MP4 - Write Up

ONLINE BOOKSHOP

December 15, 2024

Aidan Tran, Asiyah Speight, Chantelle Chan, & Ethan E. Lopez

Overview

The BookShop Project is an extensive system designed to simulate the operations of an online bookstore. Containing essential functionalities such as user account management, stock inventory control, and order processing, the interface ensures customers' browsing through a diverse catalog of books, management of shopping carts, and placing orders. Administrators are further empowered to manage book inventories, monitor orders, and maintain the overall system.

Our BookShop system was constructed based on object-oriented programming because it was the best type suited for access controls implemented throughout the program. Assigning different functionalities to specific users made the program more secure and organized, allowing future additions to be made more easily. Overall, the code produced culminates all the topics we learned throughout the semester, embodying the four pillars of OOP.

Description of Classes/Interfaces

1. ***BookShop***

The driver for the bookshop (online or physical). This class manages the core functionality of a bookstore application. Handling user accounts, book inventory, and order processing, the following is a breakdown of its components:

- ➢ <u>Private member variables</u>
  - ○ private User currentUser;
    - ■ Stores the user that is currently logged in to the shop's website
  - ○ private HashMap<String, User> savedUsers;
    - ■ Uses a HashMap to store users that have created an account, with their emails as keys
- ➢ <u>Static Helper Methods</u>
  - ○ userNameGuideLines();
    - ■ Prints the guidelines for usernames, including minimum and maximum character amounts and which characters are allowed

    - passWordGuideLines();
        - prints the guidelines for passwords, including minimum and maximum character amounts and which characters are allowed
    - defaultBooks(LinkedList<Book> 11);
        - called whenever a customer or administrator interacts with books
        - generates a list of eight books, four fiction and four non fiction, to return a LinkedList of these books
        - books are sorted according to their prices in ascending order using the compareTo() method for the Book class (Book class implements Comparable<Book>)
    - printBooks(LinkedList<Book> 11);
        - called whenever books need to be pushed to the terminal
        - takes in a LinkedList of books and prints them with line numbers
        - relies on Book's toString() method to display details
    - listBooks(LinkedList<Book> 11);
        - called whenever a LinkedList of Books need to be saved into a txt file
        - uses a PrinterWriter to Write to the file Books.txt, including exception handling (**try**-catch) to manage potential IOExceptions
    - defaultOrders(LinkedList<Book> ll, Administrator a)
        - called whenever an administrator is interacting with past orders
        - generates five default orders from five default customers with varied order statuses ("Delivered," "Shipped," or "Pending")
        - books in these orders are from the LinkedList of default books
        - demonstrates the use of the Order class and its methods (i.e. addToOrder() and setOrderStatus())
    - listOrders(ArrayList<Order> o)
        - takes in an ArrayList of orders and writes them into the file Order.txt
        - similar to listBooks(), it uses a PrinterWriter and includes **try**-catch for exception handling
- <u>Instance Methods</u>
    - logout()
        - Used in the case when a user is logged in (currentUser != null)
        - Logged-in users are saved into savedUsers HashMap
        - Print logout messages to the console
    - saveUser(User user)
        - If the user's email is not already a key in the savedUsers HashMap, they are added to the HashMap
        - Prints a registration success or failure message after
- <u>Main Method</u>
    - Main method where the program begins
    - Creates a BookShop object (shop) and a Scanner (scnr) for user input
    - User interaction and Account Creation
        - Welcomes the user and prompts them for account information (email, name, username, password, user type)

        - Uses loops (while) to repeatedly prompt for input until valid data is entered
        - Calls userNameGuidelines() and passWordGuidelines() for displaying input rules
        - Creates a Customer or Administrator object based on the users input, sets it as the currentUser.
        - Calls saveUser() to store the new user
        - Allows the user to review or change account details (using methods presumably defined in the Customer and Administrator classes)
- *Book and Order Management:*
    - Calls defaultBooks() to generate a default book list
    - Calls listBooks() to write the book list to "Books.txt"
    - Calls printBooks() to display the book list in the console
    - For customers:
        - Provides options to add books to a shopping cart or purchase directly
        - Implements cart management (add, remove, clear) using methods from the Customer class
        - Simulates order placement and payment using the Order class
        - Updates the book stock and order lists
    - For administrators:
        - Assigns the default book list to the administrators books variable (assuming this variable exists in the Administrator class)
        - Presents an administrator menu with options to manage the bookstore (review/change account, add/remove books, update stock/prices, view/update orders)
        - Calls defaultOrders() to generate sample orders for the administrator to work with
        - Uses a switch statement to handle different administrator actions
- *Logout and Exit:*
    - Calls logout() to log out the current user, after prints a goodbye message and closes the Scanner using scnr.close()

2. *User*

A public abstract parent class that takes in a person's identification information (emails, names, usernames, passwords, etc.) and sets up a regular BookShop account. Two kinds of Users are accepted: Customers and Administrators. This body functions as the foundation for common User functions:

- <u>protected String (email / username / password / firstName / lastName)</u>
    - A "User" is not a defined object; thus, the class was implemented as an abstract
    - Protected variables are Strings for flexibility in handling text and multiple characters
- <u>Overloaded Constructor</u>
    - Default constructors are prohibited when UNKNOWNs cannot operate our website
    - Overloaded constructors assign account information to the right identity sections
    - Copy constructors are illogical because it's not possible to copy an entire account
- <u>Accessor Methods</u>

    - getEmail(), getUsername(), getPassword(), getName()
    - Return methods for Users to “check” details in their account
    - getName() returns both first and last name (you don’t return only your last or first name)
- <u>Mutator Methods</u>
    - setEmail(), setUsername(), setPassword()
    - A User is permitted to change any account details except their name (no catfishing!)
- <u>@Override toString()</u>
    - toString() with User account details for the User to view in whole
- <u>@Override equals()</u>
    - Compares two accounts by the persons’ email and name to see if they are equal
    - Usernames and passwords vary across a single person’s accounts

3. Book

A public abstract parent class for book identification information (title, author, language, price, category, etc.). Creating a Book object, our group wanted to utilize abstract methods for different kinds of Books in Fiction and Nonfiction, so we implemented Book as an abstract to create contracts for all Book types:

- <u>Member Variables</u>
    - protected String (title / author / language / category)
        - Fiction and Nonfiction Books share these details, thus they are protected
        - These variables are most appropriate as Strings because they have letters
    - protected int (ISBN / stock) / protected double (price) / protected boolean (availability)
        - ISBNs (Book IDs) and stock (shelf count) are natural integers
        - Price is a double not only for decimals but adding totals in the Customer’s Order
        - Book availability is true or false depending on the stock # being 0 or not
- <u>Overloaded Constructor</u>
    - All variables create Book objects through parameters except the ISBN, a randomized ten-digit number generated for publishing by the item’s establishment
    - Zero default constructors when there are no null / blank Books
- <u>Copy Constructor</u>
    - Books can be copied so in reality this makes sense
    - All variables are copied except the ISBN, different for every Book’s copy
- <u>Accessor Methods</u>
    - All protected variables have a get() to check each Book’s specific detail
- <u>Mutator Methods</u>
    - setStock(), setPrice(), setAvailability()
    - Stock, price, and availability are the only attributes altered for Books in the BookShop
    - All other book information is held constant
- <u>@Override toString()</u>
    - toString() with Book details for Users and Administrators to view
- <u>@Override equals()</u>
    - A Book is equal to another Book when it shares the same title and author

4. ***Order***

The Order class stores details associated with a customer's purchase within the BookShop system. It represents an individual order placed by a customer, maintaining information such as the order ID, total cost, status, date, associated customer, and the list of books included in the order. This class provides methods to manage the entire process of an order, including adding or removing books, updating order status based on delivery progress, and calculating the total cost.

- Member Variable
    - private int orderID;
        - Uniquely identifies each order within the system
    - private double total;
        - Represents the total cost of the order
    - private String status;
        - Tracks the current status of an order
    - private LocalDate date;
        - Records the date when the order was placed
    - private Customer customer;
        - Associates the order with the customer who placed it
    - private ArrayList<Book> items;
        - Maintains a list of books included in the order
- Constructor
    - Initializes a new instance of the Order class associated with a specific customer
        - The customer field is assigned to the Customer object passed into the constructor
        - Sets total is set to 0.0, the status to "Pending," and initializes the items field
        - The date field is assigned based on the local date of the customer when they made the order
        - A random 5-digit order ID is generated with the order
    - No copy constructor, as the order's date changes based on when the customer made the order, and unique order numbers are assigned to each order
- Methods
    - public void addToOrder(Book book)
        - Adds a specified book to an order
    - public void removeBookFromOrder(Book book)
        - Removes a specified book from an order
    - public String updatedStatus()
        - Updates the status of the order based on the number of days elapsed since the order date
            - It takes 5 days for an order to be shipped and 14 days after ordering for it to be delivered
    - public double calculateTotal()
        - Calculates the total cost of the order and returns the cost
    - public String toString()

        - Provides a string representation of the Order object, including customer details, order ID, date, books included, total cost, and current status
    - public boolean equals(Object o)
        - Determines if two Order objects are equal based on the orders' ID, the customer associated with the orders and the orders' total price
- <u>Accessor Methods</u>
    - All variables get an accessor method for public access to variables for use in other classes, as access to these variables is necessary to run the website
- <u>Mutator Methods</u>
    - Only the order total and order status variables have mutator methods to account for possible discounts that can be retroactively applied to orders or faster processing time of the order than expected
    - All other variables should remain unchanged once an order is created

5. *Customer*

The Customer class is a concrete subclass of the abstract User class. It represents a customer within the BookShop system, storing functions that allow users to interact with the bookstore. This class defines behaviors exclusive to customers, such as managing a shopping cart and placing orders. By extending the User class, the Customer inherits common user attributes while introducing methods tailored to enhance the shopping experience.

- <u>Member Variable</u>
    - protected LinkedList<Book> cart;
        - Manages the list of books that a customer intends to purchase
    - protected LinkedList<Order> orders;
        - Maintains a history of all the orders placed by the customer
- <u>Constructor</u>
    - Initializes a new instance of the Customer class with specified attributes, including the information needed for User objects
- <u>Methods</u>
    - public void addToCart(Book book)
        - Adds a book to the customer's shopping cart
    - public void removeFromCart(Book book)
        - Removes a book from the customer's shopping cart
    - public void placeOrder()
        - Places an order based on the current contents of the shopping cart
            - Creating an Order object to hold order details
    - public void clearCart()
        - Empties the contents of a customer's shopping cart
    - public void firstAdd(LinkedList<Book> ll)
        - Adds books from the shop's default book catalog to a Customer's cart
    - public void firstRemove(LinkedList<Book> ll)
        - Removes books from the Customer's cart

    - public void clearMyCart()
        - Prompts the customer to clear their cart and performs this action
- Accessor Methods
    - All variables have an accessor method for public access to the variables in other classes, as access to these variables is necessary to run the website
- Mutator Methods
    - None, as methods to change member variables already exist
- Overridden Methods/Inheritance
    - @Override public String toString()
        - Provides a string representation of the Customer object, including account details, cart items, and order history.
    - @Override public boolean equals(Object o)
        - Determines if two Customer objects are equal based on cart items

6. *Administrator*

The Administrator class is a concrete subclass of the abstract User class. It represents an administrator within the BookShop system with specific functionalities to manage the book inventory and oversee orders. This class defines behaviors exclusive to administrators facilitating tasks such as adding or removing books, updating book details, and tracking order histories.

- Member Variable
    - protected List<Book> books;
        - Maintains a list of all books currently available in the bookstore.
    - protected ArrayList<Order> completedOrders;
        - Stores all orders that have been completed
    - protected ArrayList<Order> allOrders;
        - Keeps track of all orders placed within the system, regardless of their status.
- Constructor
    - Initializes a new instance of the Administrator class with specified attributes, including the information needed for User objects
- Methods
    - public void addBook(Book book)
        - Allows an Administrator to add a new book to the shop's catalog
    - public void removeBook(Book book)
        - Removes a book from the shop's existing catalog
    - public void updateBookStock(Book book, int newStock)
        - Changes the stock of a specified book in the shop
    - public void updateBookPrice(Book book, double newPrice)
        - Changes the price of a specified book in the shop
    - public Book findBookByTitle(String title)
        - Allows a book in the shop's catalog to be found based on the book's title alone
    - public void viewOrders()

        - Provides an overview of all of the shop’s orders (both completed and ongoing)
    - public void addFinishedOrders()
        - Updates the completedOrders variable to hold any orders marked as “Delivered”
    - public void addOrder(Order order)
        - Passes in an order as a parameter to add to the allOrders variable
    - public void administratorMenu()
        - Provides an overview of all the actions an Administrator can take to interact with the shop’s website
    - public void createBooksToAdd(Scanner scnr)
        - Adds new books to the shop’s catalog based on Administrator input
    - public void removeBookFromStore(Scanner scnr)
        - Removes book from a the shop’s existing catalog based on Administrator input
    - public void updateStock(Scanner scnr)
        - Updates the stock of a book in the shop using Administrator input
    - public void updatePrice(Scanner scnr)
        - Updates the price of a book in the shop using Administrator input
- <u>Accessor Methods</u>
    - All variables have an accessor method for public access to the variables in other classes, as access to these variables is necessary for the implementation of the website
- <u>Mutators Methods</u>
    - None, as methods to change member variables already exist
- <u>Overridden Methods/Inheritance</u>
    - @Override public String toString()
        - Provides a string representation of the Administrator object, including account details and the store’s book catalog
    - @Override public boolean equals(Object o)
        - Determines if two Administrator objects are equal based on books variable (store catalog being the same)

7. *Fiction*

The Fiction class is a concrete subclass of the abstract Book class. It represents fiction books within the BookShop system, encapsulating attributes and behaviors specific to fictional literature. This class allows the system to handle diverse genres and target audiences.

- <u>Member Variables</u>
    - protected boolean bestseller
    - protected String (genre/ageGroup)
        - These member variables are attributes that are specific to the fiction book class
- <u>Constructor</u>
    - Initializes a new instance of the Fiction class with specified attributes
- <u>Mutator Methods</u>
    - public void setBestseller(boolean bestseller)
    - public void setGenre(String genre)

        - public void setAgeGroup(String ageGroup)
            - Updates the age groups of each Fiction book
- Overridden Methods/Inheritance
    - public String getTargetAudience()
        - Returns Fiction-specific information such as the genre of the book and intended age group of readers
    - public String toString()
        - Extends the superclass's toString method by appending fiction-specific details, ensuring comprehensive and formatted output when printed

8. *Nonfiction*

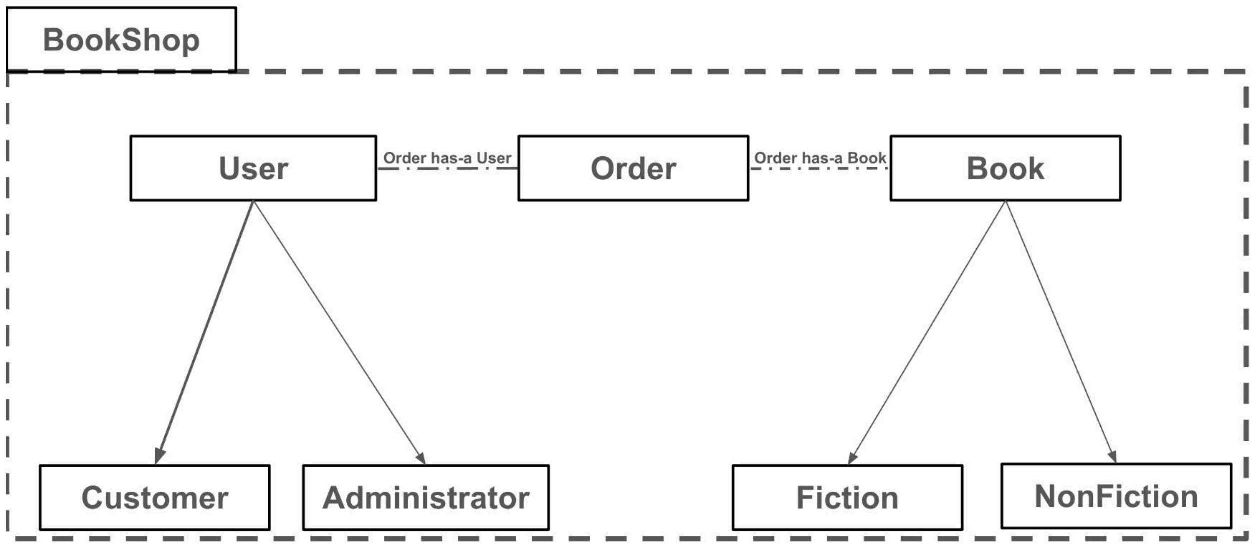
The Nonfiction class is a concrete subclass of the abstract Book class. It represents nonfiction books within the BookShop system, encapsulating attributes specific to characteristics of a nonfiction novel/book. This class enables the system to manage diverse genres, editions, peer-reviewed statuses, and topics of nonfiction books.

- Member Variables
    - protected String genre;
    - protected int edition;*
    - protected boolean isPeerReviewed;*
    - Protected String topic; *
        - The three asterisked member variables are characteristics we deemed unique to a nonfiction book
- Constructor
    - Initializes a new instance of the Nonfiction class with specified attributes
- Mutator Methods
    - public void setPeerReviewed(boolean isPeerReviewed)
    - public void setEdition(int edition)
    - public void setGenre(String genre)
    - public void setTopic(String t)
        - Returns Nonfiction-specific information such as the peer review status and edition
- Overridden Methods/Inheritance
    - public String getTargetAudience()
        - Returns Nonfiction-specific information such as the peer review status of the book and its edition
    - public String toString()
        - Extends the superclass's toString method by appending nonfiction-specific details, ensuring comprehensive and formatted output when printed

9. *Comparable<Book>*

The Comparable interface is aligned to organize Book objects in an order or list. Arranging them by their prices in ascending fashion, we felt this very natural in a BookShop for customers to compare Book prices and choose which ones they were willing to purchase in that mannerism.

Diagram



Architecture Rational

We chose a design that focused on object-oriented programming (OOP) for the MP4 project not only because it was a requirement for our system but because it was the most efficient way to simulate the functionality of an online bookshop. We achieved a clear and logical hierarchy that promoted code reusability, maintainability, and scalability by utilizing abstract classes like User and Book and extending them through specialized subclasses such as Administrator, Customer, Fiction, and Nonfiction. This structure allows us to encapsulate shared attributes and behaviors in the parent classes while enabling subclasses to implement specific functionalities, ensuring that our codebase remains organized and easy to manage.

If we were to design our project sequentially, its structure would have been significantly more fragmented. Unlike OOP, a sequential structure would have made it difficult to have shared functionalities. Those types of design are more rigid in nature due to the lack of inheritance and polymorphism. Additionally, this would have made future changes to the program difficult to implement since it is less flexible. Therefore, OOP was the most intuitive approach to a project like this and the most efficient.

File I/O implementation

In the BookShop system, File Input/Output (File I/O) is handled using a Books.txt file, which stores all the information about the books available in the store. This file uses comma-separated value (CSV) formats, where lines represent books and their details, including title, author, publication year, price, availability, stock, and other specific categories in Fiction or Nonfiction books. When the program starts, it creates Book objects that are read into the Books.txt file line by line, loading all the books into its memory. Converting Book objects into a LinkedList, the program interacts to update this list and txt file through customers adding and removing books and administrators updating the book inventories. Static methods found in our BookShop driver take the updated lists of Book objects and convert them to a CSV String, rewriting them into the Books.txt file whenever changes are made. Write mode ensures any updates made to the file are handled and referenced consistently in the following application runs.

The BookShop system also uses an Order.txt file, which stores all the relevant information about a customer's order. This file is organized the same way as the Books.txt file, with the only difference being the information stored in it. It also uses the same processes implemented before so the program can efficiently read from and write into the file.

Java packages/API usage

The BookShop system does not implement any packages other than the ones already incorporated in standard Java libraries. It was decided that in order to maintain simplicity and optimal functionality for the user, no other additional packages were needed. This decision not only streamlined the development process by reducing the need for configurations but also eliminated the time-consuming error checking process that could have happened with packages. However, to manage various events, such as order placement and delivery dates, the application leverages Java's modern java.time API, specifically the ChronoUnit enumeration. This allows for precise calculations like determining the number of days between two dates, ensuring accurate tracking of order statuses and delivery timelines. Utilizing ChronoUnit not only simplifies these operations but also enhances the system's reliability in data-time management. As a result, the BookShop system remained efficient because of its simpler implementation.

Distribution of Work

Our team comprised a total of four people. Aidan was responsible for the Book subclasses, Fiction and Nonfiction, and helped check/improve various other classes as needed for the project, also handling a majority of the write-up. Chantelle was responsible for the Order and Customer classes and assisted with the Administrator implementation in the BookShop Driver class. Asiyah was responsible for the Administrator class and portions of the Customer classes, also working in conjunction with Aidan to finish the write-up and handle final checks and tests before submission. Ethan was responsible for User, Book, and a majority of the BookShop class.

Future Development

In the future, we see our BookShop system able to implement more advanced Java packages and external APIs. The initial development of this system excluded such ideas due to time commitments. However, when given the opportunity, others could expand on this system by utilizing databases rather than flat file storage .txt files, significantly improving data management with enhanced structure. Handling significantly larger amounts of data, allowing the overall system to be scaled with larger magnitudes. Another modification that could significantly improve user experience would be to incorporate a more digital user interface. Although this concept is beyond the scope of our own knowledge and would likely take months to perfect, it would be a significant improvement over the current console-based user interface. Of course, there are many possibilities out there to improve this system, but these previously mentioned options are the most appealing considerations.