239758130\_class\_diagram

7.9: Assessment 2 - summative, system development using Python

[**Introduction to System Design 1**](#_3bmdqxjwt0o)

[Overview of the Implementation Approach 1](#_pgpzbutcr1dm)

[Addressing the Requirements 2](#_n35oi06r2dpa)

[Diagram Explanation 2](#_42j04yctwdb)

[**Class Diagram (description) 3**](#_i63bpzk5lizp)

[**Class Diagram (image) 5**](#_emc4k94gmnyu)

# Introduction to System Design

This document presents the system design for a **fictitious library management system**, implemented using the principles of **Object-Oriented Programming (OOP)** in Python. The goal of this system is to model real-world library operations, including managing books, users, and the loan process, in a modular and scalable manner that reflects good software engineering practices.

The main requirements for the system are as follows:

1. To create a fully functional **library record system** using Python classes to represent different entities (Books, Users, Loans, etc.).
2. The system must implement core functionalities such as **adding books and users**, **borrowing and returning books**, and **tracking overdue loans**.
3. The project must demonstrate the principles of **modularity**, **reusability**, and **encapsulation** by appropriately structuring the classes.
4. It must include **error handling** to ensure robustness and manage invalid inputs effectively.

## 

## Overview of the Implementation Approach

The library management system is designed with **five key classes**: Book, BookList, User, UserList, and Loan. Each class focuses on a specific domain of the library system, with its responsibilities clearly defined:

* **Book**: Represents individual book records, including attributes like title, author, and the number of available copies. The class includes methods to set and retrieve these attributes.
* **BookList**: Manages a collection of Book objects. This class is responsible for storing, adding, removing, and searching books.
* **User**: Represents library users and their personal details, with appropriate methods for setting and retrieving user information.
* **UserList**: Manages a collection of User objects, handling operations such as adding users, removing users, and counting users.
* **Loan**: Manages the borrowing process by associating users with books, tracking due dates, and providing methods to return books and manage overdue items.

## Addressing the Requirements

1. **Consistency with OOP Principles**:
   * The design adheres to OOP principles, using **classes** and **objects** to model real-world entities.
   * Each class encapsulates its data and provides appropriate methods to interact with its attributes.
2. **Modular Approach**:
   * By breaking the system into Book, BookList, User, UserList, and Loan, the implementation ensures a **modular design** that is easy to understand, extend, and maintain.
3. **Error Handling**:
   * The system includes **robust error handling** mechanisms, such as ensuring that book availability is checked before borrowing, validating user input formats, and managing unexpected conditions (e.g., attempts to remove non-existent users).
   * **Exception handling** is implemented throughout the classes to prevent crashes and handle invalid input gracefully.
4. **Relationships Between Classes**:
   * The relationships between the classes are clearly defined:
     + BookList and UserList both **aggregate** collections of Book and User objects, respectively.
     + The Loan class **associates** a User with a Book, representing the borrowing process.

Below, I have detailed each class along with their attributes and methods. I have also included notes that reflect my thought process and the purpose of each method.

### 

### Diagram Explanation

* **Book:** Represents individual book records with methods to set and retrieve attributes.
* **BookList:** Manages a collection of Book objects using a dictionary.
* **User:** Represents individual users with methods to set and retrieve personal details.
* **UserList:** Manages a collection of User objects using a dictionary.
* **Loan:** Manages the borrowing process, associating users with books and tracking loan details.

# 

# 

# Class Diagram (description)

**--------------------------------------------**

**Relationships:**

**--------------------------------------------**

1. BookList contains Books (Aggregation)

2. UserList contains Users (Aggregation)

3. Loan associates Users and Books (Association)

**--------------------------------------------**

**Class: Loan**

**--------------------------------------------**

- user: User

- book: Book

- due\_date: Date

+ \_\_init\_\_(self, user, book, due\_date) *#note to self: Handles invalid due dates and ensures both user and book exist in the system.*

+ borrow\_book(self, user, book, duration) *#note to self: Checks book availability, and handles cases where the book is not available or the user is invalid.*

+ return\_book(self, user, book) *#note to self: Validates if the book was borrowed by the user before allowing it to be returned.*

+ count\_books\_borrowed(self, user) *#note to self: Handles the case where the user has not borrowed any books.*

+ list\_overdue\_books(self) *#note to self: Handles exceptions if no books are overdue.*

**--------------------------------------------**

**Class: BookList**

**--------------------------------------------**

- books: Dictionary (key: book\_id, value: Book)

+ \_\_init\_\_(self) *#note to self: Initialises an empty collection and ensures correct setup.*

+ add\_book(self, book) *#note to self: IChecks for duplicate book IDs and handles invalid book data.*

+ search\_book(self, title=None, author=None, publisher=None, publication\_date=None) *#note to self: Validates search parameters and handles cases where no books match the criteria.*

+ remove\_book(self, title) *#note to self: Handles cases where the book does not exist in the collection.*

+ get\_total\_books(self) *#note to self: Handles calculation when there are no books.*

**--------------------------------------------**

**Class: Book**

**--------------------------------------------**

- book\_id: String

- title: String

- author: String

- year: Integer

- publisher: String

- num\_copies: Integer

- publication\_date: Date

+ \_\_init\_\_(self, title, author, year, publisher, num\_copies, publication\_date) *#note to self: Generates a random book ID and handles invalid input.*

+ set\_title(self, title) *#note to self: Each set\_ handles incorrect data types and prevents invalid data from being assigned.*

+ set\_author(self, author)

+ set\_year(self, year)

+ set\_publisher(self, publisher)

+ set\_num\_copies(self, num\_copies)

+ set\_publication\_date(self, publication\_date)

+ get\_title(self) *#note to self: Each get\_ is a simple method to retrieve attributes.*

+ get\_author(self)

+ get\_year(self)

+ get\_publisher(self)

+ get\_num\_copies(self)

+ get\_available\_copies(self)

+ get\_publication\_date(self)

**--------------------------------------------**

**Class: UserList**

**--------------------------------------------**

- users: Dictionary (key: username, value: User)

+ \_\_init\_\_(self) *#note to self: Initialises an empty user list.*

+ add\_user(self, user) *#note to self: Checks for duplicate usernames and handles invalid user information.*

+ remove\_user(self, first\_name) *#note to self: Handles ambiguity if multiple users have the same first name and ensures user existence.*

+ count\_users(self) *#note to self: Returns 0 if there are no users and ensures correct calculation.*

+ get\_user\_by\_username(self, username) *#note to self: Handles cases where the username does not exist.*

**--------------------------------------------**

**Class: User**

**--------------------------------------------**

- username: String

- first\_name: String

- surname: String

- house\_number: String

- street\_name: String

- postcode: String

- email: String

- date\_of\_birth: Date

+ \_\_init\_\_(self, username, first\_name, surname, house\_number, street\_name, postcode, email, date\_of\_birth) *#note to self: Initialises the user with error handling for empty fields, invalid email formats, and properly formatted dates.*

+ get\_username(self) *#note to self: Each get\_ is a simple method to retrieve attributes.*

+ get\_first\_name(self)

+ get\_surname(self)

+ get\_house\_number(self)

+ get\_street\_name(self)

+ get\_postcode(self)

+ get\_email(self)

+ get\_date\_of\_birth(self)

+ set\_first\_name(self, first\_name) *#note to self: Each set\_ handles incorrect data types and prevents invalid data from being assigned.*

+ set\_surname(self, surname)

+ set\_email(self, email)

+ set\_date\_of\_birth(self)

# Class Diagram (image)

BookList

UserList

Book

User

Loan