Introduction

This report outlines the design and implementation of a Bookstore Application developed as part of the COE528 course final project. The application uses JavaFX for the graphical user interface to manage the bookstore, which allows two types of users to interact with the system: owner and customers. The owner can oversee the bookstore by adding and removing books/customers, while the customers can browse books, purchase books, or redeem points for discounts on their purchases. The project utilizes object-oriented programming fundamentals and implements the State Design Pattern to organize customer status transitions.

Overview

The Bookstore Application is a single window GUI application developed using JavaFX. It allows user login authentication (owner and customers), book inventory management (add/delete books), customer management (add/delete customers), book purchasing functionality, points collection and redeeming, and customer state management (silver/gold) based on points. The application stores and loads all book and customer data using two files: customer.txt and book.txt, which store customer and book information respectively. This data is loaded into the application at launch and updated when closed.

Use Case Description

Name: Purchasing Books

Actors:
- Customer (Primary actor)

Entry Conditions:
- This use case starts when the customer meets the following criteria:
    - The customer has successfully logged into the system
    - The customer is viewing the book selection screen

Flow Of Events:
1. The customer views the available books which is displayed in a table with columns for book name, book price, and a select checkbox
2. The customer selects one or more books by checking off the corresponding checkboxes
3. The customer clicks either the [Buy] or [Redeem Points and Buy] button
4. If [Buy] is clicked:
    - The system calculates the total cost by summing the prices of all selected books
    - The system awards the customer with 10 points per CAD spent by the customer
    - The system displays the total cost, updated points, and the customers status on the cost screen
5. If [Redeem Points and Buy] is clicked:
    - The system calculates the subtotal cost by summing the prices of all selected books

- The system applies the available points to calculate the discount, 1 CAD per 100 points
- The system calculates the total cost by subtracting the discount from the subtotal
- The system awards the customer with 10 points per CAD spent by the customer
- The system displays the subtotal, discount, total cost, updated points, and the customers status on the cost screen

6. The system updates the customers status based on their points:
    - Silver: less than 1000 points
    - Gold: more than 1000 points

Exit Conditions:

- This use case concludes under the following circumstances:
    - The purchase is complete, and the customer cost screen displays the transaction details
    - The customer logs out of the application, returning to the login screen
    - The customer navigates back to customer start screen to perform additional actions

Exceptions:

- No books selected: If the customer attempts to proceed to [Buy] or [Redeem Points and Buy] without selecting a book, the system displays an error message
- Insufficient points: If a customer attempts to [Redeem Points and Buy] without the sufficient points to reduce the total cost, the system will displays an error message

Special Requirements:

- Nonfunctional Requirements:
    - The GUI must reflect real time changes to in points, status, and transaction costs
    - All calculations must execute instantly with precision
- Constraints:
    - Customer status updates must happen in real time as points change
    - Total cannot be negative after points redemption
    - Points calculation and redemption must be done according to specific rules (10 points per CAD spent, 100 points = 1 CAD discount)

State Design Pattern Implementation

Rationale:

In order to meet the needs of the changing customer status, a Stage design pattern was used. This ensures that the statuses of the customer can be efficiently managed and opens opportunities for future implementations of customer features while still very maintainable. The State design pattern allows for changes in behaviour of the customer when the [Status] silver or gold is achieved which can be used to apply various discounts to the books being bought. The simple implementation and readability makes it easy to replicate or further expand on future management/purchasing systems for both the owner and the customer. For example, the gold status allows for the user to get big discounts on the books, incentivising them to use the service more if they regularly read books. A future extension of this idea could be a diamond status that could be implemented alongside the addition of audiobooks in the BookStore. This

diamond status could allow the customer to get bundles of audio and physical books for free with enough purchases. The State design pattern allows possibilities like this to be easily implemented into the BookStore app with the intentions of the behaviour being reusable and easy to maintain.

Implementation:

The State Design Pattern was implemented with the following components:

1. **CustomerState Interface:** Defines the common interface for all concrete state classes, which contain the methods to return the appropriate status
2. **Concrete State Classes:**
    - **Silver:** Implements the behaviour and status of customers with less than 1000 points
    - **Gold:** Implements the behaviour and status of customers with more than 1000 points
3. **Customer (Context Class):** Maintains a reference to the current state objects and allows CustomerState to handle state-specific behaviour
4. **State Transitions:** Implemented in the Customer class, which checks the customers points and changes the state object accordingly

Conclusion

The Bookstore Application successfully implements a functional application for owners and customers. By utilizing the State Design Pattern and object-oriented programming principles we were able to design a single window GUI application which effectively handles customer status changes, while ensuring a clean user experience.