Final Project: OOP Python Application

Objective: Create a fully functional Object-Oriented Python application that incorporates abstraction, interfaces, exception handling, and custom class design.

Project Theme: Build a simplified "Library Management System."

Requirements:

- All `LibraryItem` and `User` instances should initially be read from respective data files (e.g., `items.json`, `users.json`).
- When the administrator adds a new item or user, the updated information should be saved back to these files upon exiting the system.

1. **Abstract Classes and Interfaces**:
    - Create an abstract class `LibraryItem` with attributes like `title`, `author`, and methods like `display_info()` and `check_availability()`.
    - Subclasses: `Book`, `Magazine`, `DVD` each with specific attributes and behaviors.
    - Create an interface-like abstract class `Reservable` with method `reserve(user)` and implement it in `Book` and `DVD`.
2. **Class Composition and Relationships**:
    - Create a `User` class with attributes like `user_id`, `name`, `borrowed_items`.
    - Create a `Library` class to manage items and users. It should allow:
        - Adding/removing items
        - Adding/removing users
        - Borrowing and returning items
        - Reserving items using the `Reservable` interface
3. **Exception Handling**:
    - Handle the following scenarios with try-except:
        - Borrowing an unavailable item
        - Reserving an item that is already reserved
        - Invalid user inputs (nonexistent users or items)
        - File errors when saving/loading data (optional bonus)
    - Use `finally` to ensure consistent application flow.
4. **Custom Exceptions**:
    - Create and raise custom exceptions like `ItemNotAvailableError`, `UserNotFoundError`, `ItemNotFoundError`.
5. **File Structure**:
    - Organize your project using multiple files:

- `main.py`: contains the CLI interface or main loop
- `models/`: directory with `library_item.py`, `book.py`, `user.py`, etc.
- `exceptions/`: directory with custom exceptions

6. **System Interaction Scenario**:
    - Upon launching `main.py`, the system should:
        - Load existing items from `items.json` and users from `users.json`.
        - Present a CLI menu with options:
            1. View all available items
            2. Search item by title or type
            3. Register a new user
            4. Borrow an item
            5. Reserve an item
            6. Return an item
            7. Exit and Save
    - Full Interaction Example:

Upon launching `main.py`, the CLI displays the following menu:

```
Welcome to the Library System
1. View all available items
2. Search item by title or type
3. Register as a new user
4. Borrow an item
5. Reserve an item
6. Return an item
7. Exit and Save
```

Example session:

- **User selects option 1**: The system displays all `Book`, `Magazine`, and `DVD` items with their availability status.
- **User selects option 2**: They are prompted to input a search keyword. The system returns matching items.
- **User selects option 3**: They input name and email, and are assigned a unique `user_id`. The new user is added to `users.json`.
- **User selects option 4**: They provide their `user_id` and item ID. If the item is available, it is marked as borrowed and added to the user's `borrowed_items` list.
- **User selects option 5**: They provide their `user_id` and item ID. If the item supports reservation (i.e., implements `Reservable`) and is not already reserved, the reservation is recorded.
- **User selects option 6**: They input their `user_id` and item ID. If matched, the item is marked as available again.
- **User selects option 7**: The system saves all changes back to `items.json` and `users.json` using exception-safe file handling, and exits.

Throughout:

- All invalid inputs (e.g., non-existent `user_id`, unavailable items, wrong menu choices) raise appropriate custom or built-in exceptions.
- File errors during loading or saving are caught and reported.
- The use of abstraction, interfaces (`Reservable`), and structured exception handling is integral to system operation.

Submission Instructions:

- Submit your full project as a `.zip` file.
- The archive should include all Python files and a `README.md` explaining how to run your project and the design decisions you made.
- Make sure your code is well-documented and commented.