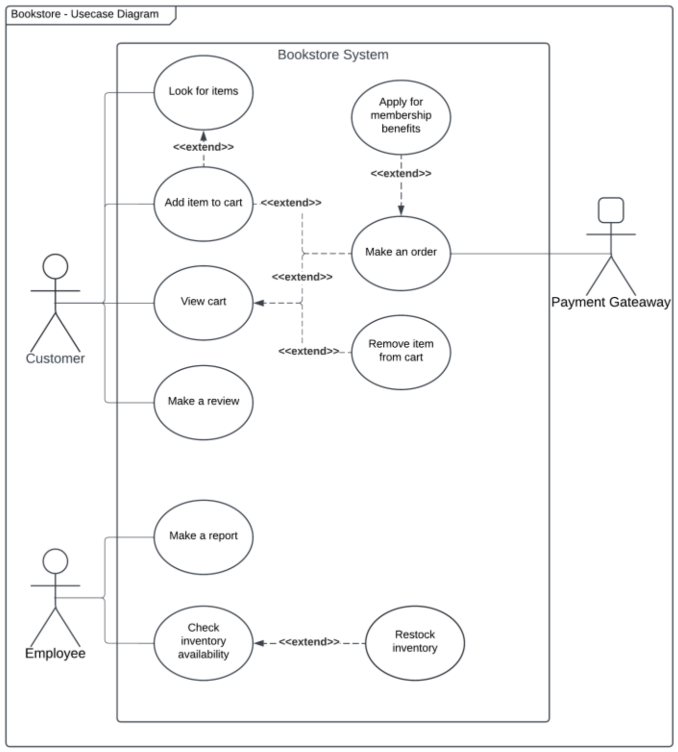
Bookstore

User Requirements

Bookstore is aiming to establish an online system to streamline its operations and improve customer experience by providing a seamless platform for customers and employees. Customers will be able to browse and purchase products, while employees will manage inventory, orders, and daily tasks efficiently. The system will allow customers to search through a wide selection of products, including books, accessories, and other items, and view detailed descriptions with attributes like title, price, and availability. For books, additional details such as genre and author will be provided, while accessories will include material type. Customers can easily add items to their cart, modify the contents, and proceed to place an order by providing shipping details and selecting a payment method, such as cash, card, or membership discounts. During checkout, the system will dynamically apply eligible promotions or discounts based on predefined rules, such as coupon codes or membership status, ensuring a personalized shopping experience.

Use case diagram



Use Case Scenario: “Make an order”

Make an order - use case scenario

Actor: Customer

Purpose and Context: A customer wants to complete the purchase of items they have added to their cart by providing necessary details and making a successful payment

Assumption: The customer has added at least one item to their shopping cart

Precondition: The customer has a shopping cart with items ready to be checked out

Basic Flow of Events:

1. The customer clicks on the "Continue to Checkout" button.
2. The system applies available discounts and promotions to the items in the cart.
3. The system displays the updated cart information (items, prices, discounts).
4. The customer provides personal information
5. The customer provides shipping details
6. The customer enters payment details
7. The system processes the payment:
8. The system generates a shipping tracking number.
9. The system notifies the customer of the order confirmation.
10. The system sends shipping details to the customer.

Alternative Flows of Events:

Payment Failure:
7a1 - The system displays a message: "Payment failed. Please try again with a valid payment method."
7a2 - The customer retries payment

Postconditions:

Basic: The order is successfully placed, and the customer receives confirmation and shipping details.
Payment Failure: No order is placed; the process ends unless the customer retries payment.

Use Case Scenario: “Add item to cart”

Add Item to Cart - use case scenario

Actor: Customer

Purpose and Context: A customer wants to add an item to their shopping cart after viewing its details on the website or application.

Assumption: The item the customer wants is listed on the website or application.

Precondition: The customer has access to the website or application and can browse the catalog.

Basic Flow of Events:

1. The customer clicks on an item they are interested in.
2. The system displays the detailed information about the selected item (e.g., description, price, availability).
3. The customer selects the desired quantity for the item.
4. The customer clicks on the "Add to Cart" button.
5. The system checks the availability of the selected quantity: If the quantity is available, proceed to step 6.
6. The system adds the selected item and quantity to the customer’s shopping cart.
7. The system prompts the customer with two options: **Continue Shopping** or **Go to Cart**.
8. If the customer clicks **Continue Shopping**, they return to the main catalog page.
9. If the customer clicks **Go to Cart**, the system displays the shopping cart.

Alternative Flows of Events:

Item Not Available:

5a1 - The system displays a message: "Item is unavailable at the moment."

5a2 - The customer may choose a different item or exit the process.

Customer Proceeds to Checkout:

7a1 - This leads to the "Make an Order" use case.

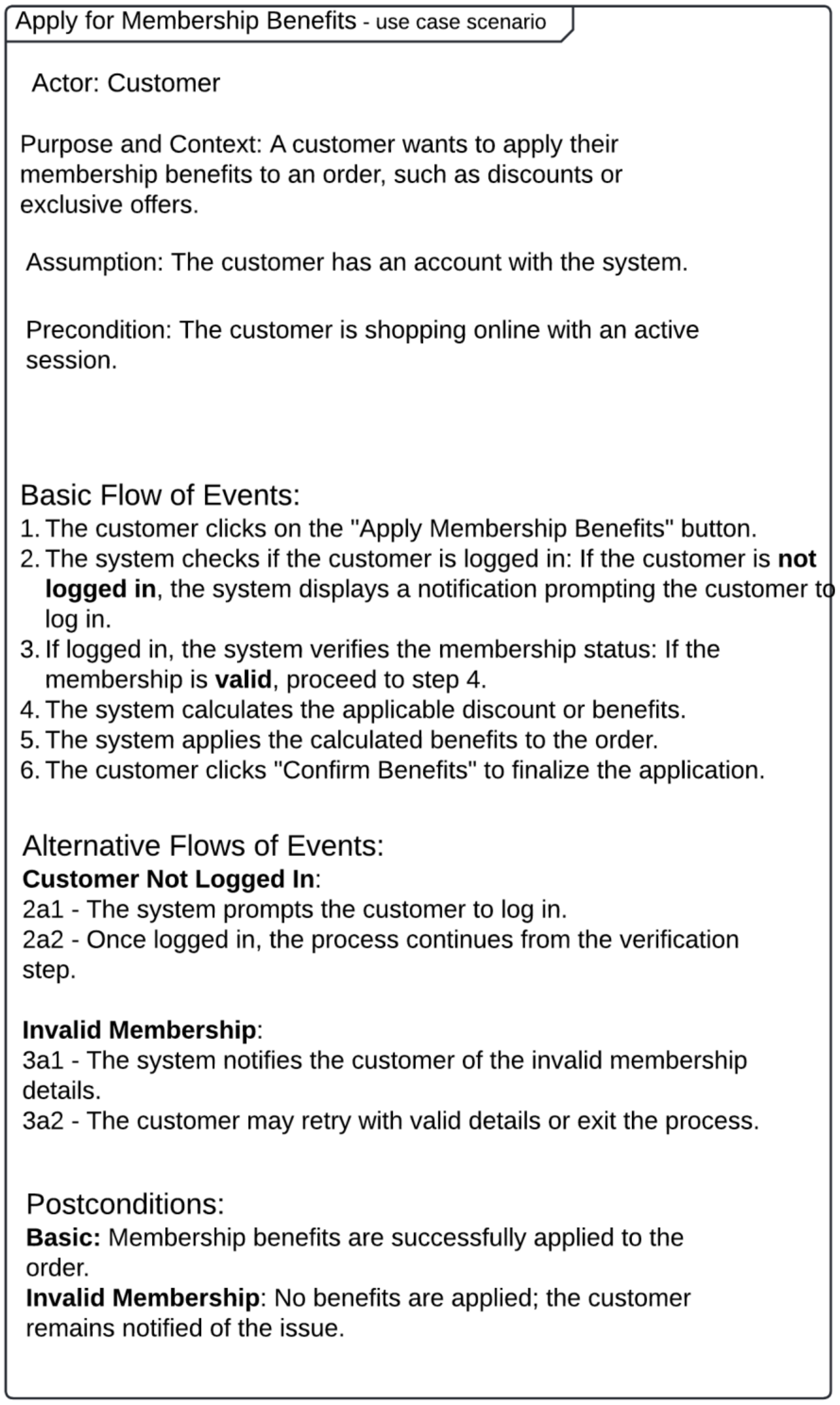
Postconditions:

Basic:
- The item is successfully added to the cart with the selected quantity.

Item Not Available:
- No changes are made to the cart.

Proceed to Checkout:
- The customer initiates the checkout process.

Use Case Scenario: ”Apply for membership benefits”



Apply for Membership Benefits - use case scenario

Actor: Customer

Purpose and Context: A customer wants to apply their membership benefits to an order, such as discounts or exclusive offers.

Assumption: The customer has an account with the system.

Precondition: The customer is shopping online with an active session.

Basic Flow of Events:

1. The customer clicks on the "Apply Membership Benefits" button.
2. The system checks if the customer is logged in: If the customer is **not logged in**, the system displays a notification prompting the customer to log in.
3. If logged in, the system verifies the membership status: If the membership is **valid**, proceed to step 4.
4. The system calculates the applicable discount or benefits.
5. The system applies the calculated benefits to the order.
6. The customer clicks "Confirm Benefits" to finalize the application.

Alternative Flows of Events:

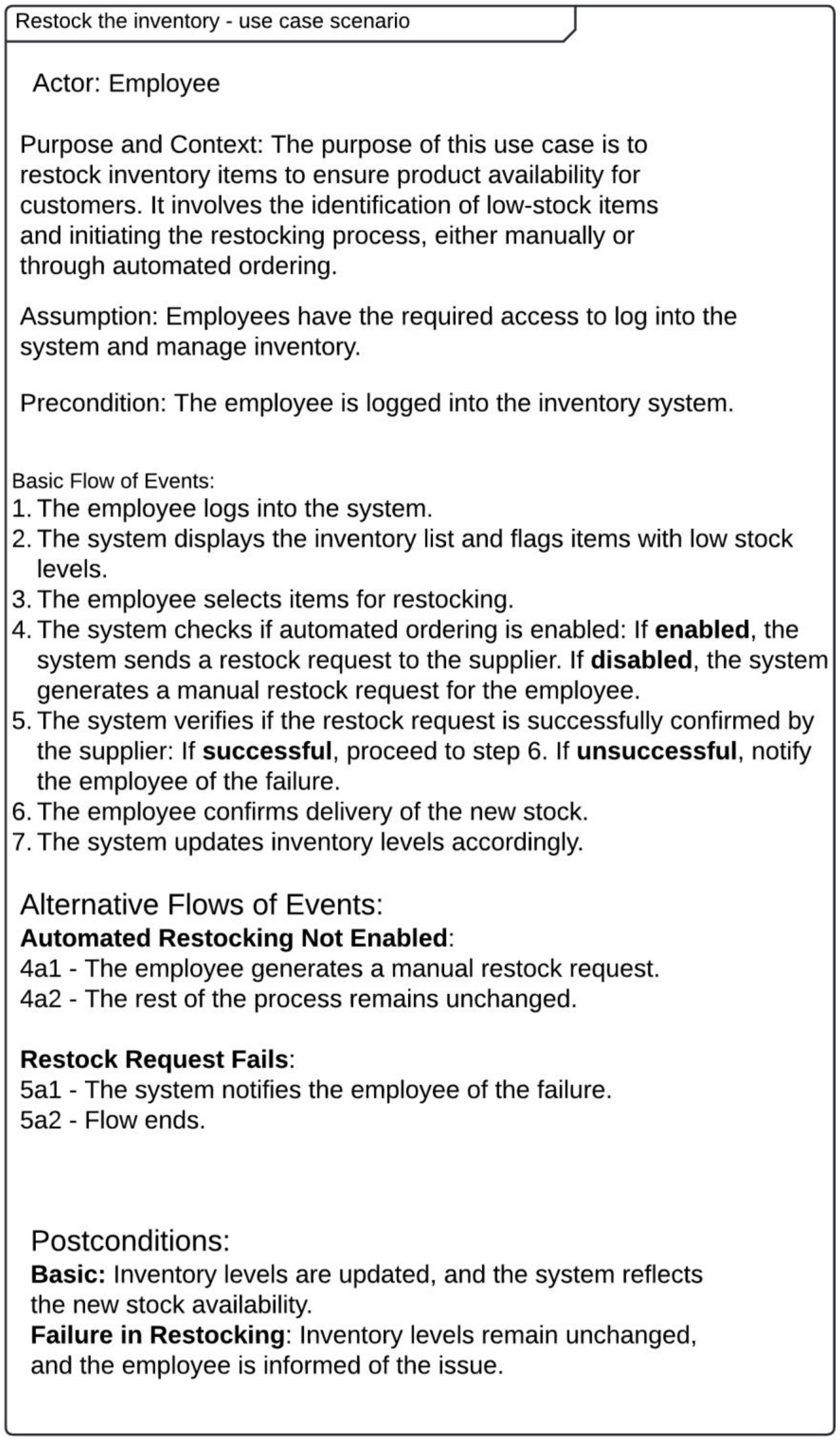
Customer Not Logged In:
2a1 - The system prompts the customer to log in.
2a2 - Once logged in, the process continues from the verification step.

Invalid Membership:
3a1 - The system notifies the customer of the invalid membership details.
3a2 - The customer may retry with valid details or exit the process.

Postconditions:

Basic: Membership benefits are successfully applied to the order.
Invalid Membership: No benefits are applied; the customer remains notified of the issue.

Use Case Scenario: “Restock the inventory”



Restock the inventory - use case scenario

Actor: Employee

Purpose and Context: The purpose of this use case is to restock inventory items to ensure product availability for customers. It involves the identification of low-stock items and initiating the restocking process, either manually or through automated ordering.

Assumption: Employees have the required access to log into the system and manage inventory.

Precondition: The employee is logged into the inventory system.

Basic Flow of Events:

1. The employee logs into the system.
2. The system displays the inventory list and flags items with low stock levels.
3. The employee selects items for restocking.
4. The system checks if automated ordering is enabled: If **enabled**, the system sends a restock request to the supplier. If **disabled**, the system generates a manual restock request for the employee.
5. The system verifies if the restock request is successfully confirmed by the supplier: If **successful**, proceed to step 6. If **unsuccessful**, notify the employee of the failure.
6. The employee confirms delivery of the new stock.
7. The system updates inventory levels accordingly.

Alternative Flows of Events:

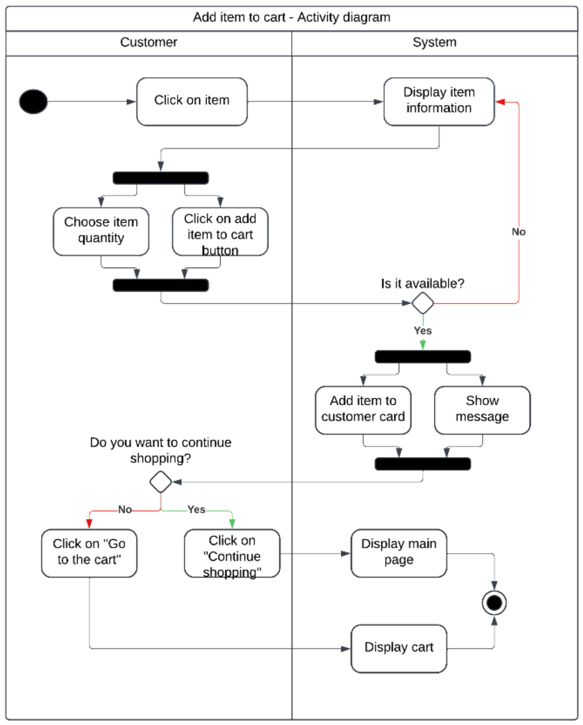
Automated Restocking Not Enabled:
4a1 - The employee generates a manual restock request.
4a2 - The rest of the process remains unchanged.

Restock Request Fails:
5a1 - The system notifies the employee of the failure.
5a2 - Flow ends.

Postconditions:

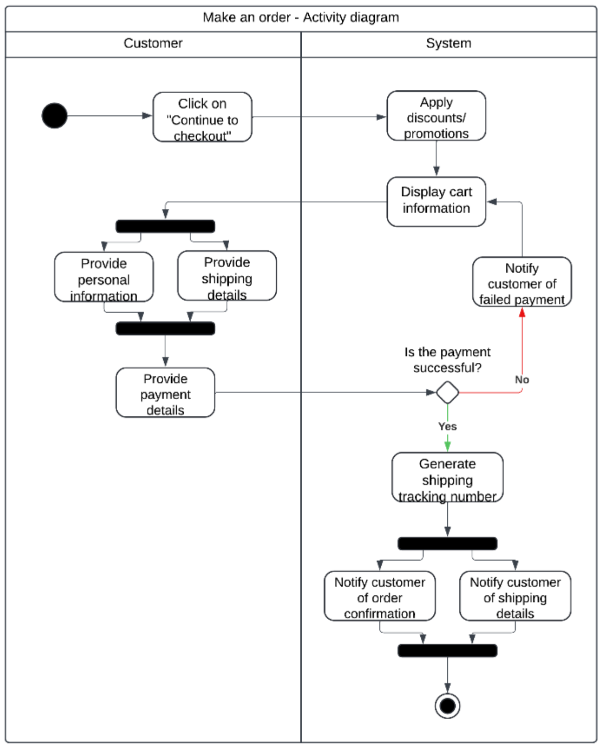
Basic: Inventory levels are updated, and the system reflects the new stock availability.
Failure in Restocking: Inventory levels remain unchanged, and the employee is informed of the issue.

Activity Diagram: "Add item to cart"


Add item to cart - Activity diagram
Customer
System
Click on item
Display item information
Choose item quantity
Click on add item to cart button
Is it available?
No
Yes
Add item to customer card
Show message
Do you want to continue shopping?
No
Yes
Click on "Go to the cart"
Click on "Continue shopping"
Display main page
Display cart

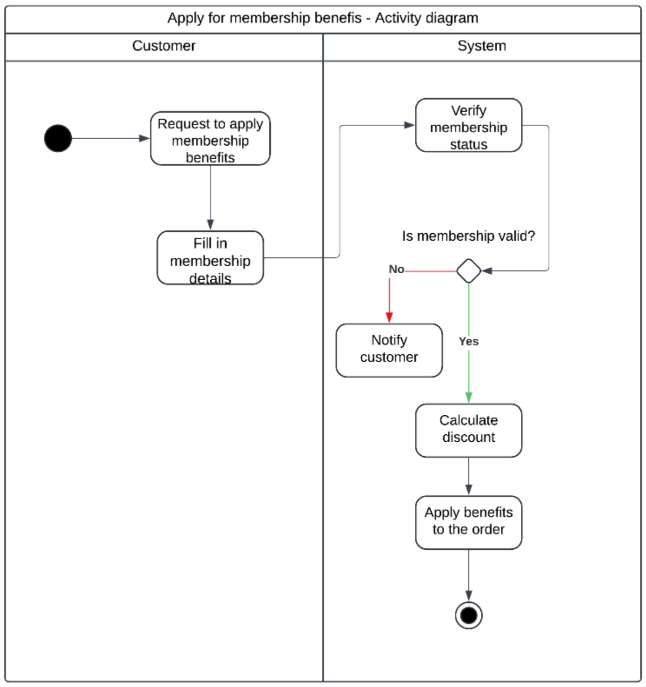
Make an Order
The customer reviews their cart, proceeds to checkout, and enters payment details. The system processes the payment via a gateway, confirms success, and collects shipping details. Upon validation, the order is confirmed, and a notification is sent. If payment fails, the customer is prompted to retry.

Activity Diagram: "Make an order"


Make an order - Activity diagram
Customer
System
Click on "Continue to checkout"
Apply discounts/ promotions
Display cart information
Provide personal information
Provide shipping details
Notify customer of failed payment
Is the payment successful?
No
Provide payment details
Yes
Generate shipping tracking number
Notify customer of order confirmation
Notify customer of shipping details

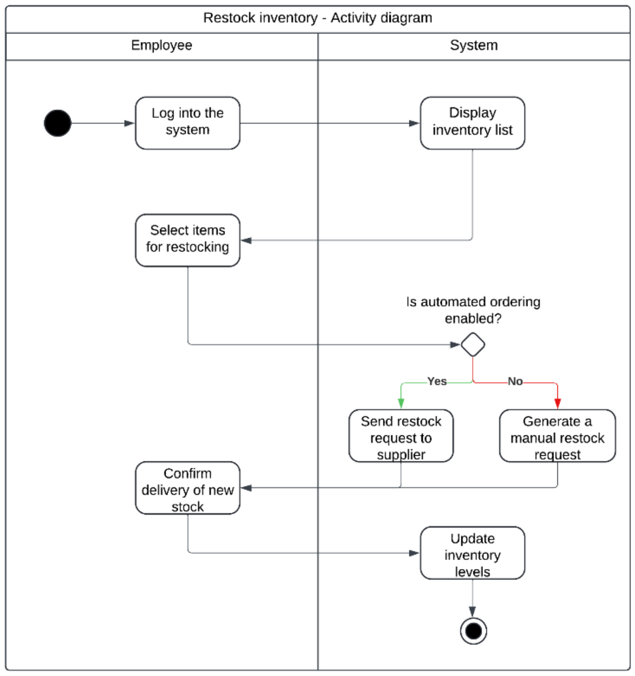
Add Item to Cart
The customer browses products, selects an item, and specifies the quantity. The system checks inventory; if in stock, the item is added to the cart, and the total is updated. If out of stock, the system notifies the customer.

Activity Diagram: "Apply for membership benefits"


Apply for membership benefis - Activity diagram
Customer
System
Request to apply membership benefits
Fill in membership details
Verify membership status
Is membership valid?
No
Yes
Notify customer
Calculate discount
Apply benefits to the order

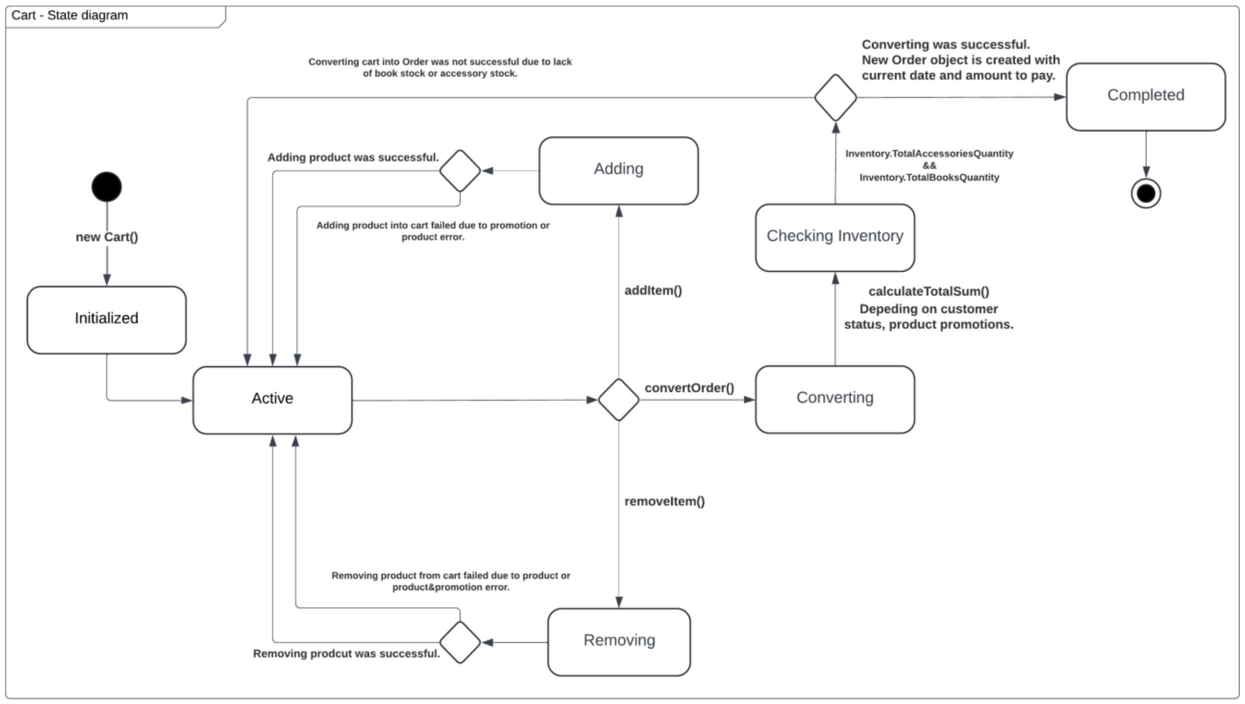
Apply for Membership Benefits
At checkout, the customer requests membership benefits. The system verifies membership status, applies discounts or promotions, and updates the cart. If the membership is invalid, the customer is notified to proceed without benefits.

Activity Diagram: "Restock inventory"


Restock inventory - Activity diagram
Employee
System
Log into the system
Display inventory list
Select items for restocking
Is automated ordering enabled?
Yes
No
Send restock request to supplier
Generate a manual restock request
Confirm delivery of new stock
Update inventory levels

Restock Inventory
Employees view low-stock items flagged by the system, select items to restock, and generate a restock request. The request is sent to suppliers or handled manually. After delivery, inventory levels are updated in the system.

State Diagram


Cart - State diagram
Converting cart into Order was not successful due to lack of book stock or accessory stock.
Converting was successful. New Order object is created with current date and amount to pay.
Completed
Adding product was successful.
Adding
Inventory.TotalAccessoriesQuantity && Inventory.TotalBooksQuantity
new Cart()
Adding product into cart failed due to promotion or product error.
Checking Inventory
Initialized
addItem()
calculateTotalSum() Depeding on customer status, product promotions.
Active
convertOrder()
Converting
removeItem()
Removing product from cart failed due to product or product&promotion error.
Removing
Removing prodcut was successful.

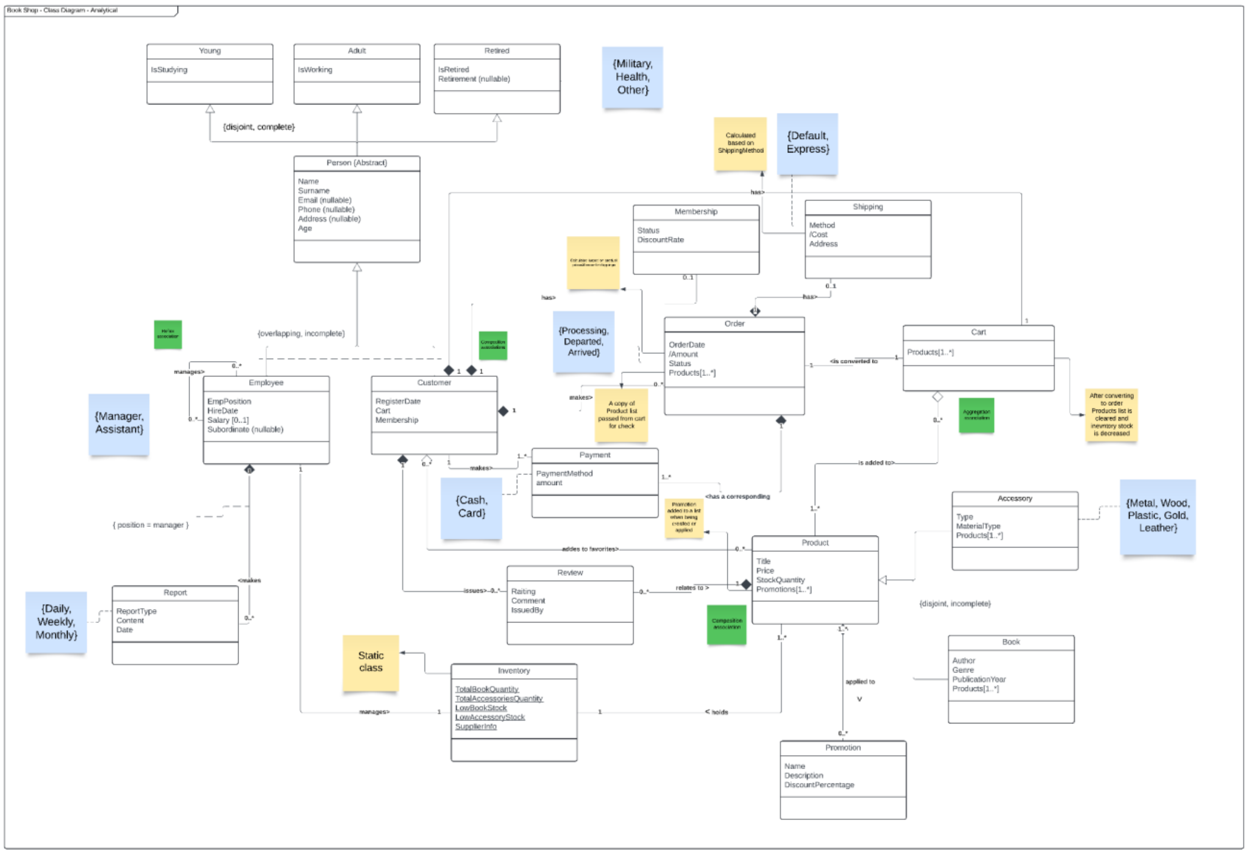
State Diagram - Design Description

The state diagram describes the lifecycle of a **Cart Object**. The cart starts in the "Initialized" state and transitions through different states based on customer actions:

1. ***Active*** - The main operational state of the cart, allowing customers to add or remove products.
2. ***Adding*** - Triggered when a product is being added to the cart.
3. ***Removing*** - Triggered when a product is being removed from the cart.
4. ***Converting*** - The cart is being finalized, and inventory is validated during this process.

If validations (e.g., promotions, stock availability) fail, the cart returns to the **Active** state. The lifecycle ends when the cart is successfully converted or remains active for further modifications.

Analytical Diagram



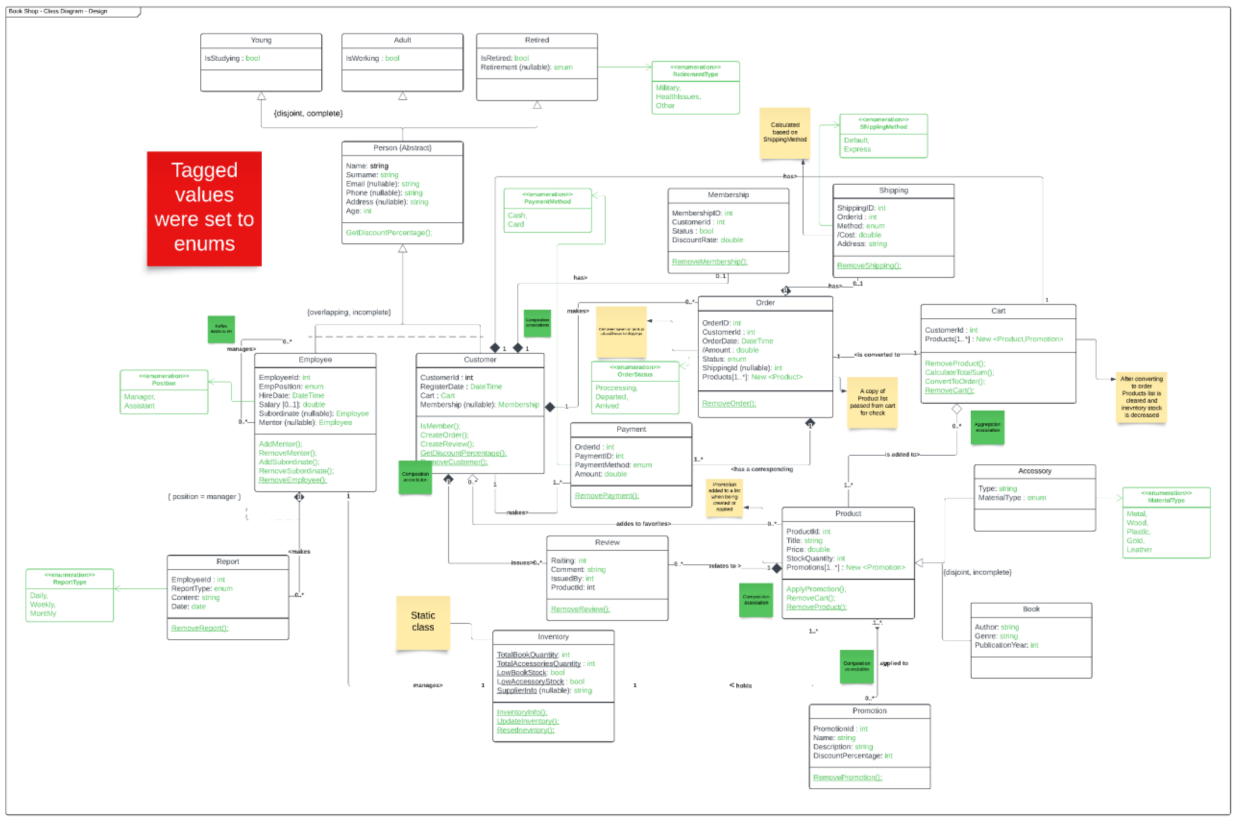
Analytical Diagram - Design Description

The above analytical class diagram visualizes the Bookshop system, where customers interact with products, place orders, and manage their carts. Employees oversee operations like inventory, orders, and reporting. The system organizes data around key entities such as Person, Product, Order, and Cart.

The **Person** class is abstract and is specialized into Employee and Customer, with disjoint and complete constraints ensuring separation. Products are categorized into Book, Accessory, and Promotion, each with unique attributes like Author, Material Type, and Discount Percentage. Customers can add products to their Cart, with each cart directly associating with specific CustomerId and a list of Products. Orders reference products, customers, and optional shipping methods (Default or Express).

This diagram includes UML constructs like tagged values (e.g., Order Status, Material Type) and associations that will be further refined in the design class diagram, focusing on attribute implementations and C# specifics.

Design Diagram



Design Decisions

This section highlights how the design class diagram was implemented in the Bookshop system using C#. It focuses on specific decisions related to attributes, validations, optional and derived attributes, multi-value attributes, and tagged values. Each choice was made to ensure alignment with system requirements and proper functionality.

Attribute Validations

Where possible, the project has implemented attribute-level validation directly within the class properties using C# accessors (get and set). This ensures data integrity during runtime. Notable examples include:

- ***CustomerId***: Validated against existing customers to ensure the ID is present in the system. Invalid IDs throw an ArgumentException.
- ***ShippingId***: Allows nullable values but validates against existing shipping records when provided. Invalid ShippingIds throw an ArgumentException
- ***OrderDate***: Prevents future dates by throwing an ArgumentException for invalid values.
- **Amount**: Prohibits negative values to maintain financial accuracy. Negative values throw an ArgumentOutOfRangeException.

Optional Attributes

Optional attributes in the system are implemented as nullable properties in C#. For instance:

- **Email**: An optional attribute that is nullable, allowing it to be left blank if a person does not provide an email address.
- **Address**: A nullable attribute that can remain unset if no address is specified by the person.

Multi-Value Attributes

The project manages multi-value attributes, such as the list of Product objects in an order, by using collections (List<Product>). This ensures scalability while maintaining associations between orders and their items. The Products attribute captures the relationship between an order and the purchased products.

Derived Attributes

Derived attributes are implemented using C#'s get functionality. For example:

- **Order Total (Amount)**: Automatically calculated from the sum of the product prices in the Products list, accounting for potential discounts or promotions.

Tagged Values

Tagged values are implemented using enumeration classes in C#. For example:

- **Order Status**: States like Pending, Accepted, Shipped, or Cancelled are represented as enums, ensuring clear and manageable state transitions within the system.
- **Material Type**: Categories such as Paper, Plastic, or Metal are represented as enums, providing a clear and structured way to classify materials used in the products.

Reverse Connection Integrity:

- Ensured that all reverse connections between associated classes are implemented and validated correctly. This ensures that when relationships (e.g., Customer and Order) are created or deleted, both sides stay consistent.

Improved Safety:

- Removed unsafe or unnecessary methods across the project to reduce the risk of unintended behavior and ensure that only essential operations are exposed.

Code Organization:

- Refactored constructors to consistently appear under field and property declarations for improved readability and logical grouping within each class.

Encapsulation Enhancements:

- Adjusted accessors and methods to better adhere to encapsulation principles, limiting direct access to sensitive fields and ensuring controlled interaction through clearly defined methods and properties.

Test Updates:

- Updated existing tests to align with the modified accessors and methods. Fixed any inconsistencies in test logic to ensure compatibility with the refactored code.
- Additionally, added reverse connection tests to validate the integrity of relationships between associated classes, ensuring that creation and deletion operations maintain consistency.

COMMENT

The quality of the images is unfortunately low, despite my efforts to fix it. Therefore, I am providing the link to the actual Lucid chart so you can read it more clearly.

https://lucid.app/lucidchart/ce55fd1c-d19e-4cd6-8c6d-a6b2b9acba9c/edit?invitationId=inv_fe923c78-1a0c-46b5-9785-c952572483a2&page=0_0#