COE 528 W2023 Project Report

Use Case Descriptions:

The following are descriptions of various use cases that are present in the bookstore application/project, which follow the seven-step model mentioned in class. To summarize, these steps are the following:

1. A (unique) Name for the use case.
2. Participating Actors - A list of actors interacting with the use case.
3. Entry Conditions - Conditions which initiate the use case.
4. Flow of Events - A numerical sequence of interactions with the use case.
5. Exit Conditions - Conditions which end the use case.
6. Exceptions - Special cases of the use case which only happen given certain situations.
7. Special requirements - Extra requirements not directly related (or stated) to the functionality.

However, to demonstrate a full understanding of the different use cases, the inclusion of a **goal** for each case has been shown - this effectively allows those not familiar with the specifics of our group's project to understand the objective of the use case. Additionally, while the project instructions weren't very specific as to how many use cases to include in this section of the report, our group has decided to include all of the **main** use cases; these are the most common actions the user can take with our project. For simplicity's sake, and for enhanced readability/understanding, use cases which are extremely similar in function to other cases have been clearly denoted, with **only** key differences being cited for these cases. This means that whatever hasn't been cited is the same as its more descriptive counterpart.

Below is the list of use cases, with formatting as described above:

Use Case Name: OwnerLogin

Participating/Primary Actor: Owner

Goal: To log in to the Book Store application and access the ***owner-start-screen***.

Entry Conditions:

- The Book Store application is installed and launched on the device.
- The user has the correct login credentials (username: admin, password: admin).

Flow of Events:

1. The owner launches the Book Store application.
2. The ***login-screen*** is displayed.
3. The owner enters the correct username and password.
4. The owner clicks on the **[Login]** button.

5. The Book Store application verifies the username and password.
6. If the verification is successful, the ***owner-start-screen*** is displayed.

Exit Conditions:

- The owner is logged in to the Book Store application.
- The ***owner-start-screen*** is displayed.

Exceptions:
6a. If the verification fails, the owner is not logged into the application.

Special Condition(s): Based on the information provided in the project description, it is assumed that there is only one owner account.

Similar Use Case Name: CustomerLogin

Key Differences:

- **Participating/Primary Actor:** Customer
- **Goal:** To log in to the Book Store application and access the ***customer-start-screen***.
- **Flow of Events:** The ***customer-start-screen*** is displayed if the verification is successful.
- **Special Condition(s):** Based on the information provided in the project description, it is assumed that only one customer is attempting to log in at a given time (allowing the project to manage resources efficiently and accordingly).

Use Case Name: AddBook

Participating/Primary Actor: Owner

Goal: To add a new book to the Book Store application.

Entry Conditions:

- The owner is logged in to the Book Store application.
- The ***owner-start-screen*** is displayed.

Flow of Events:

1. The owner clicks on the **[Books]** button in the ***owner-start-screen***.
2. The ***owner-books-screen*** is displayed.
3. The owner enters the book's name and price in the Name and Price fields, respectively.
4. The owner clicks on the **[Add]** button.

5. The Book Store application adds a new row containing the book's information to the table in the ***owner-books-screen***.

Exit Conditions:

- A new row containing the book's information is added to the table in the ***owner-books-screen***.

Exceptions:

5a. If an invalid name or price is entered, an exception is raised and the book is not added to the table.

Special Condition(s): Based on the information in the project description, it is assumed that only one book can be added to the database at any time. Additionally, it is assumed that the book information cannot be updated - instead, a new book could be added (or an already existing book can be deleted).

Similar Use Case Name: AddCustomer

Key Differences:

- **Goal:** To **register a new customer** to the Book Store application.
- **Exit Conditions:**
    - A new row containing the **customer's information** is added to the table in the ***owner-customers-screen***.
    - The points for the newly added row are **set to 0.**
- **Flow of Events:**
    - The Book Store application adds a new row containing the **customer's information** to the table in the ***owner-customers-screen***.
    - The points for the newly added row are **set to 0.**
- **Special Condition(s):** Based on the information in the project description, it is assumed that only one customer can be added to the database at any time.

Use Case Name: DeleteCustomer

Participating/Participating/Primary Actor: Owner

Goal: To delete a customer from the Book Store application.

Entry Conditions:

- The owner is logged in to the Book Store application
- The ***owner-start-screen*** is displayed.
- The ***owner-customers-screen*** is displayed.

Flow of Events:

1. The owner selects a row in the table in the ***owner-customers-screen***.
2. The owner clicks on the **[Delete]** button.
3. The Book Store application deletes the selected row from the table in the ***owner-customers-screen***.
4. The Book Store application removes the customer information corresponding to the deleted row from the Book Store application.

Exit Conditions:

- The selected row is deleted from the table in the ***owner-customers-screen***.
- The customer information corresponding to the deleted row is removed from the Book Store application.

Exceptions:

4a. If there are no customers selected, nothing happens.

Special Condition(s): It is assumed that once a customer's information is removed from the system, it is done properly, such that the information can no longer be accessed via the database.

Similar Use Case Name: DeleteBook

Key Differences:

- **Goal:** To **delete a book** from the Book Store application
- **Entry Conditions:** The ***owner-books-screen*** is displayed
- **Flow of Events:** The same flow of events occurs, but instead of it happening on the owner-customers-screen, it happens on the ***owner-books-screen***. Additionally, the deletion of information occurs for the **selected book** rather than the customer information.
- **Exit Conditions:**
    - The selected row is deleted from the table in the ***owner-books-screen***.
    - The book information corresponding to the deleted row is removed from the Book Store application.
- **Special Condition(s):** It is assumed that once a book's information is removed from the system, it is done properly, such that the information can no longer be accessed via the database.

Use Case Name: BuyBooks

Participating/Primary Actor: Customer

Goal: To buy one or more books from the Book Store application.

Entry Conditions:

- The customer is logged in to the Book Store application.
- The ***customer-start-screen*** **i**s displayed.

Flow of Events:

1. The customer selects one or more checkboxes in the table in the ***customer-start-screen***.
2. The customer clicks on either the **[Buy]** button or the **[Redeem points and Buy]** button.
3. The Book Store application calculates the transaction cost based on the selected books and the number of points to be redeemed (if applicable).
4. The Book Store application updates the customer's points and status based on the transaction cost
5. The Book Store application displays the transaction cost in the *customer-cost-screen*.
6. The customer clicks on the **[Logout]** button to log out of the Book Store application.

Exit Conditions:

- The transaction cost is displayed in the ***customer-cost-screen***.
- The customer's points and status are updated accordingly.
- The selected books are removed from the table in the ***customer-start-screen***.

Exceptions:

4a. If the transaction cost is less than 0, the Book Store application sets the transaction cost to 0 and updates the customer's points and status accordingly.

Special Condition(s): None

Use Case Name: Logout

Participating Actors: Owner or Customer

Goal: To log out of the Book Store application.

Entry Conditions:

- The owner or customer is logged in to the Book Store application.

Flow of Events:

1. The owner or customer clicks on the **[Logout]** button.
2. The Book Store application logs out the owner or customer and returns to the ***login-screen***.

Exit Conditions:

- The owner or customer is logged out of the Book Store application.

Exceptions: None.

Special Condition(s): Based on the information in the project description, it is assumed that once logged out, the user cannot directly return to the previous screen. Instead, the same process has to be repeated to get to that point again.

These are all of the **main** use cases associated with the project in its current state. However, these use cases can easily be adjusted depending on the project's requirements, should they be changed. This could mean the inclusion of more use cases for specific states of the project (ex., a “search” feature for more information about a book - something that hasn’t been implemented due to lack of necessity but which could have a unique use case if necessary), or the need for adjustment of the already existing use cases.

State Design Pattern Rationale:

The state design pattern was used to implement the Status classes. The state design pattern allows objects to have varying behaviours depending on their state. For this project, customers were able to have different statuses based on their number of points: Silver or Gold. Their status is displayed on the GUI in certain screens. We created an abstract Status class which contained a getStatus method. The Status class was implemented in the SilverStatus and GoldStatus classes, each of which had their own implementation of the getStatus method.

An advantage of this design pattern is that, when displaying the customer’s status, we did not have to manually check for the status. Assuming that the customer’s internal state is correct, the getStatus method would always return the correct value (“Silver” for SilverStatus, “Gold” for GoldStatus.) Another advantage of this pattern is its flexibility. If we were to implement a new customer status, we would not have to change any of the GUI code. We could simply create a new class that implements the Status class. It would have no impact on the existing statuses, and we could continue to use the getStatus method to display the customer’s status.