# Experiment 1 Class C Application (around 15 classes): Online Book Store (large application, low Detail)

**Description:** An online bookstore application can manage books, users, orders, and reviews.

1. **Background:** The platform aims to provide a centralized space customers to purchase books.
2. **Basic Application Design:**

The application consists of the following classes:

* **Book**: Information about books.
* **Author**: Details of authors.
* **User**: User accounts and profiles.
* **ShoppingCart**: Handling the user's current selections.
* **Order**: Managing orders placed by users.
* **OrderDetail**: Details of individual items in an order.
* **Payment**: Handling payment transactions.
* **Search**: Functionality for searching books.
* **UserAuthentication**: Managing login and registration.
* **DatabaseConnection**: Handling database connections.
* **InventoryManager**: Managing book inventory.

1. **Modular Program Design:** Separate classes for each of the above.
2. **Database Structure:** name= BookStoreC1: Multiple tables as described below:

Here's a breakdown of the potential tables and their columns:

**1. Books Table**

* **BookID** (Primary Key): Unique identifier for each book.
* **Title**: Title of the book.
* **AuthorID**: Foreign key linking to the Authors table.
* **ISBN**: International Standard Book Number.
* **Price**: Selling price of the book.
* **PublishDate**: Date of publication.
* **CategoryID**: Text category name
* **Thumbnail**: Image or link to the book's cover.
* **Description**: A brief description of the book.

**2. Authors Table**

* **AuthorID** (Primary Key): Unique identifier for each author.
* **Name**: Full name of the author.
* **Biography**: Short biography of the author.
* **Photo**: Link to the author's photo or image.

**4. Users Table**

* **UserID** (Primary Key): Unique identifier for each user.
* **Username**: Username for the user's account.
* **PasswordHash**: Hashed password for security.
* **Email**: User's email address.
* **Role**: Role (e.g., customer, admin).
* **DateJoined**: Date when the user account was created.

**5. Orders Table**

* **OrderID** (Primary Key): Unique identifier for each order.
* **UserID**: Foreign key linking to the Users table.
* **OrderDate**: Date when the order was placed.
* **TotalAmount**: Total cost of the order.
* **Status**: Status of the order (e.g., pending, completed).

**6. OrderDetails Table**

* **OrderDetailID** (Primary Key): Unique identifier for each order detail.
* **OrderID**: Foreign key linking to the Orders table.
* **BookID**: Foreign key linking to the Books table.
* **Quantity**: Number of copies ordered.
* **Price**: Price of the book at the time of ordering.

**8. Shopping Cart Table**

* **CartID** (Primary Key): Unique identifier for each cart item.
* **UserID**: Foreign key linking to the Users table.
* **BookID**: Foreign key linking to the Books table.
* **Quantity**: Number of copies in the cart.

1. **GUI Framework:** A responsive design suitable for both desktop and mobile views, with a focus on user experience.

Do you have any reflection questions or need clarifications?

# Class C Application (around 15 classes): Online Book Store (Large Application, Medium detail)

**Description:** An online bookstore application can manage books, users, orders, and reviews.

1. **Background:** The platform aims to provide a centralized space customers to purchase books
2. **Basic Application Design:**

The application consists of the following classes:

* **Book**: Information about books.
* **Author**: Details of authors.
* **User**: User accounts and profiles.
* **ShoppingCart**: Handling the user's current selections.
* **Order**: Managing orders placed by users.
* **OrderDetail**: Details of individual items in an order.
* **Payment**: Handling payment transactions.
* **Search**: Functionality for searching books.
* **UserAuthentication**: Managing login and registration.
* **DatabaseConnection**: Handling database connections.
* **InventoryManager**: Managing book inventory.

1. **Modular Program Design:** Separate modules
2. **Database Structure:** name=bookstoreC2: Multiple tables – MySQL database.

**1. Books Table**

* **BookID** (Primary Key): Unique identifier for each book.
* **Title**: Title of the book.
* **AuthorID**: Foreign key linking to the Authors table.
* **ISBN**: International Standard Book Number.
* **Price**: Selling price of the book.
* **PublishDate**: Date of publication.
* **CategoryID**: Text category name
* **Thumbnail**: Image or link to the book's cover.
* **Description**: A brief description of the book.

**2. Authors Table**

* **AuthorID** (Primary Key): Unique identifier for each author.
* **Name**: Full name of the author.
* **Biography**: Short biography of the author.
* **Photo**: Link to the author's photo or image.

**4. Users Table**

* **UserID** (Primary Key): Unique identifier for each user.
* **Username**: Username for the user's account.
* **PasswordHash**: Hashed password for security.
* **Email**: User's email address.
* **Role**: Role (e.g., customer, admin).
* **DateJoined**: Date when the user account was created.

**5. Orders Table**

* **OrderID** (Primary Key): Unique identifier for each order.
* **UserID**: Foreign key linking to the Users table.
* **OrderDate**: Date when the order was placed.
* **TotalAmount**: Total cost of the order.
* **Status**: Status of the order (e.g., pending, completed).

**6. OrderDetails Table**

* **OrderDetailID** (Primary Key): Unique identifier for each order detail.
* **OrderID**: Foreign key linking to the Orders table.
* **BookID**: Foreign key linking to the Books table.
* **Quantity**: Number of copies ordered.
* **Price**: Price of the book at the time of ordering.

7. **Payment table**

* + - Paymentid
    - OrderID
    - Cardnumber
    - Expiration dates
    - cvv

**8. Shopping Cart Table**

* **CartID** (Primary Key): Unique identifier for each cart item.
* **UserID**: Foreign key linking to the Users table.
* **BookID**: Foreign key linking to the Books table.
* **Quantity**: Number of copies in the cart.

1. **GUI Framework:** A responsive design suitable for both desktop and mobile views, with a focus on user experience.
2. **Error Handling:** Comprehensive error messages for various scenarios like "Not Found", "Content Unavailable", etc.
3. **Authentication:** Robust authentication and authorization mechanisms for Customers
4. **Concurrency:** Designed to handle multiple users accessing the platform simultaneously, with features like real-time content updates.
5. **External Libraries:** N/A.
6. **Deployment:** Initial deployment VMWare local machine to Cloud-based deployment to ensure scalability and accessibility from various locations.

Do you have any reflection questions or need clarifications?

# Class C Application (around 15 classes): Online Book Store (Large application, High detail)

**Description:** An online bookstore application can manage books, users, orders, and reviews.

1. **Background:** The platform aims to provide a centralized space customers to purchase books
2. **Basic Application Design:**

The application consists of the following classes:

* **Book**: Information about books.
* **Author**: Details of authors.
* **User**: User accounts and profiles.
* **ShoppingCart**: Handling the user's current selections.
* **Order**: Managing orders placed by users.
* **OrderDetail**: Details of individual items in an order.
* **Payment**: Handling payment transactions.
* **Search**: Functionality for searching books.
* **UserAuthentication**: Managing login and registration.
* **DatabaseConnection**: Handling database connections.
* **InventoryManager**: Managing book inventory.

1. **Modular Program Design: Separate modules**
2. **Database Structure: Multiple tables – MySQL database.**

**1. Books Table**

* **BookID** (Primary Key): Unique identifier for each book.
* **Title**: Title of the book.
* **AuthorID**: Foreign key linking to the Authors table.
* **ISBN**: International Standard Book Number.
* **Price**: Selling price of the book.
* **PublishDate**: Date of publication.
* **CategoryID**: Text category name
* **Thumbnail**: Image or link to the book's cover.
* **Description**: A brief description of the book.

**2. Authors Table**

* **AuthorID** (Primary Key): Unique identifier for each author.
* **Name**: Full name of the author.
* **Biography**: Short biography of the author.
* **Photo**: Link to the author's photo or image.

**4. Users Table**

* **UserID** (Primary Key): Unique identifier for each user.
* **Username**: Username for the user's account.
* **PasswordHash**: Hashed password for security.
* **Email**: User's email address.
* **Role**: Role (e.g., customer, admin).
* **DateJoined**: Date when the user account was created.

**5. Orders Table**

* **OrderID** (Primary Key): Unique identifier for each order.
* **UserID**: Foreign key linking to the Users table.
* **OrderDate**: Date when the order was placed.
* **TotalAmount**: Total cost of the order.
* **Status**: Status of the order (e.g., pending, completed).

**6. OrderDetails Table**

* **OrderDetailID** (Primary Key): Unique identifier for each order detail.
* **OrderID**: Foreign key linking to the Orders table.
* **BookID**: Foreign key linking to the Books table.
* **Quantity**: Number of copies ordered.
* **Price**: Price of the book at the time of ordering.

7. **Payment table**

* + - Paymentid
    - OrderID
    - Cardnumber
    - Expiration dates
    - cvv

**8. Shopping Cart Table**

* **CartID** (Primary Key): Unique identifier for each cart item.
* **UserID**: Foreign key linking to the Users table.
* **BookID**: Foreign key linking to the Books table.
* **Quantity**: Number of copies in the cart.

1. **GUI Framework: A responsive design suitable for both desktop and mobile views, with a focus on user experience.**
2. **Error Handling: Comprehensive error messages for various scenarios like "Not Found", "Content Unavailable", etc.**
3. **Authentication: Robust authentication and authorization mechanisms for Customers**
4. **Concurrency: Designed to handle multiple users accessing the platform simultaneously, with features like real-time content updates.**
5. **External Libraries: N/A.**
6. **Deployment: Initial deployment VMWare local machine to Cloud-based deployment to ensure scalability and accessibility from various locations.**
7. **User Roles and Permissions:** Student, Faculty, Teacher Assistant.
8. **Data Validation and Constraints:** Validate where appropriate. The date fields should be numeric.
9. **Backup and Recovery:** After the application is developed, back and recovery classes will be developed.
10. **Performance Requirements:** No specific requirements.
11. **Integration with Other Systems:** Not applicable.
12. **User Experience (UX) and User Interface (UI) Design:** Use Metal, but no specific design guidelines, color schemes, or branding elements are provided – use your judgement.
13. **Logging and Auditing:** Not applicable.
14. **Feedback Mechanism:** Not applicable.
15. **Training and Documentation:** Consideration for user training materials or documentation to help faculty and staff get acquainted with the application. Develop after the application is generated.
16. **Scalability and Future Expansion:** Not applicable.

Process Flow for the Book Class

**Detailed Process Flow for the Book Class**

Step 1: Class Properties

* **BookID** (int or String): A unique identifier for the book.
* **Title** (String): The title of the book.
* **AuthorID** (int or String): A reference to the Author's unique identifier.
* **ISBN** (String): The International Standard Book Number.
* **Price** (double): The selling price of the book.
* **PublishDate** (Date or String): The publication date of the book.
* **CategoryID** (String): Category name
* **Thumbnail** (String): A URL or file path to the book's cover image.
* **Description** (String): A brief description or synopsis of the book.

Step 2: Constructor

* **Book Constructor**: Initialize a new instance of the **Book** class with the required properties.

Step 3: Getters and Setters

* **Getters and Setters**: Implement getter and setter methods for each property to encapsulate the class fields. This allows for controlled access and modification of the **Book** properties.

Step 4: Database Interaction Methods

* **Load from Database**: Method to load book details from the database using the **BookID** or **ISBN**.
* **Save to Database**: Method to save the current state of the **Book** object to the database. This could be a new entry or an update to an existing entry.
* **Delete**: Method to remove the book entry from the database.

Step 5: Additional Functionalities

* **Search Related Books**: Method to find books that are similar or related, based on criteria like category, author, or keywords.
* **Format Details for Display**: Method to format the book details for displaying them in the UI, such as creating a summary view.

Step 6: Validation and Error Handling

* **Validate Data**: Ensure that all the required fields are present and in the correct format before database operations.
* **Handle Errors**: Catch and handle potential errors during database interactions or data processing.

Step 7: Additional Features

* **Inventory Management**: If the application manages inventory, methods related to stock management could be included.

**Summary**

The **Book** class serves as a data model representing books in the system. It encapsulates book properties, handles database interactions, and provides additional functionalities like searching and data formatting. This class should be designed with clean and maintainable code practices, ensuring it can be easily integrated with other parts of the application like the user interface and database management system.

Detailed Process Flow for the Author Class

Step 1: Class Attributes

* **AuthorID** (int or String): A unique identifier for the author.
* **Name** (String): The full name of the author.
* **Biography** (String): A brief biography or background information about the author.
* **Photo** (String): A URL or path to the author's photo.

Step 2: Constructor

* **Author Constructor**: Initialize a new instance of the **Author** class with the necessary properties. This could include **AuthorID**, **Name**, and optionally **Biography** and **Photo**.

Step 3: Getters and Setters

* **Getters and Setters**: Implement getter and setter methods for each attribute. These methods provide controlled access to the class's properties, allowing for data encapsulation.

Step 4: Database Interaction Methods

* **Load from Database**: Implement a method to load author details from the database using **AuthorID**.
* **Save to Database**: A method to save or update the author's details in the database. This method handles both creating a new author record and updating an existing one.
* **Delete**: A method to delete the author's record from the database.

Step 5: Associated Books

* **List Books**: A method to retrieve a list of books written by the author. This could involve a database query joining the **Author** table with the **Books** table based on **AuthorID**.

Step 6: Validation and Error Handling

* **Validate Data**: Prior to database operations, validate the data to ensure that mandatory fields are present and correctly formatted.
* **Error Handling**: Implement error handling for database interactions and data processing, catching exceptions and providing useful feedback.

Step 7: Additional Functionalities (Optional)

* **Search Authors**: A method to search for authors based on criteria like name, genre, or other attributes.
* **Author's Awards and Achievements**: If relevant, methods to handle the author's awards and other achievements could be included.

Step 8: Format for Display

* **Format for UI**: Methods to format author details for display in the user interface, like creating a concise author bio view or formatting the author's name and photo for listings.

**Summary**

The **Author** class is responsible for managing author-related data. It encapsulates author attributes, provides methods for database interactions, and offers functionalities to retrieve and display author-related information. This class should be designed to be robust, ensuring seamless integration with other components like the **Book** class and the user interface.

Detailed Process Flow for the User Class

The **User** class in an online bookstore application is key for managing user accounts, profiles, and their interactions within the system. Here's a detailed process flow for the **User** class:

Step 1: Class Attributes

* **UserID** (int or String): A unique identifier for each user.
* **Username** (String): The user's chosen username.
* **PasswordHash** (String): Hashed password for secure authentication.
* **Email** (String): Email address of the user.
* **Role** (String or Enum): Role of the user (e.g., customer, admin).
* **DateJoined** (Date or String): The date the user joined the platform.
* **ProfilePicture** (String): URL or path to the user's profile picture.

Step 2: Constructor

* **User Constructor**: Initialize a new instance of the **User** class with necessary attributes. This typically includes **UserID**, **Username**, **Email**, and other relevant details.

Step 3: Getters and Setters

* **Getters and Setters**: Implement getter and setter methods for each property. These methods provide controlled access to the class's properties, allowing for data encapsulation and validation.

Step 4: Database Interaction Methods

* **Load from Database**: Method to load user details from the database using **UserID** or **Username**.
* **Save to Database**: Method to save or update the user's details in the database. It should handle both new user registration and updating existing user profiles.
* **Delete**: Method to remove the user's data from the database, typically used for account deactivation or removal.

Step 5: Authentication and Authorization

* **Login**: A method for validating user credentials during the login process.
* **Change Password**: Method to allow users to change their password, including necessary security checks.
* **Verify Permissions**: For users with different roles, methods to verify if a user has the required permissions to perform certain actions.

Step 6: User Profile Management

* Registration: Methods to register a user.
* **Update Profile**: Methods to update various aspects of the user's profile, such as email, profile picture, and other personal details.
* **View Profile**: Functionality to retrieve and display the user's profile information.

Step 7: Validation and Error Handling

* **Validate Data**: Ensure all user data is valid and complete before database operations, especially for sensitive information like email and passwords.
* **Handle Errors**: Catch and handle potential errors during database interactions or data processing.

Step 8: Additional Functionalities (Optional)

* **User Activity Log**: Methods to track user activities, such as logins, purchases, or interactions with the platform.
* **Password Recovery**: Functionality for users to recover or reset their password.
* **User Preferences**: Manage user-specific settings or preferences within the platform.

**Summary**

The **User** class is a central component for user management, handling everything from user authentication to profile management. It should be designed with security and privacy in mind, especially concerning user credentials and personal information.

Detailed Process Flow for the ShoppingCart Class

The **ShoppingCart** class in an online bookstore application plays a crucial role in managing the user's current selections, including adding, removing, and updating items in the cart. Here's a detailed process flow for the **ShoppingCart** class:

Step 1: Class Attributes

* **CartID** (int or String): A unique identifier for the shopping cart.
* **UserID** (int or String): The identifier of the user to whom the cart belongs.
* **Items** (List of ShoppingCartItem): A list of items (books) in the cart, with quantity and other relevant details.

Step 2: Inner Class for Cart Items

* **ShoppingCartItem**: An inner class or a separate class to represent individual items in the shopping cart. It includes **BookID**, **Quantity**, **Price**, etc.

Step 3: Constructor

* **ShoppingCart Constructor**: Initialize a new shopping cart with an empty list of items and associated with a specific user.

Step 4: Cart Operations

* **Add Item**: Method to add a new item (book) to the cart. It includes checking if the item is already in the cart and updating the quantity.
* **Remove Item**: Method to remove an item from the cart.
* **Update Item Quantity**: Method to update the quantity of an existing item in the cart.
* **Get Total**: Method to calculate the total cost of the items in the cart.
* **Clear Cart**: Method to remove all items from the cart.

Step 5: Persistence and Retrieval

* **Save Cart State**: Method to save the current state of the cart to a database or session, depending on the application's architecture.
* **Load Cart State**: Method to load the cart's state from the database or session for a returning user.

Step 6: Validation and Error Handling

* **Validate Quantity**: Ensure the quantity updates are within acceptable limits (e.g., not below zero, not exceeding available stock).
* **Handle Errors**: Catch and handle errors during cart operations, like database failures or invalid operations.

Step 7: Checkout Process Integration

* **Prepare for Checkout**: Method to prepare the cart's contents for the checkout process, including verifying stock and calculating totals.
* **Integration with Order Processing**: Ensure smooth transition of cart items to the order processing system once the user proceeds to checkout.

Step 8: Additional Functionalities (Optional)

* **N/A**

**Summary**

The **ShoppingCart** class is a dynamic component designed to enhance the user's shopping experience by effectively managing their selections. It interacts closely with the book inventory, user sessions, and the checkout process. The class should be optimized for performance, as it handles frequent updates and queries.

The **Order** class in an online bookstore application is pivotal for managing orders placed by users, encompassing aspects like order creation, tracking, and status updates. Here's a detailed process flow for the **Order** class:

Detailed Process Flow for the Order Class

Step 1: Class Attributes

* **OrderID** (int or String): A unique identifier for each order.
* **UserID** (int or String): The identifier of the user who placed the order.
* **OrderDetails** (List of OrderDetail): A list representing the details of each item in the order.
* **OrderDate** (Date or String): The date when the order was placed.
* **TotalAmount** (double): The total cost of the order.
* **Status** (String or Enum): The current status of the order (e.g., pending, shipped, delivered).

Step 2: Inner Class for Order Details

* **OrderDetail**: An inner class or a separate class representing individual items in the order, including **BookID**, **Quantity**, **Price**, etc.

Step 3: Constructor

* **Order Constructor**: Initialize a new order with necessary details, possibly including the user ID and an initial list of order items.

Step 4: Order Management Operations

* **Add Item to Order**: Method to add a new item to the order, typically called when creating the order from the shopping cart.
* **Remove Item from Order**: Method to remove an item from the order, if needed.
* **Update Order Details**: Method to update details of the order, like quantities of items.
* **Calculate Total**: Method to calculate the total amount of the order based on individual item prices and quantities.

Step 5: Order Status Handling

* **Set Status**: Method to update the status of the order (e.g., from pending to shipped).
* **Track Order**: Method to retrieve the current status and potentially detailed tracking information of the order.

Step 6: Persistence and Retrieval

* **Save Order**: Method to save the order details to the database when the order is placed or updated.
* **Load Order**: Method to load an existing order from the database, typically using the **OrderID**.

Step 7: Validation and Error Handling

* **Validate Order**: Ensure that the order details are valid before saving, such as verifying stock availability for ordered items.
* **Handle Errors**: Catch and handle potential errors during order processing, such as database operation failures.

Step 8: Integration with Other Components

* **Link with Payment Processing**: Integrate with a payment processing system to handle payments for the order.
* **Link with User Accounts**: Associate orders with user accounts for order history tracking and user-specific order management.

Step 9: Additional Functionalities (Optional)

* **Generate Invoice**: Method to generate an invoice or receipt for the order.
* **Cancel Order**: Functionality to allow users to cancel the order, if the system permits.

**Summary**

The **Order** class is essential for handling the lifecycle of orders within the online bookstore. It must interact seamlessly with other components like user accounts, payment systems, and inventory management to ensure a smooth and reliable ordering process.

Detailed Process Flow for the OrderDetail Class

The **OrderDetail** class in an online bookstore application is crucial for capturing the details of individual items within an order. This class works in conjunction with the **Order** class to provide comprehensive information about each purchased item. Here's a detailed process flow for the **OrderDetail** class:

Step 1: Class Attributes

* **OrderDetailID** (int or String): A unique identifier for each order detail entry.
* **OrderID** (int or String): The identifier of the order to which this detail belongs.
* **BookID** (int or String): The identifier of the book being ordered.
* **Quantity** (int): The number of copies of the book ordered.
* **Price** (double): The price of the book at the time of ordering.
* **Subtotal** (double): The total cost for this line item (Price \* Quantity).

Step 2: Constructor

* **OrderDetail Constructor**: Initialize a new instance of the **OrderDetail** class with necessary attributes such as **OrderID**, **BookID**, **Quantity**, and **Price**.

Step 3: Computation Methods

* **Calculate Subtotal**: Method to compute the subtotal for the order detail. This is typically the product of the **Quantity** and the **Price** of the book.

Step 4: Database Interaction Methods

* **Save to Database**: Method to save this order detail to the database. This usually happens as part of saving the overall order.
* **Load from Database**: Method to load order detail information from the database, typically using the **OrderDetailID** or as part of an order retrieval using **OrderID**.

Step 5: Validation and Error Handling

* **Validate Data**: Ensure that the data (e.g., quantity, price) is valid and consistent before saving to the database.
* **Handle Errors**: Catch and handle potential errors during database interactions or data processing.

Step 6: Integration with Order and Book Classes

* **Link with Order**: Associate each **OrderDetail** with an **Order** object, ensuring that the details correctly reflect the items within an order.
* **Retrieve Book Details**: Provide a method to retrieve detailed information about the book (referenced by **BookID**), possibly integrating with the **Book** class or database to fetch this data.

Step 7: Additional Functionalities (Optional)

* **Update Order Detail**: If the system allows, provide methods to update the details of an order item, such as changing the quantity.
* **Display Information**: Method to format the order detail information for displaying it in the user interface, like in an order summary view.

**Summary**

The **OrderDetail** class serves as a detailed record of each item within an order, including quantities, pricing, and calculated subtotals.

Detailed Process Flow for the Payment Class

**Payment**: Handling payment transactions.

The **Payment** class in an online bookstore application is vital for managing payment transactions associated with user orders. This class handles the processing of payments, recording of transaction details, and communication with any external payment services if necessary. Here's a detailed process flow for the **Payment** class:

Step 1: Class Attributes

* **PaymentID** (int or String): A unique identifier for each payment transaction.
* **OrderID** (int or String): The identifier of the order this payment is for.
* **UserID** (int or String): The user who is making the payment.
* **Amount** (double): The total amount being paid.
* **PaymentMethod** (String or Enum): The method of payment (e.g., credit card, PayPal, etc.).
* **PaymentDate** (Date or String): The date when the payment was made.
* **Status** (String or Enum): The status of the payment (e.g., processed, failed, pending).
* **Credit card number:**
* **CVV**
* **Expiration date**

Step 2: Constructor

* **Payment Constructor**: Initialize a new payment instance with necessary details, typically including the **OrderID**, **UserID**, **Amount**, and **PaymentMethod**.

Step 3: Payment Processing Methods

* **Process Payment**: Method to process the payment. This might involve:
  + Validating payment details.
  + Interacting with external payment gateways if necessary.
  + Updating the payment status based on the transaction result.

Step 4: Database Interaction Methods

* **Save Payment Details**: Method to save the payment transaction details to the database, including the payment amount, method, and status.
* **Load Payment Details**: Method to retrieve payment details from the database, typically using the **PaymentID**.

Step 5: Validation and Error Handling

* **Validate Payment Information**: Ensure all payment information is valid, such as checking the validity of payment method details.
* **Handle Payment Errors**: Catch and handle errors during the payment process, including communication errors with external payment services.

Step 6: Receipt Generation

* **Generate Receipt**: Method to generate a payment receipt, which could be sent to the user via email or made available for download.

Step 7: Integration with Order Management

* **Link with Order**: Associate the payment with the corresponding order, ensuring that the order status is updated based on the payment status (e.g., marking an order as paid).

Step 8: Refund Handling (Optional)

* **N/A**

**Summary**

The **Payment** class is a critical component for managing financial transactions in the online bookstore. It ensures secure and accurate processing of payments and integrates closely with order management to provide a seamless shopping experience.

Detailed Process Flow for the Search Class

The **Search** class in an online bookstore application is fundamental for providing users with the capability to search for books based on various criteria like title, author, ISBN, category, etc. Here's a detailed process flow for the **Search** class:

Step 1: Define Search Criteria

* **Search Parameters**: Define the parameters that can be used for searching, such as **title**, **author**, **ISBN**, **category**, and potentially others like **keywords** or **price range**.

Step 2: Constructor and Initialization

* **Search Constructor**: Initialize the **Search** class, possibly with default search parameters or settings.

Step 3: Search Method

* **Perform Search**: Implement a method that takes the search parameters and executes the search. This could involve:
  + Formulating a query to search the database based on the provided criteria.
  + Handling different types of searches, like a broad search across all fields or a specific search within a category.

Step 4: Database Interaction

* **Database Query**: Connect to the database and execute the search query. Ensure efficient querying to optimize performance, especially for broad searches.

Step 5: Process Search Results

* **Handle Results**: Process the results returned from the database. This might involve:
  + Sorting the results based on relevance, popularity, or other criteria.
  + Paginating results for better usability in the user interface.
  + Mapping database records to **Book** objects or similar data structures for display.

Step 6: Validation and Error Handling

* **Validate Search Inputs**: Before executing the search, validate the input parameters to ensure they are in a proper format and within acceptable bounds.
* **Handle Search Errors**: Catch and handle potential errors during the search process, like database connection issues.

Step 7: Integration with User Interface

* **Display Results**: Ensure the search results can be easily integrated into the user interface, providing users with a list of books that match their search criteria.
* **Search Feedback**: Provide feedback to the user, especially in cases where no results are found.

Step 8: Additional Functionalities (Optional)

* **N/A**

**Summary**

The **Search** class is a critical component in enhancing the user experience by enabling efficient and effective search functionality. It bridges user queries with the database, returning relevant search results.

Detailed Process Flow for the UserAuthentication Class

The **UserAuthentication** class in an online bookstore application is vital for managing user login and registration processes. This class handles user credentials, ensures secure access, and maintains user session information. Here's a detailed process flow for the **UserAuthentication** class:

Step 1: Class Attributes

* **Database Connection**: Attributes to hold database connection information for accessing user credentials.
* **Current User**: Attribute to keep track of the currently logged-in user (if any).

Step 2: Constructor

* **UserAuthentication Constructor**: Initialize the class, possibly setting up the database connection.

Step 3: User Registration

* **Register User**: Implement a method for user registration. This involves:
  + Collecting user details like username, password, email, etc.
  + Hashing the password for secure storage.
  + Inserting the new user's details into the database.
  + Handling any errors or exceptions, such as username conflicts.

Step 4: User Login

* **Login User**: Implement a method for user login. This method should:
  + Accept username and password as inputs.
  + Retrieve the corresponding user record from the database.
  + Compare the provided password with the stored hashed password.
  + Establish a user session if the credentials are correct.
  + Provide appropriate feedback for successful or unsuccessful login attempts.

Step 5: Password Hashing

* **Hash Password**: Utilize a secure method to hash passwords before storing them in the database and when comparing them during login. For this experiment just store the plain text.

Step 6: Database Interaction

* **Database Queries**: Implement methods to interact with the database for retrieving and storing user information.

Step 7: Validation and Error Handling

* **Validate User Inputs**: Validate inputs during registration and login, such as checking the format of the email and the strength of the password.
* **Handle Authentication Errors**: Handle potential errors during login and registration, such as database connection issues or invalid credentials.

Step 8: Session Management

* **Manage User Session**: Implement functionalities to create, maintain, and terminate user sessions upon successful login and logout.

Step 9: Logout User

* **Logout**: Provide a method to allow users to log out, which would terminate the user session.

Step 10: Additional Functionalities (Optional)

* **N/A**

**Summary**

The **UserAuthentication** class is a cornerstone for ensuring secure access to the online bookstore application. It manages user credentials and sessions, providing functionalities for registration, login, and logout, along with necessary security measures.

Detailed Process Flow for the DatabaseConnection Class

The **DatabaseConnection** class in an online bookstore application is responsible for managing the connections to the database. This class acts as a central point for all database interactions, ensuring efficient and reliable access to the database. Here's a detailed process flow for the **DatabaseConnection** class:

Step 1: Define Connection Parameters

* **Database URL**: The URL for connecting to the database, which includes the database type, host, port, and database name.
* **Username and Password**: Credentials for accessing the database.
* **Driver**: The JDBC driver or equivalent database driver needed for connecting to the database.

Step 2: Class Attributes

* **Connection Object**: An attribute to hold the database connection instance.
* **Instance**: For singleton implementation, an attribute to hold the instance of the **DatabaseConnection** class.

Step 3: Constructor and Initialization

* **Private Constructor**: In the case of a singleton pattern, the constructor is made private to control instance creation.
* **Initialize Connection**: A method or part of the constructor that initializes the database connection using the provided parameters.

Step 4: Singleton Implementation (Optional)

* **Get Instance**: If implementing a singleton pattern, a static method to get the instance of the **DatabaseConnection** class, ensuring that only one instance of the connection exists.

Step 5: Open Connection

* **Open Database Connection**: A method to establish a connection to the database. This involves:
  + Loading the database driver.
  + Connecting to the database using the JDBC URL and credentials.
  + Handling any exceptions or errors during connection.

Step 6: Close Connection

* **Close Database Connection**: A method to close the database connection when it is no longer needed, ensuring that resources are properly released.

Step 7: Execute Queries

* **Execute Query**: Methods to execute SQL queries, including:
  + Executing SELECT, INSERT, UPDATE, and DELETE queries.
  + Returning results for SELECT queries.
  + Handling exceptions during query execution.

Step 8: Transaction Management

* **Begin Transaction**: Method to begin a database transaction.
* **Commit Transaction**: Method to commit the transaction.
* **Rollback Transaction**: Method to rollback the transaction in case of errors.

Step 9: Error Handling and Logging

* **Handle Errors**: Implement error handling for database operations, including logging errors and providing appropriate feedback.
* **Logging**: Incorporate logging for key events and errors for troubleshooting and monitoring.

Step 10: Utility Methods (Optional)

* **Utility Methods**: Include utility methods for common database operations, like checking the connection status, resetting the connection, etc.

**Summary**

The **DatabaseConnection** class serves as the backbone for all database-related activities in the online bookstore application, providing a centralized and efficient way to manage database interactions. Whether using a singleton pattern or standard class instantiation, it ensures that the application can reliably interact with the database for various operations.

Detailed Process Flow for the InventoryManager Class

The **InventoryManager** class in an online bookstore application is essential for managing the inventory of books. This class tracks the stock levels of each book, handles updates to the inventory, and provides information necessary for order processing and stock management. Here's a detailed process flow for the **InventoryManager** class:

Step 1: Class Attributes

* **Database Connection**: An instance or reference to the **DatabaseConnection** class to interact with the database.
* **Inventory Data**: Data structure to hold the current inventory status of books (e.g., a map of **BookID** to stock quantity).

Step 2: Constructor and Initialization

* **InventoryManager Constructor**: Initialize the class, possibly setting up the initial inventory data by fetching it from the database.

Step 3: Fetch Inventory from Database

* **Load Inventory**: Implement a method to load the complete inventory data from the database. This method populates the inventory data structure with current stock levels for each book.

Step 4: Inventory Update Methods

* **Update Stock**: Method to update the stock level for a specific book. This could be triggered by various events, such as a new shipment arrival or stock adjustment.
* **Reduce Stock**: Method to reduce the stock level, typically used when an order is placed.
* **Increase Stock**: Method to increase the stock level, typically used when new stock is added or an order is canceled.

Step 5: Inventory Check Methods

* **Check Stock**: Method to check the stock level for a specific book, often used during the order processing to ensure that a book is in stock.
* **IsBookAvailable**: A quick check method to determine if a book is available (stock > 0).

Step 6: Database Interaction for Stock Updates

* **Save Stock Updates**: After any stock update, save the new stock levels to the database to ensure data consistency.

Step 7: Reporting and Alerts

* **Generate Inventory Report**: Method to generate reports on current stock levels, which can be used for inventory management and planning.
* **Low Stock Alert**: Implement a feature to trigger alerts or notifications when the stock level of a book falls below a certain threshold.

Step 8: Validation and Error Handling

* **Validate Inventory Changes**: Ensure that any changes to the inventory (increase or decrease) are valid and do not result in negative stock levels.
* **Handle Inventory Errors**: Catch and handle potential errors during inventory operations, such as database failures or invalid stock adjustments.

Step 9: Integration with Other Components

* **Order Processing Integration**: Work closely with the **Order** class to reflect inventory changes based on order placements, cancellations, and returns.
* **Book Information Integration**: Retrieve detailed book information (if needed) from the **Book** class or database for inventory reporting.

**Summary**

The **InventoryManager** class is a key component in managing the bookstore's inventory, ensuring accurate tracking of stock levels and facilitating effective inventory control. It plays a critical role in order processing, stock management, and strategic planning for inventory replenishment.

Do you have any reflection questions or need clarifications?