Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	**Object Oriented Eng Analysis Desgin**
Course Number:	**COE528**
Semester/Year (e.g.F2016)	**W2023**

Instructor:	**Olivia Das**

Assignment/Lab Number:	***Final Project***
Assignment/Lab Title:	***Final Project***

Submission Date :	**April 2nd, 2023**
Due Date:	**April 2nd, 2023**

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Najafi	**Ahmad**	**501124585**	**07**	**A.N**
Parasteh	**Ali AJ**	**501021960**	**07**	**A.P.**
Khalifa	**Redwan**	**501028027**	**07**	**R.K.**
Amir Mohammad Nuri	**Ahmad Reshad**	**501102500**	**07**	**A.A**
Rahman	**Iwan**	**501105526**	**03**	**I.R**

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/content/dam/senate/policies/pol60.pdf

COE 528 Final Project Report

Use Case Discription:

• **Participating actors:** Owner, Customer

• **Entry conditions:** The user has successfully logged in as either an Owner or a Customer.

• **Flow of events:**

1. The user enters their username and password in the login screen
2. The system validates the provided credentials against the list of registered customers and owner credentials
3. If the credentials are valid and the user is an owner, the system grants access to the owner menu scene where the owner can manage books and customers.
4. To add a new book into the book table, the owner enters the book name and the price of the book then clicking the [Add] button. The owner can delete a book from the book table by clicking the specific row and clicking the [Delete] button.
5. To add a new customer to the customer table, the owner enters the customer's username, password then clicks the [Add] button. The owner can delete a customer from the customer table by clicking the specific row and clicking the [Delete] button.
6. If the credentials are valid and the user is a customer, the system grants access to the customer menu scene where the customer can browse and purchase books.
7. To purchase one or more books, the customer will browse the catalog and select the desired books by checking the corresponding checkboxes on their rows. They can proceed to make the purchase by clicking either the [Buy] button or [Reedem and Buy] button. If the customer opts to "redeem and buy", 100 points will be subtracted from the total cost for every 1 CAD. If the customer chooses purchase the books without redeeming any points, they simply need to click [Buy] button.
8. If the credentials are invalid, the system displays an error message, and the user must re-enter their credentials or exit the application.

• **Exit conditions:**

1. Owner logs out, returning to the login screen.
2. Customer logs out, returning to the login screen.

• **Exceptions:**

1. Invalid login credentials: Display an error message and prompt the user to try again.
2. Insufficient points for redeeming: Display an error message and prompt the user to choose another option.
3. Adding duplicate books or customers: Display an error message and prevent duplicate entries.

• Special Requirements:

1. After a customer purchases a book, the book should be deleted from the book table

Rationale Behind Using State Diagram

The State Design Pattern is a behavioral design pattern that allows an object to change its behavior based on its internal state.When an object has multiple states with distinct behaviours and these states need to be managed or switched between with ease, this pattern is especially helpful. It promotes the separation of concerns and enables a cleaner, more organized, and maintainable codebase.

In the context of the bookstore application, the design pattern can be applied to manage the different statuses of states of a customer based on their reward points. The customers can have different states, such as Silver or Gold, which might impact the customer's visits. As a customer earns reward points, their state may change from Silver to Gold, or vice versa. By using the State Design Pattern, the customer object can delegate state-specific behavior to the current state object, allowing for smooth transitions between states without the need for complex conditional statements in the customer class. If we were to implement the same application without the use of the state design pattern we would need the use of large conditional statements to handle state-specific behaviour, leading to complex and hard to maintain code. By instead implementing the state design pattern, each state's behaviour is encapsulated in a separate class which eliminates the need for such conditional statements and simplifies the code. Furthermore, if a new customer (e.g) platinum needs to be introduced, it can be easily added by creating a new state class without modifying the existing customer or state classes.

Moreover, the state design pattern can also be applied to manage the different states of the Person class, with both the Owner and Customer as subclasses of Person. In this case, the Person class can act as the context for the state pattern, with Owner and Customer being the concrete state classes. Both Owner and Customer share common attributes, such as login credentials, while having distinct behaviors and privileges within the application.

By applying the state design pattern to the bookstore application, the management of the owner and customer states and their associated behaviours becomes more organized, maintainable, and extensible which ultimately leads to a cleaner and more efficient code.