Professional Project Idea: "Library Management System (LMS)"

This system manages books, members (customers), and librarians (employees), with a single admin. It perfectly fits the three user roles and all functional requirements.

---

1. The 6 Core Classes:

1. **User (Abstract Base Class)**
    - **Attributes: user_id, username, password, is_active**
    - **Purpose: The base class for all users. Handles common login/logout functionality and the is_active flag.**
2. **Admin (Inherits from User)**
    - **Attributes: (Inherits from User)**
    - **Purpose: Can manage all users (add, remove, activate/deactivate, view counts).**
3. **Librarian (Inherits from User)**
    - **Attributes: employee_id (optional, can use user_id)**
    - **Purpose: The "Employee" role. Can add, delete, update, and search for books.**
4. **Member (Inherits from User)**
    - **Attributes: member_id, borrowed_books (a list)**
    - **Purpose: The "Customer" role. Can borrow and return books.**
5. **Book**
    - **Attributes: book_id, title, author, isbn, is_borrowed**
    - **Purpose: Represents a book in the library's collection.**
6. **LibraryManager (The "System" Class)**
    - **Attributes: users_list, books_list**

- **Purpose: This is the most important class. It orchestrates everything:**
  - **Manages the lists of all users and books.**
  - **Handles user registration (with unique username check).**
  - **Contains methods for all operations (borrow, return, add book, etc.).**
  - **Handles reading from and writing to files (users.txt, books.txt).**
  - **Contains input validation methods.**
  - **Uses try-except blocks for exception handling.**

---

2. How It Meets All Your Requirements with 6 Classes:

- **Users & Authentication (3 Types):**
  - **Admin, Librarian (Employee), Member (Customer). All inherit from User.**
  - **Unique Username: The LibraryManager.add_user() method checks the users_list before adding a new user.**
  - **Login/Logout: Handled by methods in the LibraryManager that search the users_list.**
  - **Activation/Deactivation: The User class has an is_active attribute. The LibraryManager checks this before allowing any operation.**
- **Permissions:**
  - **Admin: Can call methods in LibraryManager like view_all_users(), deactivate_user(), get_users_count().**
  - **Librarian (Employee): Can call methods like add_book(), delete_book(), search_book_by_id(title/id).**

- **Member (Customer): Can call borrow_book() and return_book(). The borrow_book() method is their "purchase" operation. It can also calculate fines, acting as the "invoice".**

- **OOP Principles:**
  - **Inheritance: Admin, Librarian, Member all inherit from User.**
  - **Encapsulation: Attributes are private (e.g., __password). Methods like get_username() are used.**
  - **Abstraction: The User class is abstract; you don't create a generic "User" object.**
  - **Polymorphism: While simple, you can have a display_role() method that behaves differently for each user type.**
- **Data Structures & Functions:**
  - **Lists: The LibraryManager holds two main lists: list[User] users_list and list[Book] books_list.**
  - **Functions: All actions are methods within the LibraryManager or the classes themselves (e.g., book.is_available()).**
- **Input Validation & Exception Handling:**
  - **Validation: Methods in LibraryManager will validate:**
    - **Phone Number (for member registration): Length 9, starts with 77/78/71.**
    - **Password: Length >7, contains letters, numbers, symbols.**
    - **Names: Not just numbers/symbols.**
  - **Exceptions: Try-Except blocks used in LibraryManager for file operations and when converting input (e.g., converting book_id string to integer).**
- **File Handling:**
  - **The LibraryManager will have load_data() and save_data() methods.**

- **These methods will read from/write to users.txt and books.txt to persist all data after the program closes.**