# Library Management System

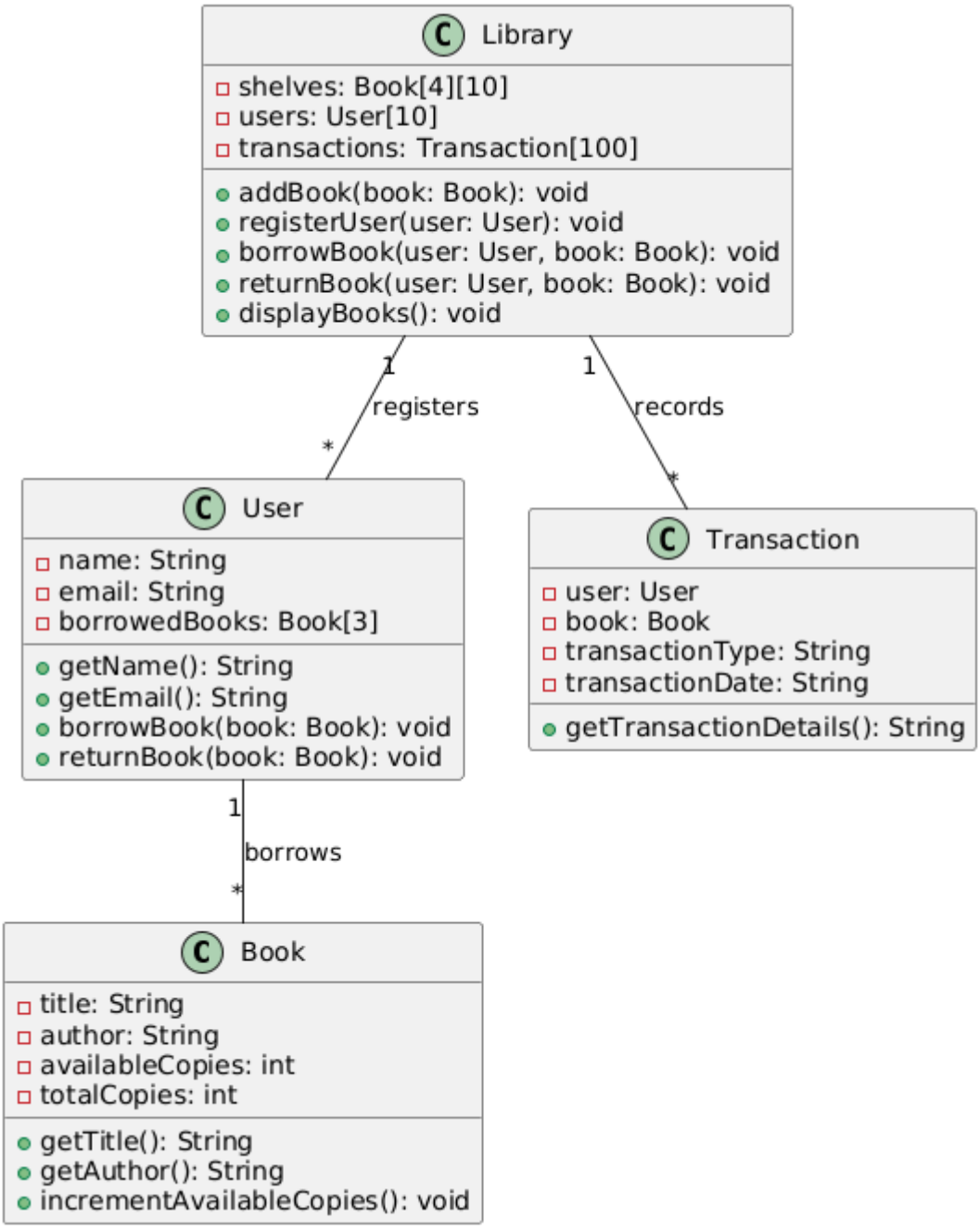
## Introduction

A Library Management System (LMS) is a software solution designed to facilitate the management of library resources, patrons, and their interactions with the library. With advancements in technology, it has become essential to automate the process of book borrowing, returning, and keeping track of the available books. A Library Management System helps streamline the daily operations of libraries, ensuring better organization and access to resources. This system provides functionality for librarians to manage book inventories, users, and transactions, while also ensuring user-friendly interactions for members.

This project involves developing a Java-based Library Management System that allows for:

- **Book Management** (Adding books, viewing books).
- **User Registration** (Registering library users).
- **Transaction Management** (Borrowing and returning books).
- **Transaction History** (Tracking borrowed and returned books). The system also ensures that all users are within borrowing limits and that books are available

before borrowing. The overall goal is to create a user-friendly, reliable, and efficient system for managing library resources.



## Objectives

The primary **objective** of this project is to design and implement a **Library Management System** to automate and simplify the manual library management tasks. The objectives include:

 **Develop a System to Handle Book Management**:

- Add new books to the library.
- Track available and total copies of each book.
- Display a list of all books in the library.

 **Allow User Registration**:

- Register new users by their name and email.
- Ensure no duplicate email entries.

 **Manage Borrowing and Returning of Books**:

- Allow users to borrow books (with validation on available copies).
- Track the borrowing history of each user.
- Allow users to return borrowed books.
- Ensure users cannot borrow more than three books at once.

 **Transaction Recording**:

- Log each transaction, capturing user, book, transaction type, and date.
- View the history of transactions.

 **Maintain Library Shelves**:

- Organize books in shelves, each with a fixed capacity.
- Ensure that books are placed correctly and that new shelves are created as necessary when books exceed shelf space.

 **User Interface**:

- Create an interactive menu system to handle librarian tasks.
- Provide feedback messages to guide the librarian and users.

# Goals

The goals of this project are:

- **Efficient Library Management**: Automate book management, user registration, borrowing, and returning to improve efficiency.
- **User-Friendly**: Provide an intuitive user interface for both the librarian and the users.
- **Real-Time Updates**: Ensure real-time tracking of available books and transactions.
- **Error-Free Transactions**: Ensure that all transactions (borrowing and returning) are error-free by incorporating validation checks.
- **Extensibility**: Design the system in a modular way that allows easy updates and improvements in the future.

## System Design and Architecture

In this section, we will outline the overall architecture of the Library Management System (LMS), the key components and their responsibilities, as well as how they interact with each other.

---

## 1. Overview of System Design

The **Library Management System** follows a **modular** design, meaning each component or module is responsible for a specific task within the system. The system is organized into several main classes and methods, which communicate with each other to handle various functionalities like adding books, registering users, borrowing and returning books, and tracking transactions.

The system has the following primary components:

- **Book**: Represents the individual books in the library.
- **User**: Represents the library members who borrow books.
- **Transaction**: Records borrowing or returning of books.
- **Library**: Manages the collection of books, users, and transactions.
- **Main Class**: Contains the entry point to the system and handles interactions with the librarian.

## 2. Key Components and Their Responsibilities

1. **Book Class**:
    - The Book class manages the attributes of each book, including title, author, total copies, and available copies.
    - It includes methods for checking availability, updating available copies when a book is borrowed or returned, and providing book details.
2. **User Class**:

3.
    - The User class holds information about the library members, including their name, email, and a list of borrowed books.
    - This class ensures users cannot borrow more than 3 books at once and handles the logic for borrowing and returning books.
4. **Transaction Class**:
    - The Transaction class logs the borrowing or returning of books.
    - It records the user involved, the book borrowed or returned, the type of transaction (borrow/return), and the date.
5. **Library Class**:
    - The Library class is the core of the system, managing the books, users, and transactions.
    - It organizes books into shelves (with a fixed capacity per shelf).
    - It manages the book borrowing and returning process, ensuring that only available books can be borrowed.
    - It keeps track of users and their borrowing history, and it maintains a record of all transactions.
6. **Main Class**:
    - The Main class is the interface through which the librarian interacts with the system.
    - It presents a menu for performing various operations, such as adding books, registering users, borrowing/returning books, viewing all books, and viewing transaction records.

---

## 3. UML Class Diagram

Here's an overview of the UML diagram we used to structure the system:

- **Book** is connected to **Transaction** and **User** classes through relationships.
- **User** can borrow many **Books** (1-to-many relationship).
- **Library** manages the overall functionality of adding books, registering users, and logging transactions.
- **Transaction** logs each action of borrowing or returning a book, linking both the **User** and **Book**.

---

## 4. System Architecture

The system architecture is designed to follow a simple, **client-server** like structure:

- The **Client** in this case is the librarian or user interacting with the system through the **Main Class**.
- The **Server** (core system) is represented by the **Library**, **Book**, **User**, and **Transaction** classes, which handle data storage, processing, and logic execution.

*Key Functional Flow:*

1. **Book Management**:
    - The librarian can add new books to the system, which are stored on specific shelves in the library.
    - Each book is identified by its title, author, total copies, and available copies.
2. **User Management**:
    - Users can be registered in the system. Each user is allowed to borrow up to 3 books at a time.

3. **Borrowing/Returning Books**:
    - A user can borrow a book if it's available. The system tracks this by decrementing the available copies and storing the transaction.
    - When a user returns a book, the system updates the available copies and records the return transaction.
4. **Transaction Logging**:
    - Each transaction is logged, providing transparency on when books were borrowed or returned and by which user.

---

## 5. Data Storage

- **In-Memory Storage**:
    - The system uses **arrays** for storing users, books, and transactions in memory.
    - For book storage, books are placed in **shelves**, each represented as an array with a fixed capacity.
    - Users and transactions are stored in simple arrays with a maximum size.

---

## 6. Design Considerations

- **Scalability**: Although the current design uses arrays with fixed sizes, the system could be further improved by implementing dynamic data structures like **ArrayList** or using a database for persistent storage.
- **Validation and Error Handling**: The system includes various checks to prevent errors like borrowing unavailable books or duplicating user email addresses.
- **Modularity and Extensibility**: The design is modular, with each class being responsible for specific functionalities. This makes the system easy to extend, such as adding new features like book search or overdue book management.

# Main class :

```
// Intiazing the Library
Library library = new Library();
while (true) {
    Scanner sc = new Scanner(source:System.in);

    System.out.println("*   The Menu   *\n"
            + "1- Add n new book.\n"
            + "2- Register users.\n"
            + "3- Borrow books.\n"
            + "4- Return books.\n"
            + "5- Display all the books.\n"
            + "6- View all transactions.\n"
            + "7- Exit the program.\n");

    String chose = sc.next();
    if (chose.equals(anObject: "1")) {
        adding_book(lib:library);
    } else if (chose.equals(anObject: "2")) {
        Register_User(lib:library);
    } else if (chose.equals(anObject: "3")) {
        borrowBook(lib:library);
    } else if (chose.equals(anObject: "4")) {
        return_book(lib:library);
    } else if (chose.equals(anObject: "5")) {
        display_books(lib:library);
    } else if (chose.equals(anObject: "6")) {
        display_transactions(lib:library);
    } else if (chose.equals(anObject: "7")) {
        System.out.println(x: "\n************************************");
        System.out.println(x: "**** Program exited succesfully ! ****");
        System.out.println(x: "************************************\n");
        break;
    } else {
        System.out.println(x: "Please enter a valid number! (1,2,3,4,5,6,7^\n)");
    }
```

## Explanation:

1. **Library Object Creation**:
   - The program starts by creating a Library object, which will hold and manage the books, users, and transactions in the system.
2. **Menu Loop**:
   - A while(true) loop is used to continually display the menu until the librarian chooses to exit. This loop ensures that the user can perform multiple operations without restarting the program.
3. **Scanner for User Input**:
   - A Scanner object is used to read the librarian's input from the console. It waits for the librarian to input a choice from the menu.
4. **Displaying the Menu**:
   - A text-based menu is printed out to the console, listing all the available operations:
     1. Add a new book.
     2. Register a new user.
     3. Borrow a book.
     4. Return a book.
     5. Display all books.
     6. View all transactions.
     7. Exit the program.
5. **Processing the Input**:
   - The librarian enters a choice (1 to 7). The program then evaluates the input:
     - If the input matches "1", it calls the adding_book() method to add a new book to the library.
     - If the input is "2", the Register_User() method is called to register a new user.
     - If the input is "3", the borrowBook() method is called for borrowing a book.
     - If the input is "4", the return_book() method is invoked for returning a borrowed book.
     - If the input is "5", the display_books() method displays all the books in the library.

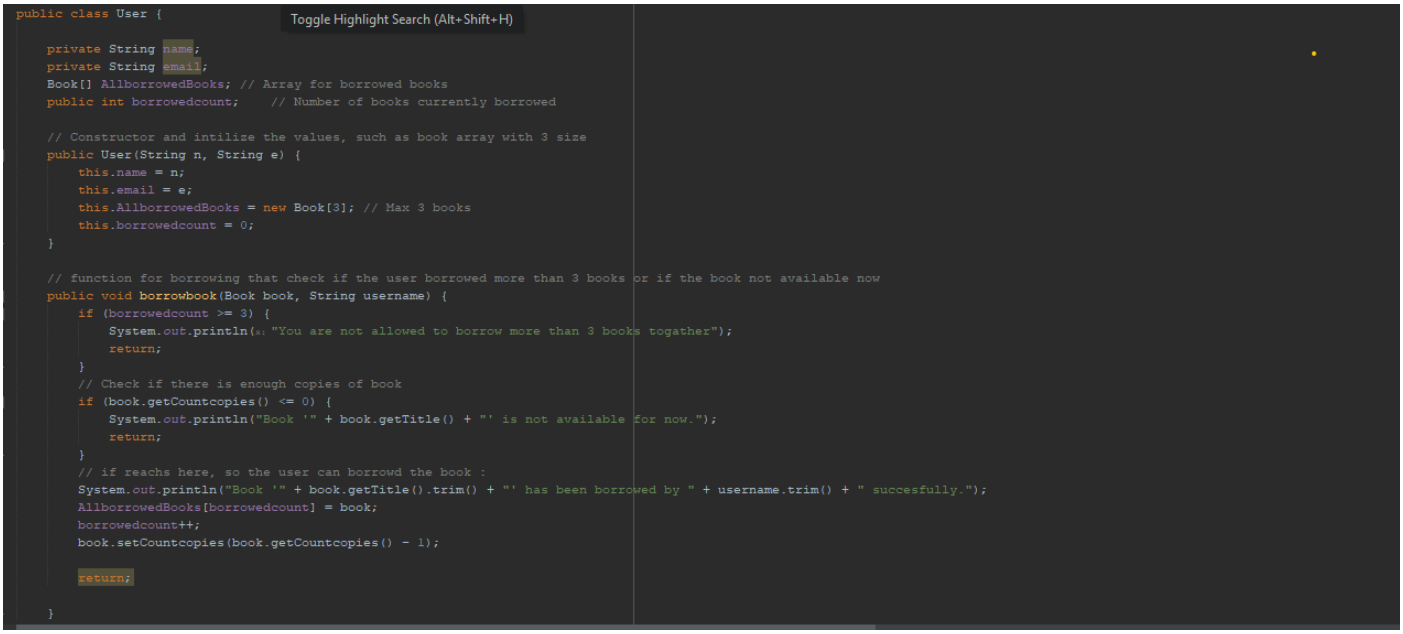
- If the input is "6", the display_transcactions() method shows all the transactions that have occurred in the system.
- If the input is "7", the program exits by printing a closing message and breaking out of the loop.

6. **Error Handling**:
   - If the librarian enters an invalid option (anything other than 1 to 7), the program displays a message asking them to enter a valid number, and the loop continues.

```
public class Library {

    private Book[][] shelves;            // 2D array for shelves
    private User[] users;                // Array for registiring every student
    private Transaction[] transactions;  // Array to store all transactions
    public static int bookCount;         // Total number of books in the library
    public static int userCount;         // Total number of registered users
    private int transactionCount;        // Total number of transactions

    // Constructor and intilize the values of Shelves with capasity 40, with 100 users, and 250 transcations etc..
    public Library() {
        this.shelves = new Book[10][4];
        this.users = new User[100];
        this.transactions = new Transaction[250]; //
        this.bookCount = 0;
        this.userCount = 0;
        this.transactionCount = 0;
    }

    public User[] getUsers() {
        return users;
    }

    // Add book fuction :
    public void AddBook(Book book) {
        // Checi if this book actully exict :
        for (int i = 0; i < shelves.length; i++) {
            for (int j = 0; j < 4; j++) {
                if (shelves[i][j] != null && shelves[i][j].getAuthor().trim().equalsIgnoreCase(anotherString:book.getAuthor().trim()) && shelves[i][j].getTitle().trim().equalsIgnoreCa
                    shelves[i][j].updatecount(newnumber: book.getTotalcopies());
                    System.out.println(x: "This book is already exist in the library, so we increased the total copies");
                    return;
                }
            }
```

## Explanation of the Library Class:

1. **Attributes (Variables)**:
   - **shelves (2D array)**:
     - This is a 2-dimensional array of Book objects. It represents the physical shelves in the library. The library initially has 10 shelves, each with a capacity to hold 4 books.
   - **users (Array)**:
     - This array holds User objects, which represent the registered users of the library. The array is initially sized to hold 100 users.
   - **transactions (Array)**:
     - This array holds Transaction objects that keep track of the borrowing and returning operations in the library. It is initially sized to hold up to 250 transactions.
   - **bookCount (Static)**:
     - This is a static variable that keeps track of the total number of books in the library. It is shared across all instances of the Library class.
   - **userCount (Static)**:
     - This is a static variable that tracks the number of registered users. It is shared across all instances of the Library class.
   - **transactionCount**:
     - This variable tracks the total number of transactions that have occurred in the library.
2. **Constructor**:
   - The constructor initializes the Library object. When an instance of the Library class is created:
     - **Shelves** are initialized with 10 rows (representing shelves) and 4 columns (representing slots for books per shelf).
     - **Users** and **Transactions** arrays are set up with their respective initial capacities (100 users and 250 transactions).
     - The counts for books, users, and transactions are set to zero.
3. **Getter Method** (getUsers()):
   - This method returns the array of users in the library, allowing other parts of the program to access the list of registered users.



```java
public class User {
    Toggle Highlight Search (Alt+Shift+H)

    private String name;
    private String email;
    Book[] AllborrowedBooks; // Array for borrowed books
    public int borrowedcount;    // Number of books currently borrowed

    // Constructor and intilize the values, such as book array with 3 size
    public User(String n, String e) {
        this.name = n;
        this.email = e;
        this.AllborrowedBooks = new Book[3]; // Max 3 books
        this.borrowedcount = 0;
    }

    // function for borrowing that check if the user borrowed more than 3 books or if the book not available now
    public void borrowbook(Book book, String username) {
        if (borrowedcount >= 3) {
            System.out.println("You are not allowed to borrow more than 3 books togather");
            return;
        }
        // Check if there is enough copies of book
        if (book.getCountcopies() <= 0) {
            System.out.println("Book '" + book.getTitle() + "' is not available for now.");
            return;
        }
        // if reachs here, so the user can borrowd the book :
        System.out.println("Book '" + book.getTitle().trim() + "' has been borrowed by " + username.trim() + " succesfully.");
        AllborrowedBooks[borrowedcount] = book;
        borrowedcount++;
        book.setCountcopies(book.getCountcopies() - 1);

        return;

    }
```

## Explanation of the User Class:

1. **Attributes**:
    - **name (String)**: The name of the user.
    - **email (String)**: The email of the user.
    - **AllborrowedBooks (Array of Book)**: An array to store up to 3 books that the user has borrowed.
    - **borrowedcount (int)**: A counter for the number of books currently borrowed by the user (max 3).
2. **Constructor**:
    - Initializes the User object with a name and email.
    - Initializes AllborrowedBooks with a size of 3 to store borrowed books.
    - Sets the borrowedcount to 0, indicating no books have been borrowed initially.
3. **borrowbook(Book book, String username) Method**:
    - Ensures the user can borrow a book only if they have not already borrowed 3 books.
    - Checks if the book is available (i.e., has copies remaining).
    - If the book is available, it allows the user to borrow it, adds it to the AllborrowedBooks array, increases the borrowedcount, and reduces the book's available copies.
4. **returnBook(Book book) Method**:
    - This method allows the user to return a borrowed book.
    - It checks the AllborrowedBooks array for the specified book and removes it if found.
    - If the book is not found in the array, it prints a message indicating the book is not in the borrowed list.

## Summary:

The User class manages user details, borrowed books, and their count. The borrowbook() method ensures a user can borrow a book only if they haven't exceeded the limit and if the book is available. The returnBook() method allows a user to return a book, removing it from their borrowed list.

```java
public class Transaction {

    private User user;              // The user
    private Book book;              // The book
    private String transactionType; // "Borrow" or "Return"
    private String transactionDate; // The date of the transaction

    // Constructor
    public Transaction(User u, Book b, String ttype) {
        this.user = u;
        this.book = b;
        this.transactionType = ttype;
        this.transactionDate = getNowTime(); //
    }
    // function for writing the date in formal style :
    private String getNowTime() {
        SimpleDateFormat formatter = new SimpleDateFormat(pattern:"dd/MM/yyyy HH:mm:ss");
        return formatter.format(new Date());
    }
    // function that prapre every transcation as string and return it :
    public String getTransactionDetails() {
        String result = transactionType + " Transaction: "
                + user.getName().trim() + " (" + user.getEmail().trim() + ") "
                + transactionType.toLowerCase() + "ed '" + book.getTitle().trim() + "' by " + book.getAuthor().trim()
                + " on " + transactionDate + ".";

        return result;
    }

}
```

## Explanation of the Transaction Class:

1. **Attributes**:
   - **user (User)**: The User object associated with the transaction (who is borrowing or returning the book).
   - **book (Book)**: The Book object involved in the transaction (the book being borrowed or returned).
   - **transactionType (String)**: The type of transaction ("Borrow" or "Return").
   - **transactionDate (String)**: The date and time when the transaction occurs.
2. **Constructor**:
   - Initializes the Transaction object with a User, Book, and a transactionType ("Borrow" or "Return").
   - The transactionDate is set to the current date and time using the getNowTime() method.
3. **getNowTime() Method**:
   - This private method formats and returns the current date and time in the format dd/MM/yyyy HH:mm:ss using SimpleDateFormat. This provides a timestamp for when the transaction occurred.
4. **getTransactionDetails() Method**:
   - Prepares the transaction details as a string and returns it.
   - The string contains:
     - The type of transaction ("Borrowed" or "Returned").
     - The user's name and email.
     - The book's title and author.
     - The date and time of the transaction.

## Summary:

The Transaction class manages the details of each transaction in the library system (either borrowing or returning a book). It stores information about the user, the book, the transaction type, and the date of the transaction. The getTransactionDetails() method generates a formatted string that provides the complete details of the transaction.

```java
public class Book {

    private String Title;
    private String Author;
    public int Countcopies;
    private int Totalcopies;

    // Constructor
    public Book(String t, String a, int ct, int tc) {
        Title = t;
        Author = a;
        Countcopies = ct;
        Totalcopies = tc;
    }

    // function for writing the details of the book
    public String writedetails() {
        String result = Title.trim() + ", " + Author.trim() + ", " + Countcopies + " copies available.";
        return result;
    }

    // updating the number of copies if we want to increase or decrease the number of copies
    public void updatecount(int newnumber) {
        // check that the entered number is valid, Like not nagative for example.
        if (newnumber < 0) {
            System.out.println(x: "Please try again with valid number!");
            return;
        }
        this.Countcopies += newnumber;
        this.Totalcopies += newnumber;

    }
```

## Explanation of the Book Class:

1. **Attributes**:
    - **Title (String)**: The title of the book.
    - **Author (String)**: The author of the book.
    - **Countcopies (int)**: The number of available copies of the book that can be borrowed.
    - **Totalcopies (int)**: The total number of copies the book has in the library (including those currently borrowed).
2. **Constructor**:
    - Initializes the Book object with the book's title, author, the number of available copies (Countcopies), and the total number of copies (Totalcopies).
3. **writedetails() Method**:
    - Returns a string that contains the book's details, including the title, author, and the number of available copies.
    - The details are formatted as: "Title, Author, X copies available." where X is the available copy count.
4. **updatecount(int newnumber) Method**:
    - This method updates the number of available copies (Countcopies) and the total copies (Totalcopies) based on the newnumber parameter.
    - It checks if the newnumber is a valid (non-negative) number. If it's invalid (negative), it prints an error message and exits the method.
    - The copies are updated by adding the newnumber to both Countcopies and Totalcopies.

## Summary:

The Book class manages the details of each book in the library system, including the title, author, available copies, and total copies. It allows updating the number of available copies and retrieves the book's details in a formatted manner. The updatecount() method ensures the correct handling of the book's available and total copies, while the writedetails() method provides a readable summary of the book.