Library System

Idea

The application idea is a library management system that allows users to build a library of books and magazines (although it can be extended to allow more types of items) and provides extended functionality through decorators.

The application addresses the need for a flexible and extensible library system. The use of design patterns like Singleton, Builder, and Decorator enhances the modularity and extensibility of the code, allowing for easy addition of new features without modifying existing classes. The decorator pattern specifically allows for dynamic extension of the library's functionality. Furthermore, the Iterator pattern enables users to iterate through various types of items that the library contains, such as books and magazines.

Previous discussion

The idea for the library project stemmed from a previous assignment where we explored various design patterns. Initially, we considered reusing the code and design patterns from that project. However, upon rereading the exercise, we realized that the context and requirements were different, making code reuse not a good idea.

Despite this setback, we believed in the concept of the library project and its potential. Thus, we decided to think again which design patterns would be most suitable for the new requirements. After consulting "The Gang of Four" file from Moodle, we identified Singleton, Builder, and Decorator as complementary design patterns that could effectively address our needs.

We then asked ChatGPT to generate code based on these instructions. However, we encountered an issue when ChatGPT started iterating over an array instead of using the Iterator pattern. Recognizing that this approach was not optimal for our codebase, we decided to rectify it. Therefore, we used ChatGPT as a tool for rapidly developing a proof of concept (PoC), enabling us to dedicate all our time to refining design patterns.

Features

1. Flexibility and extensibility: Design patterns (Singleton, Builder, and Decorator) contribute to code flexibility and extensibility, enabling the addition of features without modifying existing structures.
2. Readability and maintainability: Patterns improve code readability and maintainability. Each pattern serves a specific purpose, enhancing the overall clarity of the codebase and making it easier to modify.

3. Dynamic functionality enhancement: The Decorator pattern enables dynamic functionality enhancement. It allows for the seamless addition of new behaviors to existing classes, showcasing adaptability to changing requirements.

Complementary design patterns

- **Singleton**
  Decision: Used the Singleton pattern for the Library class to ensure there is only one instance of the library throughout the application.
  Rationale: This design decision helps maintain a single point of access to the library, preventing multiple instances and ensuring consistency in managing books.

- **Builder**
  Decision: Implemented the Builder pattern for constructing the library using the LibraryBuilder class.
  Rationale: The Builder pattern provides a fluent interface for building complex objects step by step. It enhances readability and allows for easy extension when constructing the library with multiple books.

- **Decorator**
  Decision: Applied the Decorator pattern to extend the functionality of the Library class.
  Rationale: Decorators allow dynamic augmentation of an object's behavior. In this case, it allows us to add new features to the library without modifying its core functionality. The Increase and DecreaseBooksCapacityDecorator demonstrate how to increase or decrease the capacity of items in the library.

- **Iterator**
  Decision: Introduce the Iterator pattern to facilitate traversal of the collection of items in the Library class.
  Rationale: The Iterator pattern provides a standardized way to traverse elements in a collection without exposing its underlying representation. By incorporating the Iterator pattern, the Library class can offer a consistent interface for iterating over its collection of books and magazines.

Diagram and in-depth explanation

- The Book class represents a book in the library with its title and author.
- The Magazine class represents a book in the library with its title and publisher.
- The Library class implements the singleton pattern, ensuring that only one instance of the library exists throughout the application. Additionally, it implements two iterators: one for traversing the items of the library while respecting the item capacity, and another for traversing each type of item in the library, hiding the implementation details from the user.

- The LibraryItem interface has two methods to get the title and the owner (author in the case of books and publisher in the case of magazines).
- The LibraryBuilder class provides a way to construct the library by adding books one by one and for setting the items capacity.
- The LibraryDecorator class is an abstract class that extends the functionality of the Library class.
- The IncreaseBooksCapacityDecorator and DecreaseBooksCapacityDecorator class are concrete decorators that add extended functionality to the library.
- In the App class, books are created, added to the library using the builder pattern, and displayed. Also, the extended functionality is demonstrated using the decorator pattern. Finally, iterators are used to print the elements of the library.

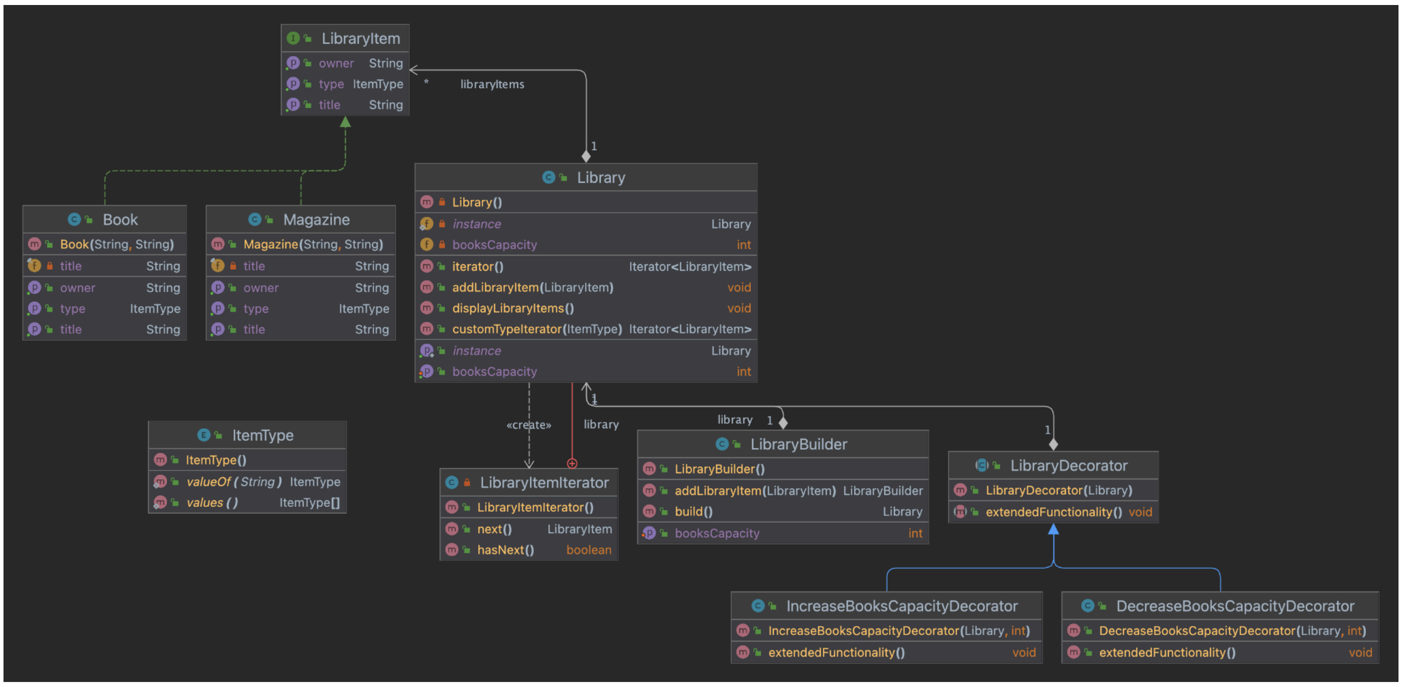

Fig. 1. Class diagram.

Github repository

https://github.com/slococo/adp-hw1