EASTERN INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

Practice Assignment – Quarter 3, 2024-2025
Course Name: .Net Programming
Course Code: CSE 443
Student's Full Name:
Student ID:

Practice Assignment 2

To help you understand and apply OOP principles in C#, we will design a library management project with 10 interconnected exercises. This project will include classes, interfaces, inheritance, polymorphism, and other OOP concepts to simulate a system for managing books, members, borrowing and returning books, and related functionalities.

LIBRARY MANAGEMENT SYSTEM

Description**:**
The library management system will include the following functions:

1. Manage book information.
2. Manage library member information.
3. Manage book borrowing and returning.
4. Statistics and reporting.

**Create a console application project to work through exercise 1 to 10*

Exercise 1: Encapsulation - Book Class

Requirements:

1. Design a `Book` class with the following properties:
    a. `string ISBN`
    b. `string Title`
    c. `string Author`
    d. `int Year`
    e. `int CopiesAvailable`
2. Encapsulate these properties using `properties` with `accessors`. Make sure that:
    a. Year cannot be less than 0.
    b. `CopiesAvailable` cannot be less than 0.
    c. Add a `DisplayInfo()` method to display detailed information about the book.

Hint: *Use* `properties` *with* `get` *and* `set` *to control access and validate data.*

Exercise 2: Inheritance - `Member` and `PremiumMember` Classes

Requirements:
- Create a base class `Member` with the following properties:

- `string MemberID`

- `string Name`
- `string Email`

- Create a `PremiumMember` class from Member with the following properties:
  - `DateTime MembershipExpiry`
  - `int MaxBooksAllowed`

- Add a `DisplayInfo()` method in both classes to display the corresponding information.

Hint: *Use inheritance to reuse code and extend functionality..*

Exercise 3: Abstraction - Transaction class and subclasses

Requirements:

- Create an abstract class `Transaction` with the following properties:
  - `string TransactionID`
  - `DateTime TransactionDate`
  - `Member Member`
- Add an `abstract` method:
  - `void Execute()`
- Create classes that inherit from `Transaction`:
  - `BorrowTransaction` with the `Book BookBorrowed` property.
  - `ReturnTransaction` with the `Book BookReturned` property.

Implement the `Execute()` method for each subclass to handle borrowing and returning books.

Hint: Use `abstract classes` to define methods that subclasses must implement.

Exercise 4: Polymorphism - Handling Transactions

Requirements:

- Use the classes from Question 3 (`Transaction, BorrowTransaction, ReturnTransaction`).
- Create a list of `Transaction` objects including `BorrowTransaction` and `ReturnTransaction`.
- Iterate through the list and call the `Execute()` method for each transaction, displaying the result.

Hint: Use `polymorphism` to call methods implemented in the child class through the base class.

Exercise 5: Interfaces - `IPrintable` and `IMemberActions`

Requirements:

1. Create the `IPrintable` interface with the method:
   - `void PrintDetails()`
2. Implement the `IPrintable` interface in the `Book` and `Member` classes. The `PrintDetails()` method will display respective information.
3. Create the `IMemberActions` interface with methods:
   - `void BorrowBook(Book book)`

    - `void ReturnBook(Book book)`
4. Implement the `IMemberActions` interface in the `Member` and `PremiumMember` classes.

Hint: Use interfaces to define behaviors that classes can implement.

Exercise 6: Constructors - `Library` Class

Requirements:

1. Design the `Library` class with the following properties:
    - `string LibraryName`
    - `List<Book> Books`
    - `List<Member> Members`
2. Create the following constructors:
    - Parameterless Constructor: Assign a default library name and initialize empty lists for books and members.
    - Parameterized Constructor: Accept the library name and an initial list of books.
    - Copy Constructor: Create a new `Library` object based on an existing one.
3. Add a method `DisplayLibraryInfo()` to display library information, including the number of books and members.

Hint: Use constructor overloading to create different ways to initialize objects.

Exercise 7: Overloading and Overriding - `NotificationService` Class

Requirements:

1. Create the `NotificationService` class with overloaded methods:
    - `void SendNotification(string message)`
    - `void SendNotification(string message, string recipient)`
    - `void SendNotification(string message, List<string> recipients)`
2. Create the `AdvancedNotificationService` class that inherits from `NotificationService` and overrides the method `SendNotification(string message)` to add timestamp information.
3. Create objects from both classes and call the `SendNotification` methods to observe the differences. (Create in the main method)

Hint:

- Overloading allows multiple methods with the same name but different parameters.
- Overriding allows a derived class to provide a specific implementation of a method defined in the base class.

Exercise 8: Properties with Access Modifiers - `LibraryCard` Class

Requirements:

1. Design the `LibraryCard` class with the following properties:
    - `string CardNumber` (read-only)
    - `Member Owner` (read and write)
    - `DateTime IssueDate` (read-only externally)

2. Add a method `RenewCard()` to renew the card, ensuring that `IssueDate` is updated.
3. Use appropriate access modifiers to protect the data.

Hint: Use private setters or init-only properties to control access.

Exercise 9: Difference Between Class and Record - `BookClass` vs `BookRecord`

Requirements:

1. Create a `class` named `BookClass` with the following properties:
    - `string ISBN`
    - `string Title`
    - `string Author`
2. Create a `record` named `BookRecord` with similar properties.
3. Compare the behavior of the two objects when performing:
    - Comparison using the == operator.
    - Using `with` to create a copy with modified properties.
4. Explain the differences between `class` and `record` based on the experimental results.

Hint:

- `class` is a reference type and compares by reference.
- `record` is a reference type but compares by value and supports with-expressions.

Exercise 10: Delegates and Events in OOP - `Library` and `NotificationService` Classes

Requirements:

1. Create the `Library` class with an event `OnBookBorrowed` using the delegate `Action<Book, Member>`.
2. Create the `NotificationService` class to subscribe to the `OnBookBorrowed` event and send notifications when a book is borrowed.
3. Implement registering multiple methods to the event and ensure all are called when the event is triggered.

Hint:

- Use delegates to define method signatures for events.
- Use events to allow other classes to subscribe to and handle them.