]), Integer.parseInt(info[4])))
        }

        scanner.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return loans;
}
```

Conclusion

Overall, I feel that the library management system that we have created has been designed using an object-oriented approach to create a modular and extensible codebase. The adoption of inheritance, polymorphism, and encapsulation principles that I feel has resulted in a well-structured system that can effectively manage the library's items and loans. We have managed to create a system which consists of the library management class which uses the items, loans, and user’s classes to perform various library management operations, such as issuing, renewing, and returning loans. The Items class and its subclasses books and multimedia are used to represent the different types of items. The loans class is used to manage the loan-related information, including the association between a user, an item, and the loan details and lastly the user’s class is used to represent the library users and their associated information successfully.