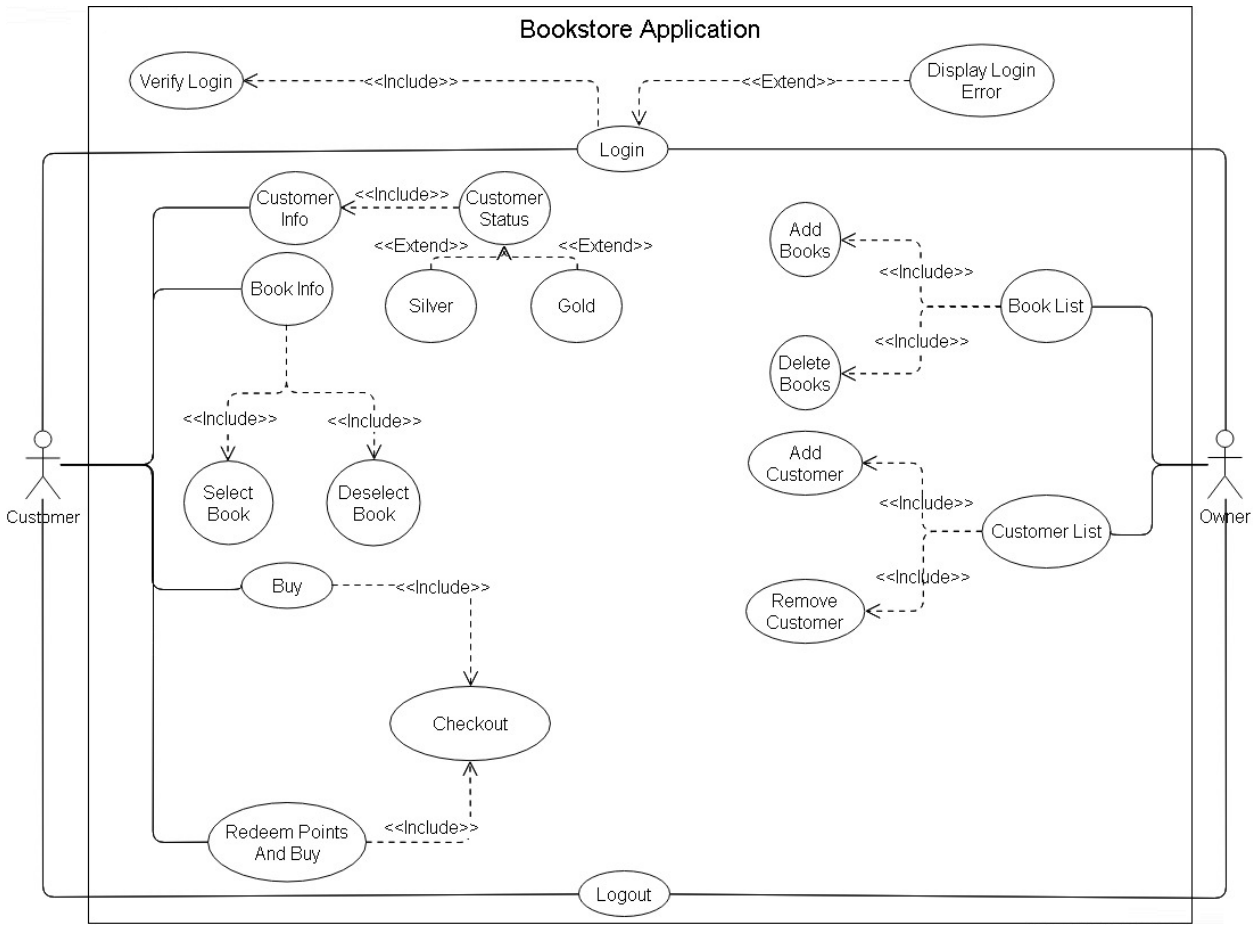
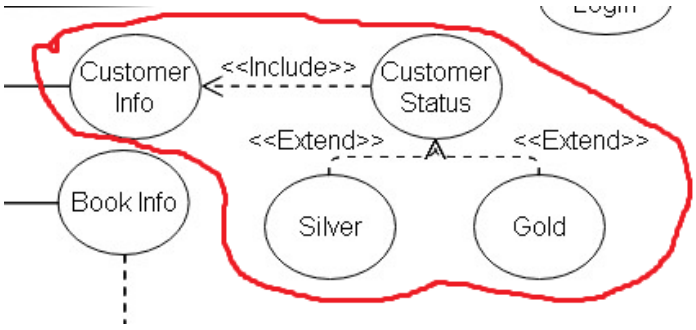
Describe a use-case following the format discussed in class:

The format of a use-case relies on the relationship between the administrator of the system and the users of the system. The relationship starts with the demand of the customers who are going to need the system. To fulfill the demands, the system has to offer the conditions the customer wants. Interact for this. Some special requests for specific customers, and countermeasures if things do not go as planned.

For example, for our project's case the use-cases occur from the demand for books, and the relationship would begin with the customer and the bookstore. The resulting use-case diagram looks like this:



The use-cases would be the interactions in the system for the owner and the customer to make a book transaction. The first use case would be the logging in to the system. The second would be the owner creating the customer and the books for the customer to buy. The final use-cases would be customers' information and the interaction to make the transaction. The special would be the ranking system for customers who like the competition, progress, be rewarded and proud about it. This is shown in this part:

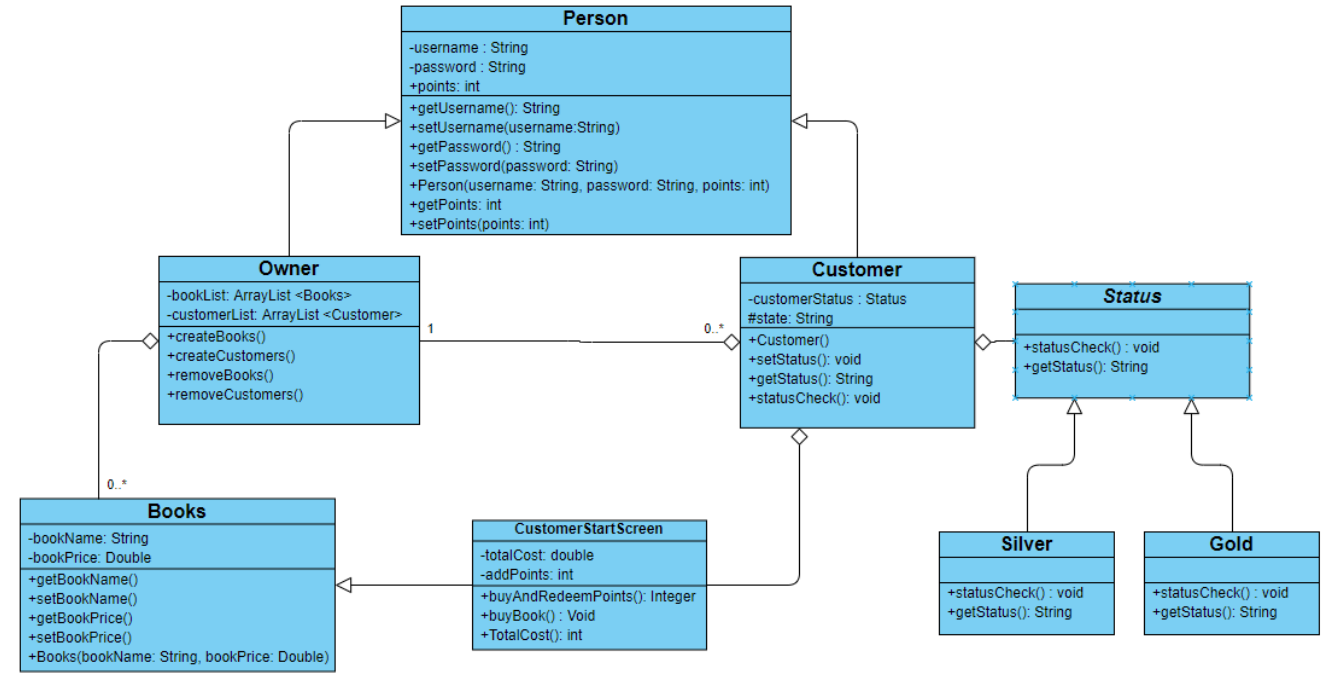


And the countermeasures would be the cases where the customers make accidental mistakes while using the system like entering their login information incorrectly. In that case, the system would have a countermeasure like giving an error for the interaction.

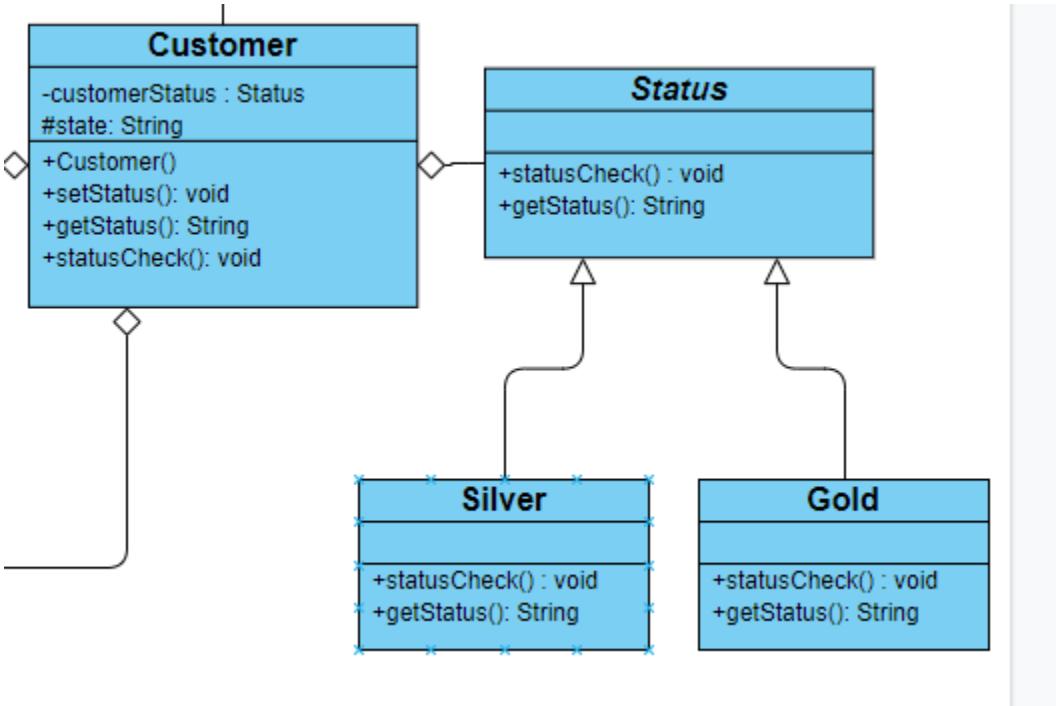
Describe the rationale behind using the State Design Pattern:

State design pattern is used mostly in situations where the status of a particular system needs to be changed based on the conditions. This type of design pattern is more likely to be used when there are several states such as status of a network connection. Network connection can be in of the many states which are established. opened and closed. This sort of problematic scenario requires the system to use a pattern which enables the object to be in one of the states. Similarly, in this project, the customer's status is identified and labeled based on the number of points accumulated through purchasing the books on the system.

The image below shows the UML Case Diagram that uses the State Design Pattern to make the design process more efficient and effective.



The portion of the diagram that uses State Design Pattern is illustrated below:



From the illustration above, it can be deduced that the design follows the generic structure. There are only two statues which include gold and silver. The status changes from silver to gold when

the customer buys enough books to have points greater than 1000. Otherwise, the customer with points lower than 1000 will have a silver status. Inorder to implement the pattern, two classes are created: Silver and Gold. Each of those inherit from the status class which then has an aggregation relationship with the Customer.

When the customer decides to redeem points, the total points decreases and the conditions are checked to see if the sum of points are greater than or equal to 1000. Based on the conditions, the state of the object is changed. The customer's status is then updated according to the points deducted after redeeming the points.

Upon thoroughly analyzing the situation, it is clear that the state design pattern is the most effective choice for the given scenario. It has numerous benefits that outweighs the drawbacks.

The State pattern reduces conditional complexity by removing the need for if and switch statements in objects with specific behaviour specifications for different state transitions. It is fairly simple to transform the diagram into the State design pattern's types and methods if it can represent the object's state using a finite state machine diagram.

The possible reasons for implementing State Design Pattern include:-

- Add new states without altering the state classes or the meaning.
- Separate the code for different states into different classes
- Eliminate bulky state machine conditionals from the context's code to make it simpler.

Using any other patterns to process a customer's status might be more complicated and time consuming. The design would have to undergo several iterations that could make the program lengthy and unexecutable. Using a design pattern like State Design helps to clarify and provide an easy solution to process the customer status and make necessary updates based on the total points gained.