**Week6 Reader’s Haven Online Bookstore Class Diagram Design Report**

Hang Yang

SYS 5013: System Engineering Analysis

Prof. Hitam Zidan

Sep 28, 2024

**Introduction**

In this week we will create a UML class diagram to illustrate the upgraded object-oriented design of Reader’s Haven online bookstore system. Readers' Haven is a comprehensive online platform that offers both physical and digital books for purchase and rental. As the platform has grown, its feature set has expanded to include customer reviews, personalized recommendations, user profiles, and transaction management. The system needs to manage these functionalities efficiently while providing a seamless user experience. This report documents the design of the UML class diagram created for the system, detailing the relationships between entities and the reasoning behind design choices.

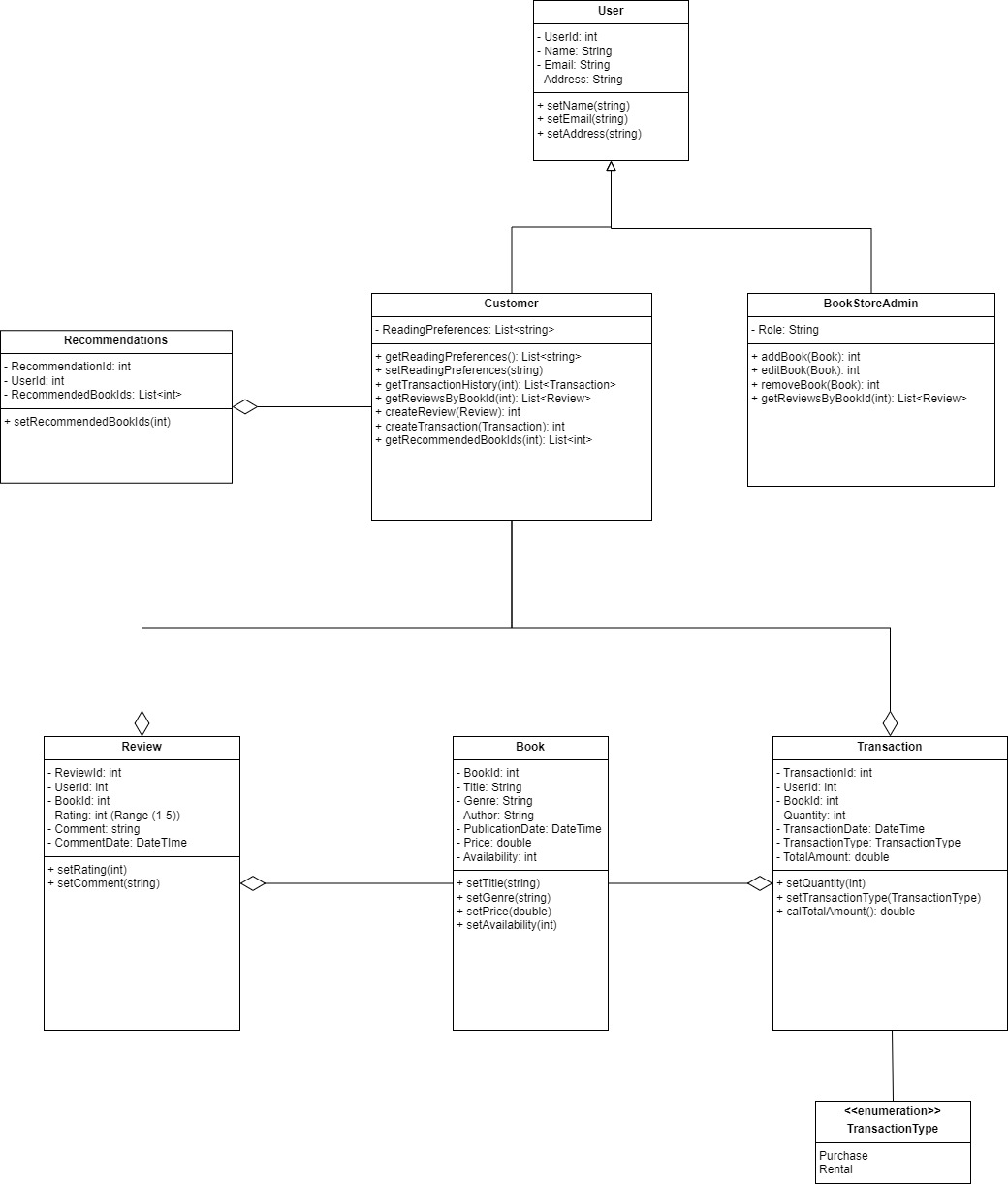
**Overview of the UML Class Diagram**

The UML diagram (refer to **Figure 1**) outlines the main entities involved in the system and their relationships, representing key aspects such as user management, book catalog management, transactions, and reviews. The core classes include:

* User (Base class for all users)
* Customer (Inherits from User)
* BookStoreAdmin (Inherits from User)
* Book (Represents the books in the system)
* Transaction (Tracks purchasing and rental transactions)
* Review (Handles customer feedback on books)
* Recommendation (Provides personalized reading suggestions)
* TransactionType (Enumeration for Purchase or Rental types)

**Figure 1**

*UML Class Diagram of Online Bookstore*



*Note*. picture by Hang Yang in 2024. Own work.

**Class Details and Design Decisions**

**User and Inheritance Structure**

Our design focus on the online bookstore users. Both customer and online bookstore admin would use this system, they will all inherit from **User** basic class. This User class serves as the base for both customers and administrators. Common attributes like *UserID, Name, Email, and Address* are defined here, which both types of users share. The class has setter methods to allow users to update their profiles. The **Customer** class Inherits from User, with specific attributes such as *ReadingPreferences* (a list of preferred genres) and methods for interacting with the system. Customers can create reviews, conduct transactions, and receive personalized recommendations. The inheritance model was chosen because both customers and admins share common properties (name, email, etc.), but each has unique functionalities. This avoids redundancy and enhances code reuse. For the **BookStoreAdmin** class, it inherits from User but includes administrative privileges such as adding, editing, and removing books from the system, as well as reviewing customer feedback. Since an admin has additional rights not available to a customer, encapsulating those features into a separate class makes the design cleaner and easier to maintain.

**Catalog Management**

The **Book** class encapsulates the attributes of each book in the system, including *BookID*, *Title, Author, Genre, Price, PublicationDate, and Availability*. Methods are provided to modify these details, allowing administrators to keep the catalog up to date. Each book has multiple attributes for identification, categorization, and price tracking. The Availability attribute tracks the number of copies available for rental or purchase. This separation ensures flexibility in managing the inventory.

**Transaction System**

The **Transaction** class records details of customer purchases or rentals, including *TransactionID, UserID, BookID, Quantity, TransactionDate, TransactionType, and TotalAmount*. Methods to calculate the total amount (calTotalAmount) and set the quantity and transaction type are included. Each transaction must track whether a book was rented or purchased, as well as the corresponding quantity and total cost. By keeping a record of the TransactionType (using the **TransactionType** enumeration), we can clearly differentiate between rentals and purchases. This **TransactionType** enumeration class defines two types of transactions: Purchase and Rental. Using an enumeration simplifies the process of identifying transaction types and reduces potential errors in transaction records.

**Customer Feedback**

Customers can provide reviews for books they have read or rented. This **Review** class captures the *ReviewID, UserID, BookID, Rating (on a scale of 1-5), and Comment*. Methods for setting the rating and comment are provided. By allowing users to leave reviews, we can enhance engagement and provide valuable feedback for future customers. By associating reviews with specific books and users, the system can provide more targeted recommendations and help admins monitor feedback.

**Recommender System**

This **Recommendation** class stores a list of recommended books for each customer. The *RecommendedBookIds* attribute holds book IDs for books suggested to the user based on their reading preferences and past transactions. By storing recommendations for each customer, the system can offer personalized book suggestions, improving the user experience and driving engagement.

**Relationships Between Classes**

The UML diagram illustrates several key relationships between the classes:

* **Inheritance (Generalization)**: The Customer and BookStoreAdmin classes inherit from the User class. This simplifies user management and ensures consistent behavior across different user types.
* **Association**: Several associations exist between classes, such as the link between the Customer and Transaction classes, where each transaction is associated with a customer and a book. Similarly, the Review class is linked to both the Customer and Book classes, indicating that reviews are tied to specific users and books.
* **Aggregation**: The Transaction class aggregates the TransactionType enumeration, meaning that each transaction has a type (either rental or purchase), but the transaction can exist independently of this enumeration.
* **Dependency**: The Customer class depends on the Recommendation class for personalized suggestions. While not tightly coupled, this relationship allows the system to dynamically generate recommendations based on user preferences and past activity.

I believe that the inheritance of User type can minimize redundancy between different types of users (customers and admins) while still allowing for role-specific behaviors. This design enhances code reusability and flexibility for future expansion, such as introducing new user types. The separation of books, reviews and transactions into distinct classes ensures that each entity has a clearly defined role within the system. This modular approach makes it easier to update individual aspects of the system without affecting others. By using aggregation and association appropriately, the design ensures that entities like books and transactions are not tightly coupled. This allows for a more flexible system that can scale as the business grows.

**Conclusion**

The UML diagram for "Readers' Haven" has been designed to reflect a scalable and maintainable architecture that can handle the platform's expanding functionality. The design choices, including inheritance, encapsulation, and modular class structures, provide a solid foundation for future development and feature enhancement. By separating concerns (user management, book management, transactions, reviews, etc.), the system remains adaptable and easy to extend.

**References**

Geekific. (2021, January 20). UML Class and Object Diagrams | Association vs. Aggregation vs. Composition | Geekific [Video]. Youtube. <https://www.youtube.com/watch?v=sN2_CoB_kbw>