Software Systems and Design - Report

by Elizaveta Zagurskih

April 28, 2024

Design Patterns

Builder

The Builder design pattern was implemented in the assignment solution due to the need to simplify the process of initializing Book objects. While a book currently only requires three attributes (title, author, and price) it is expected that in the future it may be necessary to include additional attributes such as genre, description, or tags. Additionally, this template allows the book creation process to remain manageable and adaptable.

Façade

In the program, the Façade pattern is utilized to simplify the interaction with the **BookStore** system by providing a high-level interface that encapsulates the complexity of the underlying subsystems. The **Façade** class serves as the entry point for interacting with the **BookStore** system. It provides simplified methods for performing common operations such as creating books, creating users, subscribing to notifications, unsubscribing from notifications, updating book prices, reading books, and listening to audiobooks.

Additionally, the Façade pattern provides flexibility and extensibility, allowing changes to be made to the subsystem implementation without affecting client code. If the internal structure of a subsystem evolves or new features are added, the façade can adapt to these changes internally while maintaining a consistent interface for clients.

Strategy

In the program, the Strategy pattern is implicitly applied through the use of interfaces **ReadingBehavior** and **ListenBehavior**.

The **ReadingBehavior** interface represents the behavior of reading a book, while the **ListenBehavior** interface represents the behavior of listening to a book. Concrete implementations of these interfaces (**CanRead**, **CanListen**, **CanNotListen**) encapsulate specific behaviors.

The User class utilizes these behaviors through composition, where each User object has a reference to both a ReadingBehavior and a ListenBehavior. This design allows the behavior of a user to be changed dynamically at runtime, enabling flexibility and extensibility.

By employing the Strategy pattern, the code achieves separation of concerns, making it easier to add new behaviors and modify existing ones without altering the User class. Additionally, it promotes code reuse and maintainability by encapsulating behavior in separate classes.

Observer

The Observer design pattern was integrated into the program to establish a robust notification mechanism between books and subscribers. Currently, when the price of a book is updated, all subscribers are notified about the change. This ensures that subscribers stay informed about any modifications to the books they are interested in. Additionally, Books can be updated or modified independently of the subscribers, and new subscribers can be added or removed dynamically without affecting the book's functionality.

The **Observable** interface defines the contract for objects that can be observed. It includes methods such as **registerSubscriber**, **removeSubscriber**, and **notifySubscribers**. This interface allows the BookStore to manage its subscribers and notify them of changes. The **BookStore** class acts as the subject or observable entity. It maintains lists of books, users, and subscribers. It implements the **Observable** interface, which enables it to register, remove, and notify subscribers about changes in book prices. The **User** class represents observers in the system. Each user has behaviors for reading and listening to books. Additionally, the **update** method in the **User** class allows users to receive notifications about price updates for books they are interested in.

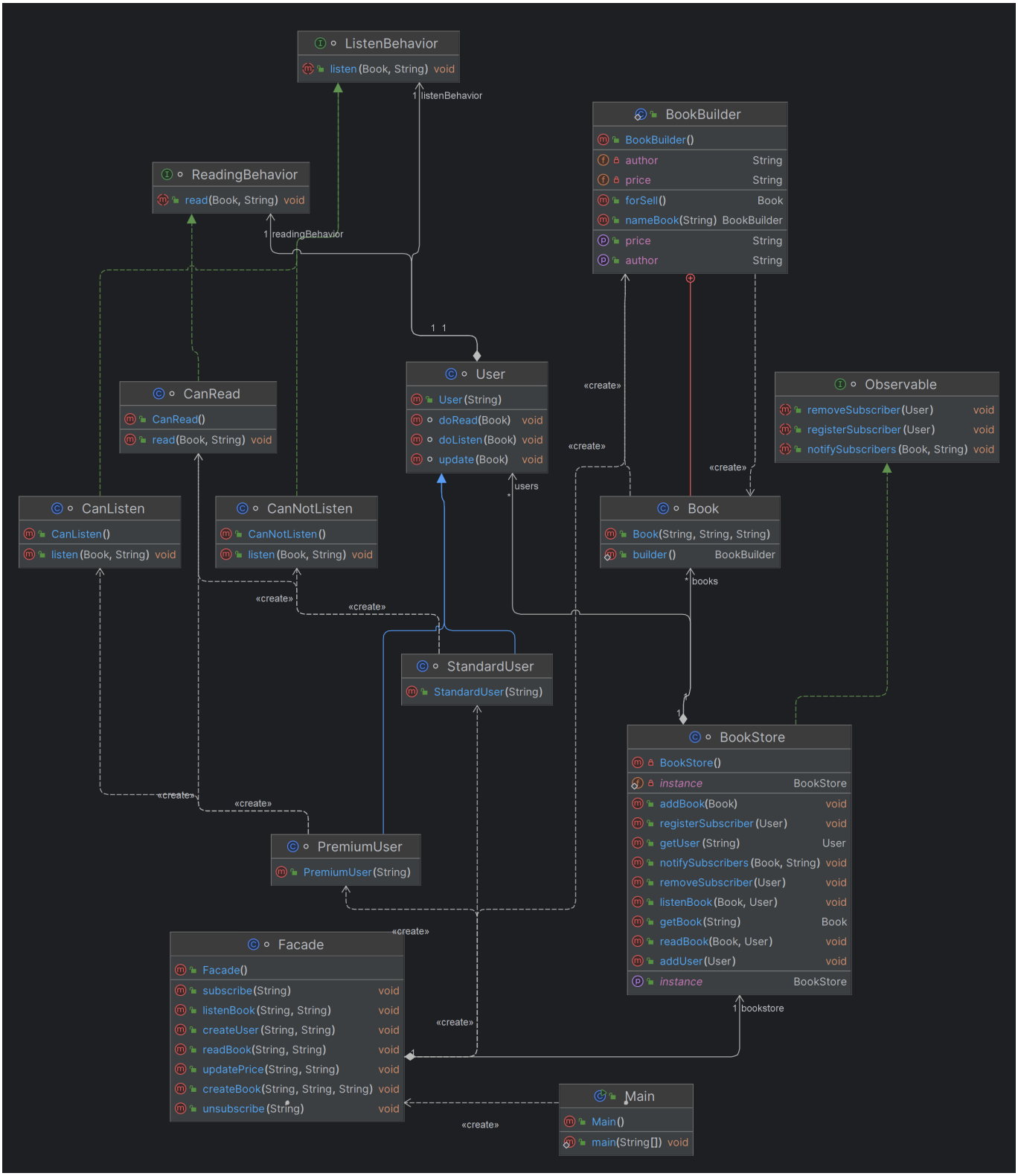

Figure 1: Assignment 4. UML diagram