Describe a use-case following the format discussed in class

1. ***Name:*** Buy Books
2. ***Participating Actor:*** Customer
3. ***Entry Conditions:***
- Customer enters the BookStore website
- Customer enters correct login credentials

4. ***Exit Condition:***
- Customer buys the book

5. ***Flow of events:***
1) Customer selects the amount of books they want
2) Customer decides if they want to buy the books or redeem their points to buy the books
3) System showcases a new screen in where it showcases the total costs, points with status and a logout option
4) System removes the book and Customer is issued for the book(s) that was purchased

6. ***Special Requirements:*** None

Describe the rationale behind using the State Design Pattern

When it came to this project, the group believed that the best way to efficiently code was to use the state design pattern. The state design pattern was used in the Customer class. The reason why it was implemented in the Customer class was because of the points and status system that was within the class. A customer accumulates points depending on how many books they buy and those points can be redeemed to get books as well. On top of that, the customer's status is dependent on the total number of points the customer has. If a customer has less than 1000 points, they have a silver status and greater than or equal to 1000 points results in a gold status. Since the points and status of a customer is constantly changing when they buy a book, the group implemented a state interface in which we would have the silver state and gold state inherit from it. The state interface would then have an aggregation relationship with the customer. This allows the customer's status to be switched at runtime, without the need of extra conditional statements within the customer class. Overall, the state design pattern in the customer class created more flexibility and readability when designing this application.