

**ENI08107-OBJECT ORIENTED SOFTWARE DEVELOPMENT**

**ASSIGNMENT**

**Assessment Title: LIBRARY MANAGEMENT SYSTEM**

**Introduction**

This paper explains the programming task you are doing in C#. You must do this task individually. This document will describe your tasks, the task's intended learning objectives, the evaluation standards, and other important information and materials.

Environment: To complete the Task, use the following:

1. C# Compiler
2. IDE: Visual Studio
3. Windows Forms

**Assessment Summary**

**Outcomes**

1.Implement and Debug C# applications using an Integrated Development Environment

2.Design ,develop and test  Object Oriented C# applications

3.Develop C# applications with graphical user interfaces ,file handling and data structures

4.Design Testcases  generate specification and reports

**Task Description**

**Library Management System Overview**

Develop a C# program that simulates a Library Management System with a graphical user interface (GUI) using Windows Forms. The system should allow users to add, search for, borrow, and return books through the GUI. Additionally, the program should manage user information and handle file I/O operations for data persistence.

**Basic Requirements**

You will need to include the following in your code for the completion of Task:

1. **Book Class**:
   * Create a class named **Book** with attributes for title, author, ISBN, and availability status.
   * Include appropriate constructors and methods to set and retrieve these attributes.
2. **Library Class**:
   * Implement a class named **Library** to manage a collection of books and library users.
   * Use suitable data structures to store the books and users.
   * Include methods to add a new book, remove a book, search for a book by title, and display the list of available books.
   * Include methods to add a user, remove a user, and display all users.
3. **User Class**:
   * Implement a class named **User** with attributes for user ID, name, and a list of borrowed books.
   * Include methods to borrow and return books.
4. **Borrowing and Returning:**
   * Implement methods in the **Library** class to handle borrowing and returning books.
   * Ensure proper error handling for scenarios like attempting to borrow an unavailable book or returning a book that was not borrowed.
5. **File I/O:**
   * Implement methods in the **Library** class to load initial book inventory and user information from files.
6. **Error Handling:**
   * Save the current state of the library (book inventory and user information) to files.
   * Implement appropriate error handling mechanisms(e.g. try catch finally , Format Exception etc.) throughout your program.
   * Clearly define and communicate error messages to users in case of invalid inputs, file read/write errors, or any other exceptional conditions.

**Extended Requirements**

**Generic Library Management System**

* Edit the **Library** class so it can manage different types of items beyond books. The library should be able to store and handle items of various types (e.g., books, DVDs, CDs).

**Item Hierarchy:**

* Create a generic **Item** class that serves as the base class for different item types.
* Implement specific derived classes for different item types (e.g., **Book**, **DVD**, **CD**). Each derived class should extend the **Item** class.

**Flexible User Class:**

* + Modify the **User** class to be generic so that it can handle different types of items when users borrow them from the library.

**Templates:**

* Utilize templates and template specialization where appropriate.

**Sample Data Files:**

To facilitate testing and evaluation, you will be provided with sample data files containing initial user and inventory information. These files will adhere to a specific format to ensure uniformity during testing. Please follow the guidelines below:

* **User Information File:**
* The user information file (**users.txt**) will have the following format:

*UserID1 UserName1 BorrowedBook1 BorrowedBook2 ... UserID2 UserName2 BorrowedBook3 BorrowedBook4 ...*

* Each line represents a user's information.
* The first entry is the user ID.
* The second entry is the user's name.
* Subsequent entries are the ISBNs of books that the user has borrowed.
* **Inventory Information File:**
* The inventory information file (**inventory.txt**) will have the following format:

*ISBN1 Title1 Author1 Availability1 ISBN2 Title2 Author2 Availability2*

* Each line represents a book's information.
* The first entry is the ISBN of the book.
* The second entry is the title of the book.
* The third entry is the author of the book.
* The fourth entry is the availability status (1 for available, 0 for checked out).

**Testing:**

**Important Information**:

* + Use the provided sample data files for initial testing.

You can create your own data files for further testing.

* + Ensure your program can correctly load and process information from these files.
  + Thoroughly test your implementation with various scenarios to ensure correct functionality.
  + Include a separate section in your report detailing the testing process and outcomes.
  + Test cases should cover:
    - Valid inputs for adding books, borrowing, and returning books.
    - Invalid inputs to assess error handling (e.g., attempting to borrow a nonexistent book).
    - Proper functioning of file I/O operations.
  + You should include your testing in your submission, even if commented out.

**Specific Testing:**

Outside of the ability for dynamically using the system functions**,** you are to include specific scenario testing too **(see scenarios below)**.

* This should be hardcoded and can be run whenever the system is compiled/executed by including it in the main function or in a specific testing section of code.
* You should make sure that you include appropriate outputs for each scenario showing whether they have worked or not.
* You should only use the given sample data files for this testing.

**Test Scenario:**

1. Load the sample data files into the system.
2. Add the book “Good Omens” by Neil Gaiman and Terry Pratchett with ISBN 978-0060853982 to the library.
3. Create a user profile for a user called “Emily”.
4. Have Emily borrow “1984”.
5. Have Emily return “1984”.
6. Search for “Pride and Prejudice” in the library.
7. Have David borrow “The Catcher in the Rye”.
8. Export the current data.

**Report:**

You should include a 1–2-page report that covers the following:

* + **Program structure.**
    - Describe the classes and functions included and any relationships between these and how they interact.
  + **Functionality**
    - You should include a list of expected functionalities and describe whether.

your system includes these and explain how it works in your system.

* + **Design Decisions**
    - You should explain the reasoning behind your design decisions, e.g. why did you choose the data structure you did to store books, the structure etc.

**How to submit the Assignment:**

Submit a zip file including the following:

* + Well-documented C# code with comments explaining key parts of the implementation.
  + A report discussing design decisions, program structure and functionality, and details of the testing process.
  + Include instructions on how to compile and use the system.
  + Any additional files required for submission.

**Code Quality:**

* + Follow good coding practices, including meaningful variable and function names.
  + Use appropriate indentation and formatting.
  + Provide clear and concise comments explaining complex sections of the code.

**Marking Criteria:**

Your coursework will be graded based on the following criteria:

* Implementation
* Code clarity and Documentation.
* Error handling
* Testing
* Report Quality and Design Explanation
* Advanced Features

**Commenting your Code**

For identification purposes, please make sure that you place a comment at the top of your files before you submit it which includes your full name, matriculation number and module code. So, it may appear thus for example:

/\*

Name : StudentID : Module code:

\*/

You can place this right at the top of the file, before any assembly directive statements.

**Marking Scheme :All the above-mentioned classes and files should be implemented in the code:**

**TOTAL:100 Marks**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Code**  **Functionality**  **(30%)** |  |  | | --- | |  | | |  | | --- | | Code Correctness (20%) |  |  | | --- | |  | | |  | | --- | | 20% |  |  | | --- | |  | | |  | | --- | | The program runs without  errors and produces the  correct output for various  test cases. |  |  | | --- | |  | |
|  | |  | | --- | | Completeness (10%) |  |  | | --- | |  | | |  | | --- | | 10% |  |  | | --- | |  | | |  | | --- | | All required features and  functionalities are implemented. |  |  | | --- | |  | |
| |  | | --- | | **Code Quality**  **(30%)** |  |  | | --- | |  | | |  | | --- | | Readability (10%) |  |  | | --- | |  | | |  | | --- | | 10% |  |  | | --- | |  | | |  | | --- | | Code is well organized and easy to  read. | |  |  |  | | --- | |  | |
|  | |  | | --- | | Modularity (10%) |  |  | | --- | |  | | |  | | --- | | 10% |  |  | | --- | |  | | |  | | --- | | Code is broken down into reusable  methods or classes. |  |  | | --- | |  | |
|  | |  | | --- | | Documentation (10%) |  |  | | --- | |  | | |  | | --- | | 10% |  |  | | --- | |  | | |  | | --- | | Class, method, and inline  comments are present and  appropriate. |  |  | | --- | |  | |
| |  | | --- | | **Testing and Error**  **Handling (20%)** |  |  | | --- | |  | | |  | | --- | | Robust error handling (10%) |  |  | | --- | |  | | |  | | --- | | 10% |  |  | | --- | |  | | |  | | --- | | Robust error handling is  implemented throughout the  solution. |  |  | | --- | |  | |
|  | |  | | --- | | Thorough testing of functionality,  including edge cases (10%) |  |  | | --- | |  | | |  | | --- | | 10% |  |  | | --- | |  | | |  | | --- | | Thorough testing of functionality  is demonstrated, including.  edge cases. |  |  | | --- | |  | |
| |  | | --- | | **Advanced Features**  **(10%)** |  |  | | --- | |  | | |  | | --- | | Implementation of additional  features (5%) |  |  | | --- | |  | | |  | | --- | | 5% |  |  | | --- | |  | | |  | | --- | | Implementation of additional  features beyond the basic  requirements. |  |  | | --- | |  | |
|  | |  | | --- | | Complexity of additional features  (5%) |  |  | | --- | |  | | |  | | --- | | 5% |  |  | | --- | |  | | |  | | --- | | The complexity of the additional  features implemented. |  |  | | --- | |  | |
| |  | | --- | | **Report (10%)** |  |  | | --- | |  | | |  | | --- | | Clear structure and coverage of  all required points |  |  | | --- | |  | | |  | | --- | | 5% |  |  | | --- | |  | | |  | | --- | | The report has a clear structure,  and covers all required points. |  |  | | --- | |  | |
|  | |  | | --- | | Clear and reasonable explanation  of design decisions |  |  | | --- | |  | | |  | | --- | | 5% |  |  | | --- | |  | | Clear and reasonable explanation of design decisions are given. |

**Note:**

In C#, unused variable warnings are typically generated by the compiler if you have a variable that is declared but not used within the scope it is declared in. If you want to explicitly enable or enforce these warnings ,you can do so through your project’s settings in Visual Studio or via your .csproj file.

Enabling Unused Variable Warnings in Visual Studio:

1. Open your project in Visual Studio
2. Go to Project Properties

* Right-click on your project in the Solution Explorer.
* Select “Properties” from the context menu.

1. Navigate to Build settings:

* In the Project Properties Window, select the “Build “ tab.

1. Configure Warning Level:

* Set the “Warning level” to “4” to enable the highest level of warnings.
* Ensure that “Treat warnings as errors” is unchecked unless you want the build to fail on warnings.