The Final Report for the Final Project

The Group Name is: Java Coffee.

The Group Members: Hanan Senah, Lynn Hajj Hassan, and Fahim Kabir.

The variant is: Project 1: CLI-Based Library Management System.

The Group members with their contributions of the CODE:

Hanan Senah (YELLOW)

Lynn Hajj Hassan (BLUE)

Fahim Kabir (GREEN)

\*Each member has a colour that describes the work he/she implemented.

Please see the pdf that is attached with this word file.

The Additional features which students developed (if applicable).

\_Hanan Senah has done a power point representation to help represent in the class .

Please see the power point that is attached with this word file.

**Each Member Contributaion :**

**Hanan Senah:**

\_Class **Book (ALL DOEN BY ME)**: Base of the librarian class, and the class that represent the book.

#Description

The `Book` class models a book in a library. It includes fields for the book's title, author, genre, ISBN, availability, price, and image path. The class provides methods to access and modify these fields, check out and return the book, and retrieve book details.It provides a structured way to represent books in the library system, including their details, availability status, and price. It includes methods for checking out and returning books, as well as getters and setters to access and modify book attributes. This class is integral to managing the library's inventory and supporting various functionalities, such as generating reports and displaying book details in the user interface.

-isAvailable boolean` - The filed which show the availability status of the book (true if available, false if checked out).

# Constructor

public Book(String title, String author, String genre, String isbn, double price, String imagePath)

Initializes a new `Book` instance with the given title, author, genre, ISBN, price, and image path. The book is set to be available by default.

# Methods

getBookDetails():

  Returns a formatted string with the book's details, including title, author, genre, ISBN, availability, and price.

-checkout():Sets the book's availability to false, indicating it is checked out.

- returnBook():Sets the book's availability to true, indicating it is available.

- isAvailable():Returns the book's availability status.

For example of how to use: Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald", "Novel", "1234567890", 10.99, "path/to/image.jpg");

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The **User** class(ALL done by me) that is a parent class for the sub class librarian and the sub class consumer. It is also the base for the UserAccountSystem that done by lynn.

The **User** class encapsulates various attributes including:

* **Username**: The name of the user.
* **User ID**: A unique identifier for the user.
* **Password**: A secret key used for authentication.
* **Balance**: The amount of money the user has in their library account.
* **Email**: The user's email address.
* **Address**: The residential address of the user.
* **Phone Number**: The user's contact number.
* **Dollars Spent**: A metric tracking the amount of money the user has spent, which could be linked to rewards or coupons.

The class includes a constructor to initialize these attributes when a new user object is created. Additionally, it provides **getter and setter methods** for accessing and modifying each attribute, ensuring encapsulation and controlled access.

A **displayInfo()** method is also provided to print out the user's information, excluding sensitive data like the password. Lastly, there's a placeholder method getRole(), which is intended to be overridden in subclasses to specify the user's role, such as "Admin" or "Member."

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The **Librarian** sub class(ALL OF IT) that extened the parent (USER) class, that is representing the worker in the library that will have special methods like add , remove, generate report . . I used the Polymorphism method of overriding to override the abstract method getRole and change it to Librarian from the user class.

So,

The **Librarian** class extends the **User** class, adding specialized functionality for a librarian role within a library management system. This class inherits all the properties and methods of the **User** class, and also provides additional methods tailored to the librarian's responsibilities.

**Key Points:**

* **Constructor**: The Librarian constructor initializes the librarian's details by calling the parent User class constructor with the necessary attributes such as username, user ID, password, balance, email, address, phone number, and dollars spent.
* **Role Specification**: The getRole() method is overridden to return "Librarian", indicating the specific role of the user within the system.

**Additional Methods for Librarians:**

* **addBook()**: This method allows the librarian to add a new book to the library's collection. It takes a LibraryClass<T> instance, where T is a subclass of Book, and the book to be added.
* **removeBook()**: This method enables the librarian to remove a book from the library using its ISBN. It interacts with the library's book collection to find and remove the specified book.
* **generateReport()**: The librarian can generate a report of the library's books. This method accepts a LibraryClass<Book> instance and a file path where the report will be saved.

These additional methods provide the librarian with the necessary tools to manage the library's collection, maintaining and updating the inventory as needed.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The **consumer** sub class(ONLY HALF OF IT ) that is extended the parent class (USER) class, I did the constructer that will initialized the list for the borrowedBooks and create a list for the coupons . I used the Polymorphism method of overriding to override the abstract method getRole and change it to consumer from the user class.

So, The **Consumer** class extends the **User** class, representing a user who can borrow books and receive coupons in the library management system. This class inherits all the attributes and methods from the **User** class and adds specific functionalities relevant to consumers.

**Key Points:**

* **Constructor**: The Consumer constructor initializes the consumer's details by calling the parent User class constructor with attributes such as username, user ID, password, balance, email, address, phone number, and dollars spent. It also initializes two additional lists: borrowedBooks for tracking borrowed books and listOfCoupons for tracking the consumer's coupons. Additionally, it initializes NumberOfPurchase to zero, representing the number of purchases the consumer has made.
* **Role Specification**: The getRole() method is overridden to return "Consumer", indicating the specific role of the user within the system.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_ **LinkedList getBooks() and generateReport()** that done by me inside the Kabir class (LibraryClass). LinkedList getBooks() to return the books in the list to use it in the classes of FileReportGenerator and GUIReportGenerator. The method generateReport to be used in case 3 in the IamLibrarain method, that will refer to the class LibraryReportSystem.

So,

* The getBooks():

Provides access to the library's book collection.

 **Purpose**: This method retrieves the list of books in the library.

 **Details**: It returns a LinkedList containing the books stored in the library. This allows other parts of the program to access the library's book collection.

* generateReport(LibraryClass<Book> library, String filePath):

Facilitates generating and saving a report of the library's books to a specified file path.

 **Purpose**: This method generates a report of the library's books and saves it to a specified file path.

 **Details**: It calls another generateReport method from the LibraryClass instance, passing the current library instance and the desired file path as parameters. This method simplifies the process of generating a library report, making it easier to create and save reports with a single method call.

\_\_\_\_\_\_\_\_\_\_\_\_

\_The method **setUpLibrary** (ALL DONE BY HANAN) is a static method that initializes and sets up a LibraryClass instance with a collection of books.

\* **Initialization**:

* A new instance of LibraryClass<Book> is created and assigned to the variable library.

\* **Adding Books**:

* Four books are added to the library's collection using the addBook method. For each book, the constructor of the Book class is called with specific details such as the title, author, genre, ISBN, price, and image path. (I added 4 books :The Great Gatsby, To Kill a Mockingbird, Intro to Java, and 1984).

**\*Returning the Library Instance**:

* After all the books are added, the method returns the library instance, which now contains the initialized collection of books.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The method **IamLibrarian** (ALL DONE BY HANAN)that will be called in the main method is designed to provide a command-line interface for a librarian to manage a library's collection. It provides a simple console-based interface for managing the library's collection, including adding and removing books and generating reports. So,

\* **Initialization**:

* A Scanner object is created to read user input.
* A UserAccountsSystem object is initialized to manage user accounts (though not utilized in this snippet).
* A LibraryReportSystem object is created with the library parameter to manage report generation.
* A boolean variable keepRunning is set to true to control the loop's execution.

\* **Main Loop**:

* The method enters a while loop that continues running until keepRunning is set to false.

\* **Librarian Menu**:

* The menu options for the librarian are displayed, including adding a book, removing a book, generating a report, and logging out.

\* **User Input Handling**:

* The program checks if the user has entered an integer using scanner.hasNextInt().
* Depending on the user's choice, the corresponding case in the switch statement is executed:
  + **Case 1 (Add Book)**:
    - Prompts the librarian to enter the details of the new book, including title, author, genre, ISBN, price, and image path.
    - Validates each input to ensure correctness (e.g., title should not be numeric, ISBN should contain only digits and hyphens).
    - If the image path exists, a new Book object is created and added to the library.
  + **Case 2 (Remove Book)**:
    - Prompts the librarian to enter the title of the book to remove and removes the book from the library.
  + **Case 3 (Generate Report)**:
    - Calls the generateReport method of the reportSystem object to generate a report of the library's collection.
  + **Case 4 (Logout)**:
    - Sets keepRunning to false to exit the loop and end the method, effectively logging out the librarian.
  + **Default Case**:
    - Handles invalid menu choices by prompting the user to try again.

\* **Error Handling for Non-Integer Input**:

* If the user input is not an integer, the program prompts the user to try again and consumes the invalid input.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The class **LibraryReportSystem (ALL DONE BY ME, HANAN)** is designed to manage the generation of reports for a library's collection of books, and it centralizes the functionality related to generating reports, offering a simple way for librarians to choose the report format and manage the report generation process. So,

\* **Attributes**:

* library: A reference to a LibraryClass object that holds the collection of books.
* scanner: A Scanner object used to capture user input.

\* **Constructor**:

* The constructor initializes the library attribute with the given LibraryClass object and creates a new Scanner instance for reading user input.

\* **generateReport Method**:

* This method prompts the librarian to choose the format for generating the report.
* The available options are:
  1. **File**: Generates a report with general book information and saves it to a specified file path.
  2. **GUI**: Displays book information in a graphical user interface along with pictures.
* **Report Generation Process**:
  1. The method asks the user to choose between the two options. It checks if the user input is an integer and, if valid, processes the choice.
  2. **Case 1 (File)**:
     + Prompts the user to enter a file path where the report should be saved.
     + Validates the file path format using the FilePathValidator class.
     + If the path is valid, it creates a FileReportGenerator object and starts a new thread to generate the report in the background.
     + The method waits for the report generation to complete and informs the user whether the report was successfully generated.
  3. **Case 2 (GUI)**:
     + Starts a new thread to generate and display the report using a graphical user interface with the help of a GUIReportGenerator object.
  4. **Default Case**:
     + Handles invalid choices by prompting the user to enter a valid option.
* **Error Handling**:
  1. If the user input for the report choice is not an integer, the method prompts the user to enter a valid choice (either 1 or 2).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The class **FileReportGenerator** (ALL DONE BY HANAN, ME)is responsible for generating a textual report of the library's book collection and saving it to a specified file (The file must be create by the user ). It uses a separate thread to generate the report, allowing the main program to continue running without being blocked by the file-writing operation. This design is useful for handling potentially long-running tasks like file I/O without freezing the user interface or main application flow. So,

\* **Attributes**:

* library: A reference to a LibraryClass object containing the collection of books.
* filePath: A string specifying the path where the report should be saved.
* reportGenerated: A boolean flag indicating whether the report has been successfully generated.

\* **Constructor**:

* The constructor initializes the library and filePath attributes with the provided values. It sets reportGenerated to false initially.

\* **run Method**:

* This method is executed when the FileReportGenerator is run as a thread. It handles the process of writing the report to the specified file.
* **File Path Validation**:
  + It first checks if the provided filePath is valid using the FilePathValidator class. If not, it prints an error message and exits.
* **Report Writing**:
  + It attempts to create a BufferedWriter to write the report to the specified file.
  + The report begins with a header ("Library Report") and a separator line.
  + It then iterates through the list of books in the library, writing each book's details to the file.
  + If the report is successfully written, the reportGenerated flag is set to true.
* **Error Handling**:
  + If an IOException occurs during the writing process, it catches the exception and prints an error message.

\* **isReportGenerated Method**:

* This method returns the value of the reportGenerated flag, indicating whether the report was successfully generated.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The class **FilePathValidator(ALL DONE BY ME , HANAN)** provides a utility method to validate file paths based on a specific pattern, particularly for Windows-style file paths. The purpose of this class is to ensure that file paths provided by users or other parts of the application are in a correct format before performing any file-related operations. This helps prevent errors and potential security issues related to file handling. So,

\* **Attributes**:

* FILE\_PATH\_PATTERN: A string that defines a regular expression pattern for a valid Windows file path. The pattern ensures that the path starts with a drive letter (e.g., "C:"), followed by backslashes and directory names, ending with a file name and extension.
* pattern: A Pattern object compiled from FILE\_PATH\_PATTERN for efficient matching.

\* **Method**:

* isValidFilePath(String filePath): This static method takes a file path as input and returns a boolean value indicating whether the path matches the defined pattern.
  + It uses the pattern object's matcher method to check if the input filePath conforms to the specified regular expression.
  + If the path matches, the method returns true; otherwise, it returns false.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_The class **GUIReportGenerator (ALL DONE BY ME, HANAN)** is designed to generate and display a graphical user interface (GUI) report of the library's books. The purpose is responsible for creating a GUI window that shows detailed information about the books in the library, including their images and details. This is useful for presenting data in a more user-friendly and visual format. So,

 **Attributes**:

* library: An instance of LibraryClass<Book> that holds the list of books to be displayed in the report.

 **Constructor**:

* Initializes the library attribute with the provided LibraryClass<Book> instance.

 **Method**:

* run(): This method is executed in a separate thread and is responsible for creating and displaying the GUI.
  + **SwingUtilities.invokeLater**: Ensures that the GUI creation and updates are done on the Event Dispatch Thread (EDT), which is the proper thread for Swing operations.
  + **JFrame**: Creates a window titled "Library Report".
    - Sets the default close operation to DISPOSE\_ON\_CLOSE, meaning the window will close when the user clicks the close button.
  + **JPanel**: Organizes the components in a vertical layout.
  + **Book Panels**: For each book, a JPanel is created to display the book's details and image.
    - **JLabel**: Displays the book's details, formatted with HTML to support line breaks.
    - **ImageIcon**: If an image path is available, the image is scaled and displayed alongside the book details.
  + **JScrollPane**: Allows scrolling if the list of books is too long to fit in the window.
  + **JButton**: Adds a "Close" button to close the window.
  + **Layout**: Adds components to the JPanel and sets the size of the frame.

Significant Topics Used in Hanan’s Work:

1. Exception Handling( isNumeric Method in IamLibararian mehod ,case 1, and the class FilePathValidator that used a way to handle the exception when the path of the I/O file is not correct, regular try-catch in LibraryReportSystem, if – else in the IamLibrarian method for checking the case if the user entered letters and characters,) *Methods to Print the Exception Information. (We used getMessage()).*
2. Collections and Data Structures (LinkedList, such lists of Book in my method line 188 , the list in the consumer sub class for borrowedBooks, ListOfCoupons).
3. Generic Class ( is used to implement sub class Librarian<T extends Book> that from any type that extends the class book , same for the consumer sub class ),
4. Java Swing (Displaying the general report for case 4 that implement in the class GUIReportGenerator , that used the creative way for inserting the pictures of the books and the information of the book).
5. Regular Expressions (Regex) (methods (!FilePathValidator.isValidFilePath(filePath) line 1367 in the pdf) .
6. *Access Control and Encapsulation (Accessor (get), Mutator (set)) and Constructors, ‘this’ Reference.. (in book class and the user class)*
7. Using super keyword in Librarian subclass and consumer subclass.
8. *Inheritance. (Librarian and consumer subclass) (extends)*
9. *Polymorphism. (used in the subclasses (@Override))*
10. Multithreading in the *case 1 in the LibraryReportSystem for creating and the FileReportGeneratorand and GUIReportGenerator that implements Runnable method to use the run method. Using the fileReportThread.start(); and the fileReportThread.join(); , used join so each thread finish its work and after it finished it will let the other thread to run the menu again with no interrupting.*
11. I/O write FileWriter*(filePath), BufferedWriter(writer) in FileReportGeneratorand case 1 in the LibraryReportSystem*

**Fahim Kabir:**

Description of my contribution in the project:

**LibraryClass<Book> Methods**

#### **Constructor**

* **Description**: The constructor of the LibraryClass<Book> initializes an empty LinkedList<Book>. This list will store Book objects, providing a flexible and efficient way to manage the library’s collection. The use of LinkedList allows for efficient insertions and deletions, making it ideal for dynamic collections where books may be frequently added or removed. By setting up this data structure, the library is prepared to handle operations such as adding new books, removing outdated ones, and managing the inventory dynamically.
* **Function**: Initializes the LinkedList<Book>.
* **Process**: Creates a new LinkedList<Book> instance to hold Book objects, preparing the library for operations such as adding, removing, or searching for books.

#### **Add a Book**

* **Description**: Adds a new Book object to the LinkedList<Book>. This method expands the library’s collection by incorporating new books. It allows the library to dynamically increase its inventory as new titles are introduced. By using the LinkedList, books are efficiently managed and the collection can grow as needed.
* **Function**: Accepts a Book object and adds it to the list.
* **Process**: Invokes the add method on the LinkedList to insert the Book object, thus updating the library’s inventory.

#### **Remove a Book**

* **Description**: Searches the LinkedList<Book> for a book by its title and removes it if found. This method helps manage the library’s collection by allowing for the removal of outdated, damaged, or unnecessary books. It ensures that the inventory remains current and relevant by efficiently handling deletions while providing feedback if the book is not found.
* **Function**: Searches for a book by title and removes it from the list.
* **Process**: Iterates through the LinkedList to find a book with the matching title. If found, uses the remove method to delete the book from the list and provides feedback if the book is not present.

#### **Search for a Book**

* **Description**: Searches the LinkedList<Book> for a book by its title and returns the Book object if found. This method facilitates quick access to specific books within the library’s collection. It allows users to efficiently locate books by title, which is useful for browsing and retrieval. If no matching title is found, the method returns null.
* **Function**: Searches for and returns a Book object by title.
* **Process**: Scans through the LinkedList to find a book with the specified title. Returns the Book object if found, or null if the title does not match any book in the list.

#### **Display Inventory**

* **Description**: Lists all books currently in the LinkedList<Book>, displaying their titles. This method provides an overview of the entire collection available in the library. It enables users to view all the books in the inventory and helps in browsing through the library’s collection. If the inventory is empty, the method notifies the user accordingly.
* **Function**: Displays the titles of all books in the inventory.
* **Process**: Iterates through the LinkedList and prints each book’s title. Notifies the user if the inventory is empty.

### **User Methods**

#### **Borrow a Book**

* **Description**: Allows a user to borrow a book from the library. This method updates the user’s record to reflect the borrowed book and marks the book as checked out. It ensures that borrowed books are removed from the library’s inventory and added to the user’s borrowed list, managing the temporary loan of books. If the book is unavailable, an appropriate message is displayed to the user.
* **Function**: Checks out a book to the user.
* **Process**: Searches for the book by title in the library’s inventory. If the book is available, it is added to the user’s borrowed list and removed from the library’s inventory. Displays a success message and an error if the book is not available.

#### **Return a Book**

* **Description**: Handles the process of returning a borrowed book. This method updates the library’s inventory to reflect the returned book and removes it from the user’s borrowed list. It manages the reintegration of books into the library’s collection and ensures that the user’s borrowed list is accurately updated. If the book is not found in the borrowed list, an error message is displayed.
* **Function**: Marks a borrowed book as returned.
* **Process**: Searches the user’s borrowed list for the book by title. Marks the book as returned and reinserts it into the library’s inventory. Displays a success message or an error if the book is not found in the borrowed list.

#### **Display Borrowed Books**

* **Description**: Displays a list of books currently borrowed by the user. This method provides an overview of the books a user has checked out, helping them track their borrowed items. If no books are currently borrowed, the method notifies the user accordingly.
* **Function**: Lists all books borrowed by the user.
* **Process**: Checks the user’s borrowed list and prints the titles of all borrowed books. Notifies the user if no books are currently borrowed.

#### **Buy a Book**

* **Description**: Allows a user to purchase a book and deduct its price from their account balance. This method handles the transaction process, including verifying the user’s balance, updating the library’s inventory, and managing the book’s purchase count. It ensures that the user has sufficient funds and updates both the inventory and the user’s balance. If the book is not found or the user has insufficient funds, appropriate messages are displayed.
* **Function**: Purchases a book and updates the user’s balance.
* **Process**: Searches for the book by title. If the book is available and the user has enough funds, deducts the book’s price from the user’s balance, removes the book from the inventory, and increments the purchase count. Displays success or error messages as needed.

### **Menu and Option Methods**

#### **IamConsumer**

* **Description**: Manages the main menu interface for a logged-in user, providing access to various actions related to library management and account settings. This method serves as the central navigation point, allowing users to perform actions such as buying, borrowing, and returning books, as well as managing their account details. It handles multiple user actions until the user decides to exit or delete their account.
* **Function**: Displays the main menu and handles user selections.
* **Process**: Presents a menu with options for different actions. Processes user input to perform actions such as buying, borrowing, returning books, and managing account settings. Continues to handle user actions until the user exits or deletes their account.

#### **optionOne**

* **Description**: Handles the process for purchasing a book, including checking availability, verifying user balance, and managing coupons. This method facilitates book purchases by checking if the book is in stock, verifying that the user has sufficient funds, and updating the user’s spending. It also manages any applicable coupons and displays relevant messages to the user.
* **Function**: Executes the purchase of a book.
* **Process**: Prompts the user for the book title, checks if the book is available and if the user has sufficient funds. Deducts the book’s price from the user’s balance and updates the inventory. Handles coupon management and displays relevant success or error messages.

#### **optionTwo**

* **Description**: Manages the borrowing of a book by checking its availability and updating the user’s borrowed list. This method facilitates the borrowing process but may not fully integrate with the user’s borrowed list. It allows users to check out books and handles the corresponding updates in the library’s inventory.
* **Function**: Initiates the borrowing process.
* **Process**: Prompts the user for the book title, checks availability, and marks the book as borrowed. Currently, it may not fully integrate with the user's borrowed book list.

#### **optionThree**

* **Description**: Manages the return of a borrowed book, updating both the user’s record and the library’s inventory. This method handles the return process, ensuring that books are marked as returned and reinserted into the library’s collection. It also updates the user’s borrowed list accordingly.
* **Function**: Processes the return of a borrowed book.
* **Process**: Prompts the user for the book title, checks the borrowed list, marks the book as returned, and reinserts it into the library’s inventory. Assumes that borrowed books are tracked elsewhere in the system.

#### **optionFive**

* **Description**: Provides search functionality for books by various criteria such as title, author, genre, or ISBN. This method enhances the ability to locate specific books based on different search parameters. It allows users to search for books using various criteria and displays details of found books or a message if no matches are found.
* **Function**: Executes book searches based on user criteria.
* **Process**: Provides search options for title, author, genre, or ISBN. Searches the library’s inventory and displays details of found books or a message if no matches are found.

#### **optionSix**

* **Description**: Displays all books available in the library, showing the complete inventory. This method calls the displayInventory method of LibraryClass to list all books in the library, providing a comprehensive view of the available collection.
* **Function**: Shows the full inventory of books.
* **Process**: Calls the displayInventory method of LibraryClass to list all books in the library.

### **Helper Methods**

#### **searchBookByTitle**

* **Description**: Searches for a book by its title within the LinkedList<Book>. This method enables users to locate a specific book by its title quickly, enhancing the search functionality within the library’s inventory. It returns the Book object if found or null if no match is found.
* **Function**: Retrieves a Book object by title.
* **Process**: Iterates through the LinkedList to find a book with the specified title. Returns the Book object if found, otherwise returns null.

#### **searchBookByAuthor**

* **Description**: Searches for books authored by a specific individual within the LinkedList<Book>. This method enhances the library’s search capabilities by allowing users to find books written by a particular author. It returns a list of books that match the specified author.
* **Function**: Retrieves books by the specified author.
* **Process**: Iterates through the LinkedList to find books authored by the specified individual. Returns a list of matching books.

#### **searchBookByGenre**

* **Description**: Searches for books belonging to a specific genre within the LinkedList<Book>. This method helps users locate books categorized under a particular genre, improving the ability to find books based on genre preferences. It returns a list of books that match the specified genre.
* **Function**: Retrieves books by the specified genre.
* **Process**: Iterates through the LinkedList to find books that match the specified genre. Returns a list of matching books.

#### **searchBookByIsbn**

* **Description**: Searches for a book by its ISBN (International Standard Book Number), a unique identifier for books, within the LinkedList<Book>. This method allows for precise searches using the book’s unique ISBN, ensuring accurate identification and retrieval of the book. It returns the Book object if found.
* **Function**: Retrieves a book by its ISBN.
* **Process**: Iterates through the LinkedList to find a book with the specified ISBN. Returns the Book object if found, otherwise returns null.

***Significant Topics used in Fahim’s work:***

**Inheritance and Polymorphism :** Inheritance is used in creating specialized types of users or books. Polymorphism Allows methods to handle objects of different subclasses through a common interface or base class. For example, methods like borrowBook or returnBook in the LibraryClass can be implemented to accommodate different types of Book or User objects, leveraging the ability to use a single method to handle various object types.

**Generics**: Utilized in the LibraryClass<Book>, allowing it to handle any type of Book objects without needing typecasting. Methods such as addBook, removeBook, and searchBook in LibraryClass<Book> are type-safe and flexible, operating on the generic Book type.

**Exception Handling**: : In the LibraryClass, methods like searchBook might throw exceptions if the book is not found.In the IamConsumer class, exception handling ensures that errors like attempting to borrow a book when the user’s balance is insufficient are managed gracefully, displaying appropriate error messages.

**Collections and Data Structures:** LinkedList is used to store and manage collections of Book objects and User accounts in LibraryClass. Methods such as addBook, removeBook, and displayInventory rely on LinkedList to efficiently handle dynamic changes in the library's collection.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Lynn Hajj Hassan:**

\_Class Coupon ( All of it ) :

#### **Description**

The Coupon class provides methods to access, modify, display coupon details, and calculate the discounted price of an item.

#### **Fields**

* **expDate**: Expiration date of the coupon.
* **couponCode**: Unique coupon code.
* **amountToReduce**: Discount percentage offered by the coupon.

#### **Constructor**

Initializes a new Coupon with the given expiration date, code, and discount percentage.

#### **Methods**

* **Getters**
  + getExpDate(): Returns the expiration date.
  + getCouponCode(): Returns the coupon code.
  + getAmountToReduce(): Returns the discount percentage.
* **Setters**
  + setExpDate(String expDate): Sets the expiration date.
  + setCouponCode(String couponCode): Sets the coupon code.
* **displayCouponInfo(JTextArea textArea)**: Displays the coupon details in the provided text area.
* **calculateNewPrice(double price, double amountToReduce)**: Calculates and returns the new price after applying the discount

\_CouponSystemManagement class ( All of it ) :

#### **Description**

The CouponsSystemManagement class designed to manage a collection of coupons like supports adding, removing, and displaying coupons. This class uses a linked list to store the coupons and provides methods to manipulate and display the list.

#### **Fields**

* **couponList**: LinkedList<T> - A linked list to store coupons.**Constructor**

CouponsSystemManagement():Initializes a new empty linked list of coupons.

#### **Methods**

* **addCoupon(T coupon)** Adds the specified coupon to the list.
* **removeCoupon(T coupon)** Removes the specified coupon from the list if it exists.
* **displayCoupons(JTextArea textArea)** Clears the provided text area and appends the details of all coupons in the list to the text area.

\_CouponDisplayPanel class(All of it) :

#### **Description**

The CouponDisplayPanel class extends JPanel and provides a graphical user interface (GUI) for displaying coupons. It integrates with the CouponsSystemManagement system to show a list of coupons in a text area, allowing users to view the details of all available coupons. Additionally, it provides a static method to generate and add welcome coupons to the system.

#### **Fields**

* **couponsSystem**: CouponsSystemManagement<Coupon> - A reference to the coupons management system that handles the collection of coupons.
* **couponTextArea**: JTextArea - A text area for displaying the details of coupons.

#### **Constructor**

Initializes a new CouponDisplayPanel with the specified coupons management system. The constructor sets up the panel layout, initializes the text area with a scroll pane, and displays the coupons automatically.

#### **Methods**

* **generateWelcomeCoupons(CouponsSystemManagement<Coupon> couponsSystem)**: A static method that generates a welcome coupon and adds it to the specified coupons management system.

\_\_UserAccountSystem ( All of it) :

#### **Description**

manages a list of user accounts in a library system. This class supports both librarian and customer accounts and includes methods for handling account balances and user authentication.

#### **Fields**

* **accountsList**: LinkedList<T> - A linked list to store user accounts, where T is a type that extends the User class.

#### **Constructor**

Initializes a new UserAccountsSystem with an empty linked list of accounts.

#### **Methods**

* **addAccount(T account)**: Adds a new account to the accounts list.
* **removeAccount(T account)**: Removes an account from the list by account ID. This operation is only allowed for librarians. It prompts the user to confirm the deletion.
* **withdrawMoney(T account, double amount)**: Subtracts a specified amount of money from the user's account balance when they buy or borrow a book.
* **depositMoney(T account, double amount)**: Adds a specified amount of money to the user's account balance.
* **changeName(T account, String newName)**: Updates the name of the specified **changePhoneNumber(T account, String newPhoneNum)**: Updates the phone
* **changePassword(T account, String newPassword)**: Updates the password of the specified account.
* **changeEmail(T account, String newEmail)**: Updates the email of the specified account.
* **changeAdress(T account, String newAddress)**: Updates the address of the specified account.
* **login(String email, String password)**: Finds and returns a user account based on the provided email and password.
* **accountIsAvailable(String email)**: Checks if an account with the specified email exists in the system.

**\_Setting up accounts, creating accounts, login accounts :**

Features handled inside LibraryProject in the main method by Lynn, like letting the user creates an account and login .

* Creating the Account System List
* Register Case Added to Handle Accounts
* Adding Accounts to the System
* Creating Coupons System
* Main Loop for User Interaction ( login , creates account , exit )
* Utility Methods for Input Validation and User Interaction( isNumeric, isValidEmail , isValidPhoneNumber , getPhoneNumber, etc...)

All from “ UserAccountsSystem<User> userAccountSystem = new UserAccountsSystem<>(); “

Till “ System.***out***.println("Check your account for more details."); “

**Inside the IamConsumer method: Implemented Case 4 for User Account Information Access**

In this case, optionFour method was used to display a menu for the user, allowing them to view and modify their account settings and information.

**Handling Coupons During Purchases**

In optionOne (implemented by Fahim, with the coupon functionality integrated by me), the coupon handling code is found between:

* System.out.print("Do you have a coupon? (yes/no): ");
* System.out.println("Invalid coupon code.");

Significant Topics Used in Lynn’s Work:

Exception Handling( isNumeric Method, removing accounts/coupons and handling when they don’t exists…), Collections and Data Structures ( LinkedList, such lists of accounts, coupons…), Generic Class( such UserAccountSystem ), Java Swing (Displaying coupons) , Regular Expressions (Regex) ( isValidPhoneNumber , isValidEmail methods ) .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Output Example Screen Shoots :** A screenshot of a computer

Description automatically generated

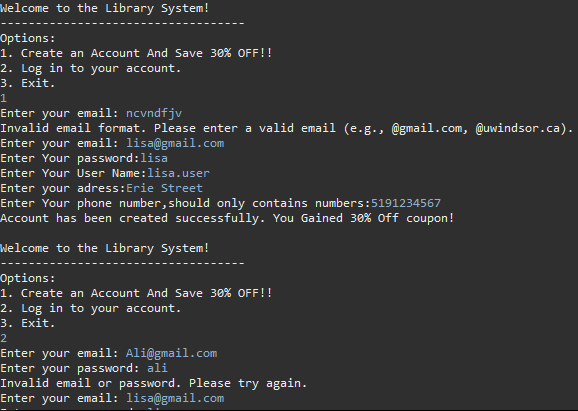
A screenshot of a book

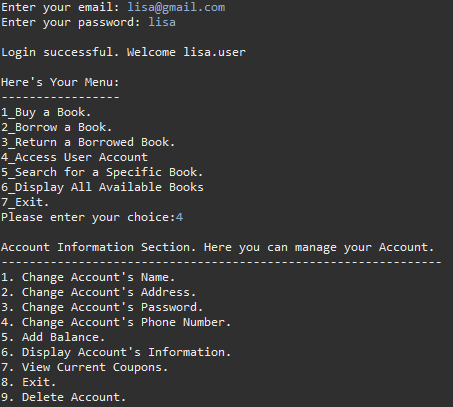
Description automatically generatedA white screen with a black background

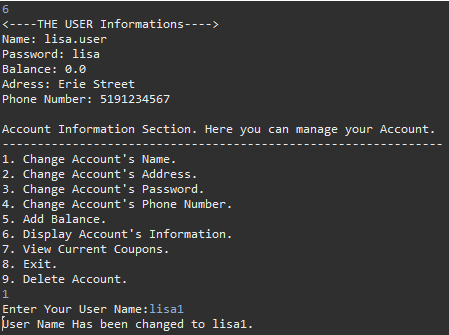
Description automatically generated with medium confidenceA screenshot of a coupon display

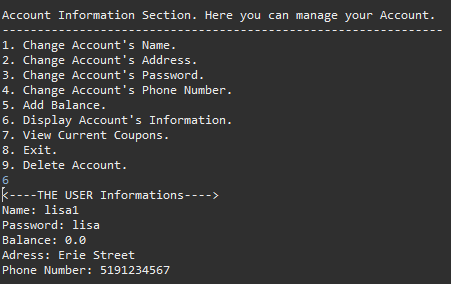
Description automatically generated

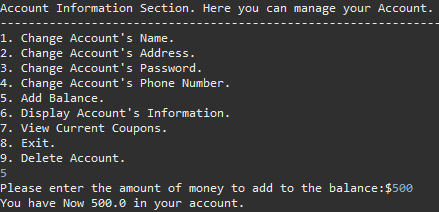
Output example :

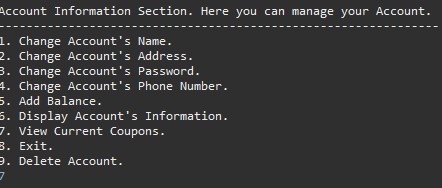


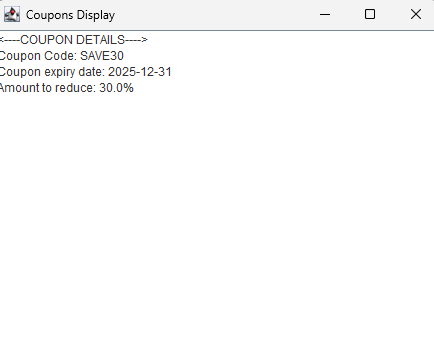


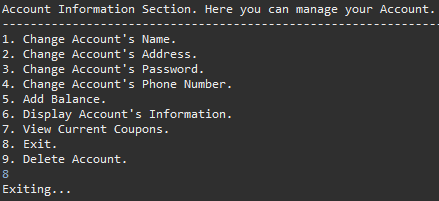


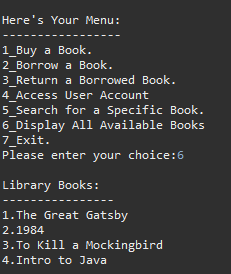


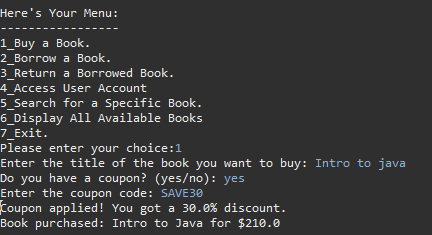


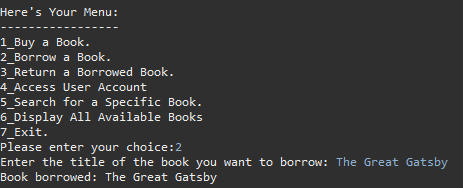




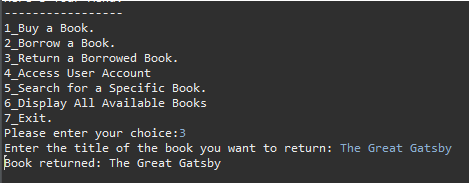




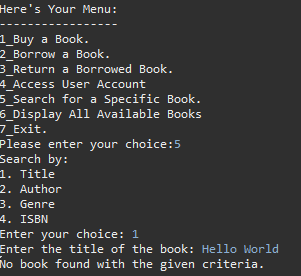


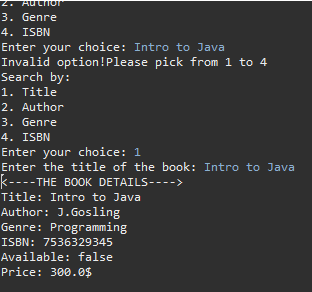


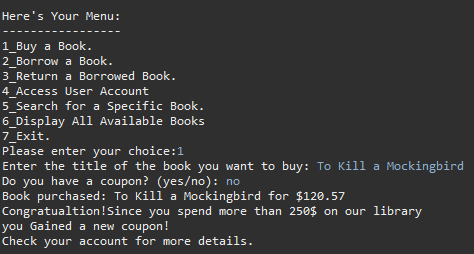
Buying the book a shown before is not for free, however borrowing it is free.



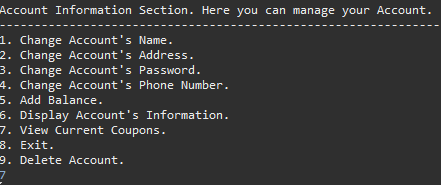
The user here returned the book .



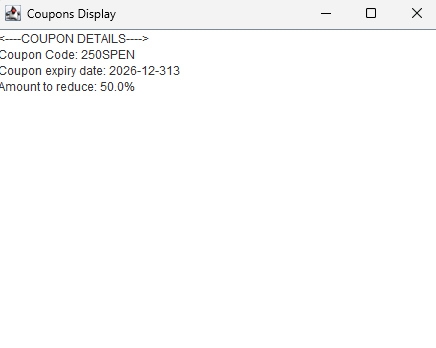




After the user spend more that 250$ in the library , they will get a new coupon!



The coupons will appear in option 7 in Access Account info section .



The new coupon appeared in the view coupon section .And the user can use it in the next purchase .

+ Many other features .

When one of the librarians enters their email ( which is built in already inside the code:

 The code will take the user to a librarian menu .

A screenshot of a computer

Description automatically generated

This Menu has different options than a normal user(Customer /Consumer) .

A screenshot of a computer screen

Description automatically generated

Generating a Report will display the books and their current status to the Librarians.

The librarian above librarian added a new book called “Intro to life”.

If The librarian wants to see current books and their status, they will have to access option 3 “Generate Report “ . A screenshot of a computer code

Description automatically generated

Choosing option 1 : The report will be written to the path specified by the librarian . As Shown below .

A screenshot of a computer

Description automatically generated

If the librarian choose GUI option , the library’s books details will be shown as Panel. A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generatedand if she want to remove a book

she will see that is got removed also from the report with its image

A screenshot of a computer

Description automatically generated

Here you can see that if she tried to enter something invalid, she will have error messages

A screenshot of a computer

Description automatically generated

You can see that it will not except what she wrote

A screenshot of a computer program

Description automatically generated

This is showing if she tried to chose something not 1 or 2 in the general report whether a different number for letter.

A screenshot of a computer

Description automatically generatedhere will handle the exception in case the user entered incorrect path for the file, whether a digits or letters

here will show that if the user entered incorrect title that does not exist , it will tell her by a message that it is not found, which mean not exist

A screenshot of a computer

Description automatically generated

This when the librarian want to logout , it will return him to the main menu.

A screenshot of a computer

Description automatically generated

**Code References :**

1-

code, A. (2020, January 29). *Show images in text pane java swing gui*. YouTube. <https://youtu.be/vx2rPM9Ns9c?si=XLDzPOQgaWN7vtWV>

(code, 2020)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2-

overflow, S. (1958, March 1). *Check if a path represents a file or a folder*. StackOverflow. <https://stackoverflow.com/questions/12780446/check-if-a-path-represents-a-file-or-a-folder>

(overflow, 1958)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3-

Java. (n.d.). *Lesson: Regular expressions*. Lesson: Regular Expressions (The JavaTM Tutorials > Essential Java Classes).<https://docs.oracle.com/javase/tutorial/essential/regex/>

(Java)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4-

platform, java. (2024, June 5). Pattern (java platform SE 8 ). <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>

(platform, 2024)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5-

java. (2024, June 5). *Class Matcher*. Matcher (Java Platform SE 8 ). <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html>

(2024)