Library Management System Requirements Document

Overview

The Library Management System is a console-based C# application designed to allow users to manage a collection of books, including adding, removing, checking out, and returning books. This project aims to reinforce fundamental OOP principles, such as encapsulation, abstraction, inheritance, and polymorphism, while providing practical experience in problem-solving and applying C# syntax.

Functional Requirements

1. **Book Class**
    a. Define a Book class to represent a book with the following properties:
        i. `Title` (string): Title of the book.
        ii. `Author` (string): Name of the author.
        iii. `ISBN` (string): Unique identifier for the book.
        iv. `IsCheckedOut` (bool): Status of the book (checked out or available).
    b. **Methods**:
        i. `CheckOut()`: Marks the book as checked out.
        ii. `Return()`: Marks the book as returned.
        iii. Override `ToString()` to display book details, including its title, author, ISBN, and availability status.
2. **Library Class**
    a. Define a `Library` class to manage the collection of books.
    b. **Properties**:
        i. Use a private list to store books, ensuring encapsulation.
    c. **Methods**:
        i. `AddBook(Book book)`: Adds a new book to the collection.
        ii. `RemoveBook(string ISBN)`: Removes a book from the collection by ISBN.

iii. `CheckOutBook(string ISBN)`: Checks out a book by ISBN if it's available.
iv. `ReturnBook(string ISBN)`: Returns a book by ISBN if it's checked out.
v. `DisplayBooks()`: Lists all books in the collection, showing each book's details using the `ToString()` method.

3. **User Interaction**
    a. Create a menu in the `Main` method for the following options:
        i. Add a book.
        ii. Remove a book.
        iii. View all books.
        iv. Check out a book.
        v. Return a book.
        vi. Exit the system.
    b. Ensure input validation and user-friendly messaging.

Non-Functional Requirements

1. **Code Quality**
    a. Follow clean code principles: use descriptive naming, consistent formatting, and comments where necessary.
    b. Avoid repetitive code; optimize methods and refactor where possible.
2. **Modularity and Encapsulation**
    a. All class fields should be private, only accessible through public properties or methods.
    b. Ensure that any change to the state of a `Book` (such as checking out or returning) is handled internally through the `Library` class methods.
3. **Error Handling**
    a. Handle potential errors gracefully, such as:
        i. Attempting to remove or check out a book that doesn't exist.
        ii. Trying to check out a book that is already checked out.
    b. Provide clear error messages to guide users.

Guidance Notes

- **Use of Constructors**: Initialize objects using constructors to practice constructor syntax and object instantiation.
- **Focus on OOP Principles**: Ensure each class has a single responsibility (SRP). Book should only represent a book, while `Library` manages the book collection.
- **Code Comments and Readability**: Comment each method and important sections of code to demonstrate understanding of each part.
- **Testing Edge Cases**: Test with scenarios such as trying to check out a book twice, removing a book not in the collection, and adding duplicate books.
- **Future Extensibility**: Think about how you could extend this application, e.g., by adding a search function or accommodating different item types.