Ryerson University

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

Course Title:	Object Oriented Analysis Design
Course Number:	COE 528
Semester/Year (e.g.F2016)	W2022

Instructor:	Dr. Olivia Das

Assignment/Lab Number:	Project
Assignment/Lab Title:	Project

Submission Date:	April 3, 2022
Due Date:	April 3, 2022

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Saini	Harsanjam	501055402	07	HS
Zaitoun	Tariq	501023061	07	TZ
Yousaf	Hamaad	501042570	07	HY
Thakkar	Parva	501033532	07	PT

*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:
http://www.ryerson.ca/senate/current/pol60.pdf

Description of Use-Case:

1. **Name of Use Case:** BookStoreApp
2. **Actors**: The actors for the BookStoreApp are Owner and Customer.
3. **Entry Condition:** The Use Case starts when Owner or Customer are logged into the App.
4. **Flow of Events:**
    a. When the owner clicks the **[Books]** button, they will be presented to an *owner-books-screen.* To add a book to the table, the Owner would type the Name and Price in the text field and click the **[Add]** button. To delete a book from the table, the owner would select the row of the book and press the **[Delete]** button. If the Owner presses the **[Back]** button, they will be taken to the *owner-start-screen.* When the owner clicks **[Logout]** button, they will be taken back to the Login screen.

    b. When the owner clicks the **[Customers]** button, they will be taken to the *owner-customers-screen*. To add a customer to the BookStoreApp, the Owner would type the Username and Password into the text fields and then press **[Add]** button. To delete a customer from the table, the owner would select the corresponding row and click the **[Delete]** button. If the Owner presses the **[Back]** button, they will be taken to the *owner-start-screen.* When the owner clicks **[Logout]** button, they will be taken back to the *Login-screen*.

    c. A customer can buy one or more books by selecting the checkbox next to the books they would like to purchase and then click **[Buy]** button. If the customer wants to redeem their points, then they would click **[Redeem and Buy]** button and 1 CAD will be deducted from the total cost of every 100 points redeemed. If the customer clicks the **[Logout]** button, they will take back to *the Login-screen*.

    d. After the customer clicks **[Buy]** or **[Redeem and Buy],** the total transaction cost will be calculated. If the customer did not redeem any points, the new points are updated such that the customer receives 10 points for every 1 CAD spent. The status of each customer is updated to either Gold or Silver depending on the customers points earned.

5. **Exit condition:** The Use Case terminates when Owner or Customer are logged out of the App.
6. **Exceptions:** An exception would occur when the password or username is incorrect. If this occurs, then it would print “Wrong username or password”. If the Customer does not select a book and clicks [Buy], it would print “No books were selected.” Furthermore, if the customer tries to [Redeem points & Buy] and has less than 100 points, it would print “You must have 100 points or more to redeem”.
7. **Special requirements:** It is assumed that the owner has only one copy of a book. It is also assumed that a few customers are already registered to the system with Username, Password and Points. Similarly, a few books are already added to the system.

Rationale behind using State Design Pattern:

The classes participating in the state design pattern are Customer.java, State.java, Silver.java, and Gold.java. The state.java class was an abstract class that had blank methods which were able to be overridden by the Silver and Gold classes. The Silver and Gold classes override these methods and perform the function of the methods. Lastly, the Customer.java class acts as the client class and interacts with these state methods.

The state pattern was used to differentiate between the status of the customer being either gold or silver. When a new customer was created by default the state was set to silver. For an existing customer, the states were changed depending on the points of the customer. The state design pattern provided an easier and more efficient way to set and change states. It made the project code less complex by allowing all the state operations to be handled in 3 simple java classes.