[Video 1] Project Overview

Build a library system where:

1. The library can have Books, magazines and Journals.
2. Some types of items can be loaned and some are only for reading at library.
3. Users (Members : Students, Professors | Librarian) can borrow and return different types of Library items.
4. Books have categories (Fiction, Non-fiction, Academic).
5. A loan management system keeps track of issued books.
6. Implement features like late fees and borrowing limits.

Milestone A : Represent User structure

- **Create the Base `User` Class**
- **Add Constructors to `User`**
- **Create a Subclass `Member`**
- **Create a Subclass `Librarian`**
- **Implement `generateUniqueId using` Static and Final Concepts**

Task 1: Create the Base `User` Class

Objective: Introduce abstract classes, encapsulation, and basic object-oriented principles.

1. **Step 1.1:** Define a class `User` with the following private attributes:
   - `String userId`
   - `String name`
   - `String contactInfo`
2. **Step 1.2:** Add getter and setter methods for `name` and `contactInfo`.
   - **Challenge:** Use encapsulation by keeping attributes private and accessing them through getters/setters.

Task 2: Add Constructors to `User`

Objective: Explore constructors (default, parameterized, and copy).

1. **Step 2.1:** Implement:
    - A **default constructor** that initializes `userId` using `generateUniqueId method` (We can return 0 from this method for now)
    - A **parameterized constructor** that initializes `name` and `contactInfo`.
    - A **copy constructor** that copies attributes from another `User`.
2. **Step 2.2:** Test constructors by creating instances using all three constructors in a test class.

Task 3: Make `User` an Abstract Class

Objective: Understand the concept of abstract classes and polymorphism.

1. **Step 3.1:** Mark `User` as `abstract` and declare the following abstract methods:
    - `void displayDashboard()`
    - `boolean canBorrowBooks()`
2. **Step 3.2:** Explain why these methods are abstract and how they enable polymorphism.

Task 4: Create a Subclass `Member`

Objective: Implement inheritance and method overriding.

1. **Step 4.1:** Create a concrete subclass `Member` that extends `User`.
2. **Step 4.2:** Add the following private attributes:
    - `int borrowedBooksCount`
    - A constant `MAX_BORROW_LIMIT = 5`
3. **Step 4.3:** Override the abstract methods:
    - `displayDashboard()` should display `Member Dashboard` and `Books Borrowed: X`.
    - `canBorrowBooks()` should return `true` if `borrowedBooksCount < MAX_BORROW_LIMIT`.

4. **Step 4.4:** Add constructors to initialize `Member`.

Task 5: Create a Subclass `Librarian`

Objective: Implement additional subclass-specific functionality.

1. **Step 5.1:** Create a subclass `Librarian` that extends `User`.
2. **Step 5.2:** Add the private attribute `String employeeNumber`.
3. **Step 5.3:** Override the abstract methods:
   - `displayDashboard()` should display `Librarian Dashboard` and the `employeeNumber`.
   - `canBorrowBooks()` should always return `true`.
4. **Step 5.4:** Add methods for librarian-specific actions:
   - `void addNewBook(Book book)`
   - `void removeBook(Book book)`
   - Leave implementations as comments for now.

Task 6: Demonstrate Static and Final Concepts

Objective: Understand `static` and `final` concepts with practical use.

Resource for Static : https://www.scaler.com/topics/static-keyword-in-java/
Resource for Final : https://www.scaler.com/topics/java/final-keyword-in-java/

1. **Step 6.1:** Add a static counter `totalUsers` and a method `getTotalUsers()` to track the total number of users.
   - **Challenge:** Use a static variable to maintain state across instances.
2. **Step 6.2:** Write a `generateUniqueId()` method to create unique user IDs. Mark this method as `final` to prevent overriding.
3. **Step 6.3:** Verify that:
   - The `generateUniqueId` method cannot be overridden in subclasses.
   - The `totalUsers` counter accurately tracks the number of users.

MileStone B

Task 1: Create the `Lendable` Interface

Objective: Introduce interfaces and compile-time polymorphism.

1. **Step 1.1:** Define the `Lendable` interface with the following methods:
   - `boolean lend(User user)`
   - `void returnBook(User user)`
   - `boolean isAvailable()`
2. **Step 1.2:** Explain the purpose of interfaces and how they enable **compile-time polymorphism**.
3. **Step 1.3:** Create a basic test class to simulate borrowing a book by defining a dummy class that implements `Lendable`.

Task 2: Implement the Abstract `Book` Class

Objective: Explore abstract classes, encapsulation, and method overriding.

1. **Step 2.1:** Create the `Book` class that implements `Lendable`. Add the following private attributes:
   - `String isbn`
   - `String title`
   - `String author`
   - `boolean isAvailable`
2. **Step 2.2:** Implement the methods from `Lendable`:
   - `lend(User user)`: If the book is available and the user can borrow, mark the book as unavailable and return `true`.
   - `returnBook(User user)`: Mark the book as available.
   - `isAvailable()`: Return the availability status.
3. **Step 2.3:** Explain why the class is abstract and add an abstract method `void displayBookDetails()`.

MileStone C

Task 1: Set Up Collections

Objective: Understand and implement collections to manage system-wide data.

1. **Step 1.1:** Create a class `LibraryManagementSystem` with:
   - A `List<Book>` named `bookInventory` to store all books.
   - A `List<User>` named `registeredUsers` to store all registered users.
2. **Step 1.2:** Explain the purpose of using collections
3. **Step 1.3:** Add methods:
   - `addBook(Book book)` to add a book to `bookInventory`.
   - `registerUser(User user)` to add a user to `registeredUsers`.
4. **Step 1.4:** Test the collections by adding a few books and users, then print their details.

Task 2: Implement Search Functionality

Objective: Demonstrate compile-time polymorphism through method overloading.

1. **Step 2.1:** Add a static method `searchBooks(String criteria)` to search for books by title or author. Use a loop to iterate over `bookInventory` and add matching books to a result list.
2. **Step 2.2:** Overload `searchBooks` with additional parameters:
   - `searchBooks(String criteria, String type)` for searching books of a specific type (`"TextBook"` or `"NovelBook"`).
   - Implement this method to filter results based on the type of book.
   - Java Enums
3. **Step 2.3:** Test the overloaded methods with different inputs and ensure they return correct results.

Task 3: Integrate Book and User Management

Objective: Combine book and user features to demonstrate system functionality.

1. **Step 3.1:** In the `main` method:
   - Create a few instances of `TextBook` and `NovelBook`.
   - Add these books to the library using `addBook`.
2. **Step 3.2:** Create instances of `Member` and `Librarian`.
   - Register them using `registerUser`.

3. **Step 3.3:** Print the details of all books and users to verify the inventory and registration system.

Task 4: Demonstrate Lending Functionality

Objective: Practice the interaction between users and books.

1. **Step 4.1:** Simulate lending a book:
   - Attempt to lend a `TextBook` to a `Member` using the `lend(User user)` method.
   - Print a success message if the lending operation is successful.
2. **Step 4.2:** Add logic to handle the following scenarios:
   - A user attempts to borrow a book that is already lent.
   - A user exceeds their borrowing limit.
3. **Step 4.3:** Test lending with different types of books and users.

Task 5: Manage Returns

Objective: Complete the book borrowing cycle.

1. **Step 5.1:** Simulate returning a book:
   - Use the `returnBook(User user)` method to mark a book as available again.
2. **Step 5.2:** Ensure the book can be lent to another user after it is returned.
3. **Step 5.3:** Test the return functionality by printing the availability status of books before and after returning.

Task 6: Advanced Features

Objective: Explore additional features to extend the system.

1. **Step 6.1:** Add a method `displayAllBooks` to print the details of all books in `bookInventory`.
2. **Step 6.2:** Add a method `displayRegisteredUsers` to print the details of all users in `registeredUsers`.
3. **Step 6.3:** Demonstrate searching:
   - Search for books by title or author using `searchBooks`.
   - Search for books by type using the overloaded method.

Task 3: Add Constructors to the `Book` Class

Objective: Practice constructor overloading and copying.

1. **Step 3.1:** Add the following constructors:
   - A **default constructor** that initializes `isAvailable` to `true`.
   - A **parameterized constructor** to initialize `isbn`, `title`, and `author`.
   - A **copy constructor** to create a new `Book` object from an existing one.
2. **Step 3.2:** Test the constructors by creating objects using each constructor.

Task 4: Create `TextBook` Class

Objective: Demonstrate inheritance and method implementation.

1. **Step 4.1:** Define the `TextBook` class as a subclass of `Book` with the following additional attributes:
   - `String subject`
   - `int edition`
2. **Step 4.2:** Add a parameterized constructor to initialize all attributes, including those inherited from `Book`.
3. **Step 4.3:** Override `displayBookDetails()` to display the textbook's details.
4. **Step 4.4:** Test the `TextBook` class by creating an object and calling its methods.

Task 5: Create `NovelBook` Class

Objective: Implement another concrete subclass to explore different book types.

1. **Step 5.1:** Define the `NovelBook` class as a subclass of `Book` with the additional attribute:
   - `String genre`
2. **Step 5.2:** Add a parameterized constructor to initialize all attributes, including those inherited from `Book`.
3. **Step 5.3:** Override `displayBookDetails()` to display the novel's details.
4. **Step 5.4:** Test the `NovelBook` class by creating an object and calling its methods.
