Lab #4

Implementing GRASP Principles in Java

Objective: In this lab, you will learn to apply **GRASP (General Responsibility Assignment Software Patterns)** principles to design and implement a small system using **Java**. You will focus on applying the following principles:

1. **Information Expert**
2. **Creator**
3. **Controller**
4. **Low Coupling**
5. **High Cohesion**

Work in your assigned groups

Problem Statement:

You are tasked with building a **Library Management System**. The system should allow you to manage users/members and to perform operations such as borrowing books, returning books, and managing the catalog of books. Your task is to design and implement the system following GRASP principles to ensure good object-oriented design.

The Java Code (Initial version)

A simplified version of the **Library Management System** that does not use design principles is contained in the archive `lab4.zip` published on the Moodle page. Everything is handled within a single class `LibraryApp`, including managing members and their borrowed books. This design is suboptimal on purpose.

Why It's Suboptimal:

- **Single Responsibility Violation**: The class `LibraryApp` is responsible for both the books and the members, violating the Single Responsibility Principle (SRP) (one of the SOLID principles).
- **Tight Coupling**: Handling everything in a single class leads to tight coupling of functionality, making the code less flexible and harder to maintain.
- **Scalability**: This implementation won't scale well as more features or complexity (e.g., member profiles, overdue books) are added.
- **No error handling** for invalid inputs or edge cases.

Instructions:

Working as a team, refactor the initial version of the Library Management System by designing and implementing the classes described below. For design, adhere to GRASP principles as outlined in the Tasks section. Continue using the code repository you created in the previous lab assignment. Alternatively, create another shared code repository in GitHub. Make it public. Decide on responsibilities related to code development and codebase management. Use the codebase branching. The branches will have to be merged with the trunk for the final submission, but you can keep them in the repo. Create pull requests and assign members of your team as reviewers.

Classes:

1. **Book**
    a. Attributes: title, author, isAvailable
    b. Responsibilities:
        i. Manage its own state (available or borrowed).
    c. GRASP Principle: **Information Expert**
2. **Member**
    a. Attributes: memberId, name, borrowedBooks (list of Book)
    b. Responsibilities:
        i. Can borrow and return books.
    c. GRASP Principle: **Information Expert** and **Low Coupling**
3. **Library**
    a. Attributes: catalog (list of Book), members (list of Member)
    b. Responsibilities:
        i. Manage the catalog of books and members.
        ii. Track which books are borrowed and available.
    c. GRASP Principle: **Creator**, **Controller**
4. **LibrarianController**
    a. Responsibilities:
        i. Handles user requests like borrowing and returning books.
        ii. Delegate tasks to the appropriate objects.
    b. GRASP Principle: **Controller**
5. **LibraryApp**
    a. The class with main() function to demonstrate the application functionality.

Tasks:

1. Design the application and draw the class diagram

- Draw a class diagram and work collectively to develop shared understanding of the application.
- Use any diagramming tool to draw the final version of your design as an UML class diagram. Possible options are Visal Paradigm, Draw.io. Maintain your class diagram in a GitHub repo.

2. Apply the Information Expert Principle

Implement the logic to borrow a book using the **Information Expert** principle:

- The Book class knows its availability, so it should be responsible for updating its status.
- The Member class knows the books it has borrowed, so it should manage the list of borrowed books.

3. Apply the Creator Principle

The Library class should be responsible for managing instances of Book and Member associated with the library. Use the **Creator** principle to assign this responsibility appropriately. Write methods like `addBook`() and `registerMember`() to demonstrate this. Use **Information Expert** principle and create methods like `findMemberByName`(`String name)` and `findBookByTitle(String title)` to demonstrate this.

4. Apply the Controller Principle

The `LibrarianController` class will be the controller responsible for handling requests (e.g., borrow and return operations). Implement this class so that it delegates its tasks to other objects.

5. Apply Low Coupling and High Cohesion

Ensure that your system minimizes dependencies between classes. For example, the Library class should not directly manipulate Member or Book internals, but instead interact via well-defined methods. This promotes **Low Coupling**. Each class should also focus on a single responsibility, promoting **High Cohesion**.

6. Create a Scenario to demonstrate functionality

Update `LibraryApp` class to demonstrate use cases: *Add/Remove a member*, *Add/Remove a Book* from the library, *Borrow a book*, *Return a book, Lookup a book, Lookup a member*. Note that the use case *Borrow a book* has an extension – the use case *Reject Request* triggered when the requested book is not available. Come up with other use cases (*Display book catalog* etc) and demonstrate them. Use the same members (Alice and Bob) and books (Dune, 1984 and Moby Dick) and emulate user interactions in the body of the `main()` function.

Deliverables:

- Create the use case diagram of the Library Management System. The diagram must include use cases necessary to perform sample scenarios realized in your LibraryApp.
- Create class diagram of your software design
- Implement the Library Management System in Java code following the GRASP principles.
- Submit the use case diagram, the class diagram, and the Java code of your application (the link to your repository. Make sure it’s public)
- Provide a brief report explaining how each GRASP principle was applied in your design.

Evaluation Criteria:

- Correct implementation of GRASP principles: Information Expert, Creator, Controller, Low Coupling, and High Cohesion.
- Correctly drawn diagrams that shows use cases, classes and their relationships (the class diagram)
- Clean, well-structured code that adheres to OOP best practices.
- Clear documentation explaining your design decisions.

Good luck!