Experiment 1

Familiarization with Rational Rose or Umbrello

Overview of Rational Rose and Umbrello

Rational Rose and Umbrello are tools for modeling software systems using the Unified Modeling Language (UML), essential for visualizing and designing software architecture.

Rational Rose

- **Purpose:** Developed by Rational Corporation (now IBM), Rational Rose is an object-oriented modeling tool that supports the entire software development lifecycle, allowing users to create UML diagrams for system architecture and design.
- **Features:**
    - Supports round-trip engineering for model-code synchronization.
    - Facilitates various UML diagrams, including class, sequence, and use case diagrams.
    - Offers a user-friendly interface for managing complex projects.

Umbrello

- **Purpose:** Umbrello is an open-source UML modeling tool within the KDE software compilation, aimed at creating UML diagrams for software design and documentation.
- **Features:**
    - Supports a wide range of UML diagrams similar to Rational Rose.
    - Provides a flexible, intuitive interface for users of all experience levels.
    - Being open-source, it allows for customization and community contributions.

Comparison

- **Licensing:** Rational Rose is a commercial product, whereas Umbrello is free and open-source, making it more accessible for individuals and small teams.
- **User Base:** Rational Rose is favored by larger enterprises due to its comprehensive features, while Umbrello is popular among individual developers and smaller projects.

Conclusion

Both Rational Rose and Umbrello are effective UML modeling tools, each with unique strengths. Rational Rose suits enterprise-level applications, while Umbrello offers a cost-effective solution for smaller projects or individual developers. Familiarity with either tool can enhance your software design capabilities

Case Tools Lab-2 Experiment

Case Study: Online Book Store

In this analysis, we'll explore events and interactions within a customer support system for an online bookstore using UML diagrams.

Key Events in the Customer Support System:
1. User Registration and Login: Users create an account or log in.
2. Book Search and Inquiry: Customers search for books or inquire about details.
3. Order Placement: Customers place an order.
4. Order Tracking: Customers track order status.
5. Customer Support Interaction: Customers contact support for assistance.
6. Feedback Submission: Customers submit feedback or reviews.

UML Diagrams for the Customer Support System:
1. Use Case Diagram: Shows interactions between users and the system.
2. Class Diagram: Outlines system structure including classes like Customer, Order, Book, and SupportTicket.
3. Sequence Diagram: Details sequence of events during interactions, like placing an order.
4. Activity Diagram: Represents workflow of customer support interactions.
5. Deployment Diagram: Illustrates physical deployment of system components.

Conclusion:
Using UML diagrams helps in understanding user interactions, system functionalities, and overall application architecture, enhancing design and implementation.

B) Use Cases for the Customer Support System of an Online Book Store

Primary Use Cases:
1. User Registration: Customers create an account.
2. User Login: Customers log into their account.
3. Book Search: Customers search for books.
4. View Book Details: Customers view detailed book info.
5. Place Order: Customers place orders.
6. Track Order: Customers check order status.
7. Contact Customer Support: Customers contact support.
8. Submit Feedback: Customers submit feedback or reviews.
9. Manage Account: Customers update account info.
10. View FAQs: Customers access FAQs.

Conclusion:
These use cases provide an overview of interactions within the customer support system, guiding design and development to enhance user experience and support efficiency.

Event Table for the Customer Support System of an Online Book Store

An event table organizes the events, actors, and system responses for the identified use cases in the customer support system. Below is the event table for the online bookstore case study:

1	User Registration	Customer	None	Customer account is created	System stores user information and sends confirmation.
2	User Login	Customer	Customer has a registered account	Customer is logged in	System verifies credentials and grants access.
3	Book Search	Customer	Customer is logged in or browsing as guest	List of books matching criteria is displayed	System retrieves and displays search results.
4	View Book Details	Customer	Customer has performed a book search	Customer views book details	System displays detailed information about the selected book.
5	Place Order	Customer	Customer has selected books	Order is processed	System confirms order and sends order details to the customer.
6	Track Order	Customer	Customer has placed an order	Customer views current order status	System retrieves and displays order status.
7	Contact Customer Support	Customer	Customer has an issue or inquiry	Customer receives assistance	System logs the inquiry and notifies a support agent.
8	Submit Feedback	Customer	Customer has completed a purchase	Feedback is recorded	System stores feedback and may notify relevant departments.

Conclusion

This event table provides a structured overview of the interactions within the customer support system for the online bookstore. It outlines the events, actors involved, preconditions, postconditions, and the system's responses, facilitating a clear understanding of how the system operates and responds to user actions.

Domain Classes for the Customer Support System of an Online Book Store

1. Customer

- Attributes: customerId, name, email, password, address, phoneNumber.
- Relationships: Has many Orders, can submit many Feedbacks, can create many SupportTickets.

2. Order

- Attributes: orderId, orderDate, status, totalAmount.
- Relationships: Belongs to one Customer, contains many OrderItems, can have one Payment.

3. Book

- Attributes: bookId, title, author, genre, price, stockQuantity.

- Relationships: Can be part of many OrderItems.

4. OrderItem

- Attributes: orderItemId, quantity, subtotal.
- Relationships: Belongs to one Order, references one Book.

5. Payment

- Attributes: paymentId, paymentDate, amount, paymentMethod.
- Relationships: Belongs to one Order.

6. SupportTicket

- Attributes: ticketId, issueDescription, status, creationDate.
- Relationships: Belongs to one Customer, can be handled by one SupportAgent.

7. SupportAgent

- Attributes: agentId, name, email.
- Relationships: Can handle many SupportTickets.

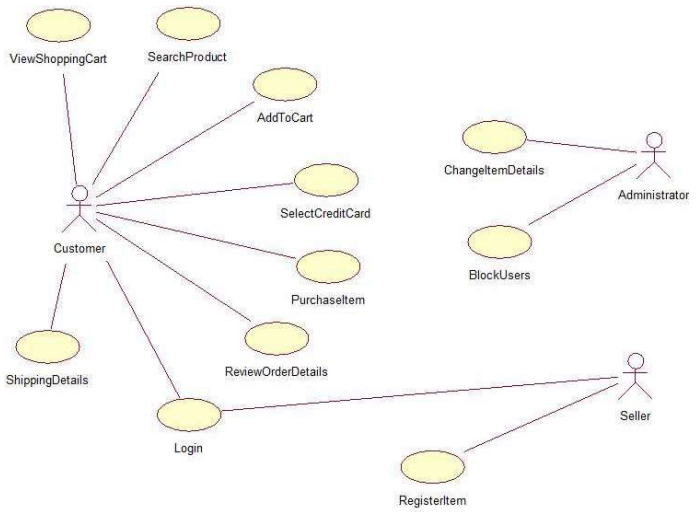
8. Feedback

- Attributes: feedbackId, rating, comments, submissionDate.
- Relationships: Belongs to one Customer, references one Book.

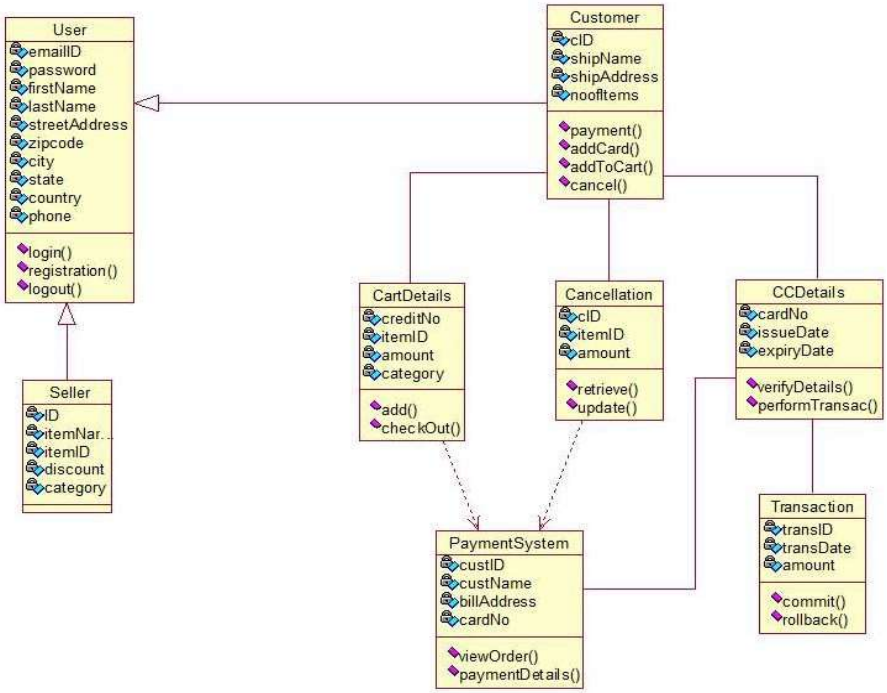
Conclusion

These domain classes encapsulate essential entities and their relationships within the customer support system for an online bookstore. They help understand the system's structure, facilitate database design, and guide application development to ensure a smooth user experience.

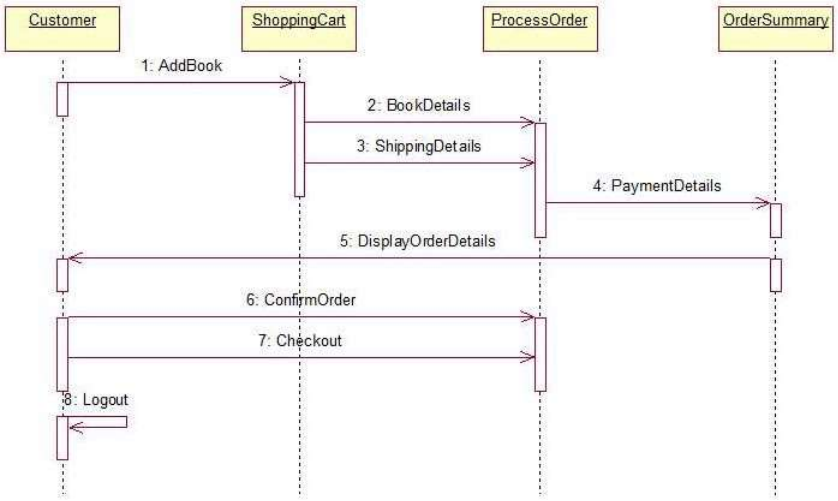
EVENT DIAGRAM



USE CASE DIAGRAM



SEQUENCE DIAGRAM


Customer
ShoppingCart
ProcessOrder
OrderSummary
1: AddBook
2: BookDetails
3: ShippingDetails
4: PaymentDetails
5: DisplayOrderDetails
6: ConfirmOrder
7: Checkout
8: Logout

Experiment 3

Case Study on Point of Sale Terminal

Key Events in the POS Terminal Case Study

1. Actors Involved

- **Customer**: Purchasing goods or services.
- **Clerk**: Operating the POS terminal.
- **Credit Payment Service**: Processing credit card transactions.

2. Use Case Diagram

Illustrates interactions:

- **Initiate Transaction**: Clerk starts a transaction.
- **Scan Items**: Clerk scans items.
- **Process Payment**: System processes payment.
- **Print Receipt**: System generates a receipt.

3. Sequence Diagram

Details order of operations:

- Customer approaches clerk.
- Clerk scans items.
- POS system calculates total and prompts for payment.
- Clerk selects payment method; POS system communicates with credit service.
- POS system prints receipt after successful payment.

4. Activity Diagram

Outlines transaction workflow:

- Start transaction.
- Scan items.
- Choose payment method.
- Handle payment confirmation.
- Print receipt or display error if payment fails.

Conclusion

UML diagrams provide a comprehensive view of POS terminal functionalities, aiding development and enhancing team communication.

Use Cases for the Point of Sale (POS) Terminal

Primary Use Cases

1. **Initiate Transaction**
- **Actor**: Clerk
- **Description**: Clerk starts a new transaction.
2. **Scan Items**
- **Actor**: Clerk
- **Description**: Clerk scans items, updating the total.
3. **Apply Discounts**
- **Actor**: Clerk
- **Description**: Clerk applies discounts or promotions.
4. **Process Payment**
- **Actors**: Clerk, Customer
- **Description**: System processes payment (cash, credit card, mobile payment).
5. **Print Receipt**
- **Actor**: Clerk
- **Description**: System generates and prints a receipt.

6. **Handle Returns/Exchanges**
- **Actor**: Clerk
- **Description**: Clerk processes returns or exchanges.
7. **View Sales Reports**
- **Actor**: Manager/Clerk
- **Description**: Access sales reports to analyze data and manage inventory.
8. **Manage Inventory**
- **Actor**: Manager
- **Description**: Update inventory levels, add new products, remove discontinued items.

Conclusion
These use cases illustrate POS terminal functionalities, ensuring the system meets user needs and operates efficiently.

Domain Classes for the Point of Sale (POS) Terminal

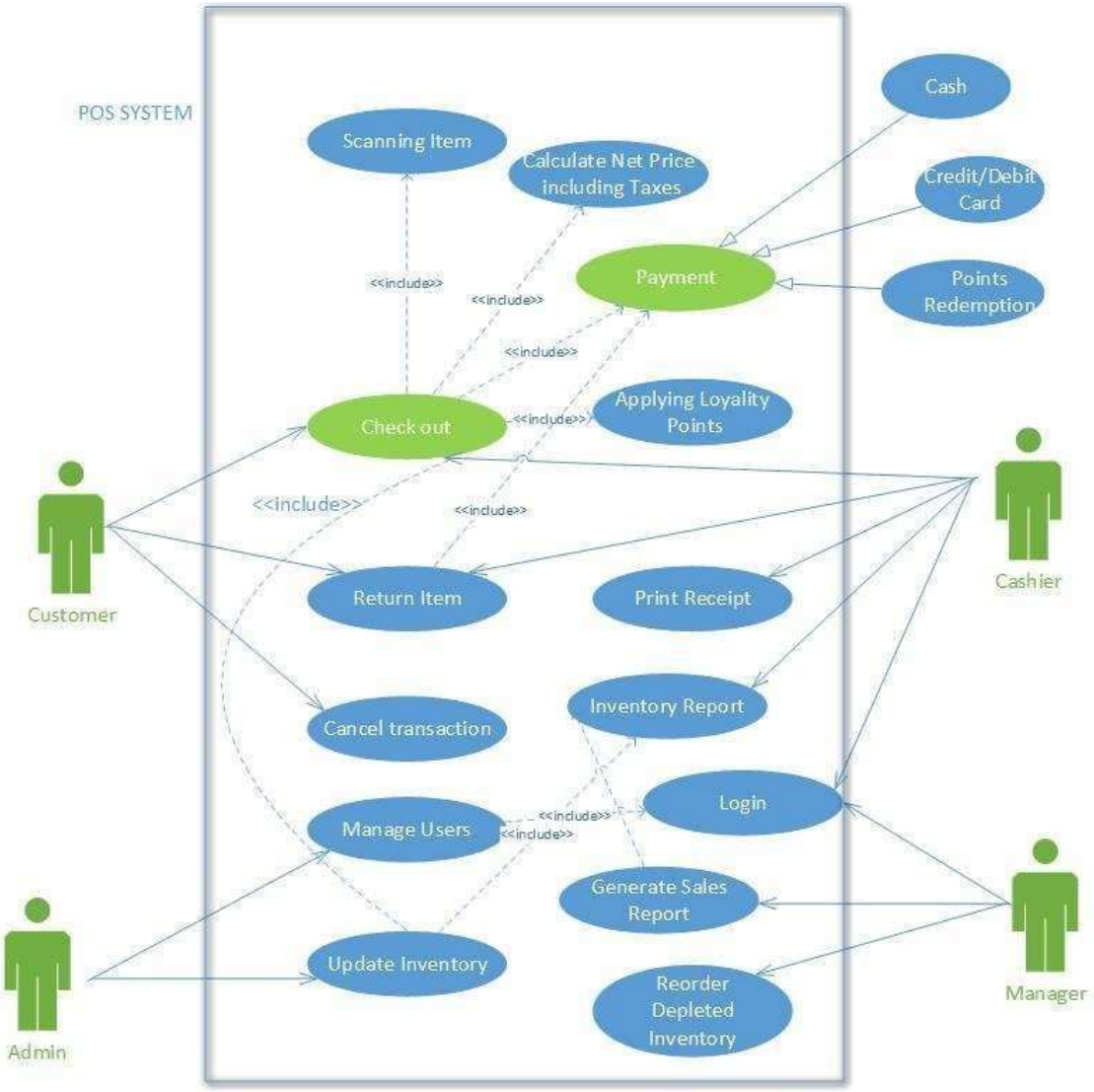
Primary Domain Classes
1. **Transaction**
- **Attributes**: transactionID, date, totalAmount, paymentMethod.
- **Relationships**: Associates with Item, Customer, Clerk, Receipt.
2. **Item**
- **Attributes**: itemID, name, price, quantity.
- **Relationships**: Associates with Transaction.
3. **Customer**
- **Attributes**: customerID, name, contactInfo (optional).
- **Relationships**: Associates with Transaction.
4. **Clerk**
- **Attributes**: clerkID, name, shift (optional).
- **Relationships**: Associates with Transaction.
5. **Receipt**
- **Attributes**: receiptID, transactionID, date, items.
- **Relationships**: Associates with Transaction.
6. **Payment**
- **Attributes**: paymentID, amount, paymentDate, paymentStatus.
- **Relationships**: Associates with Transaction.

Conclusion

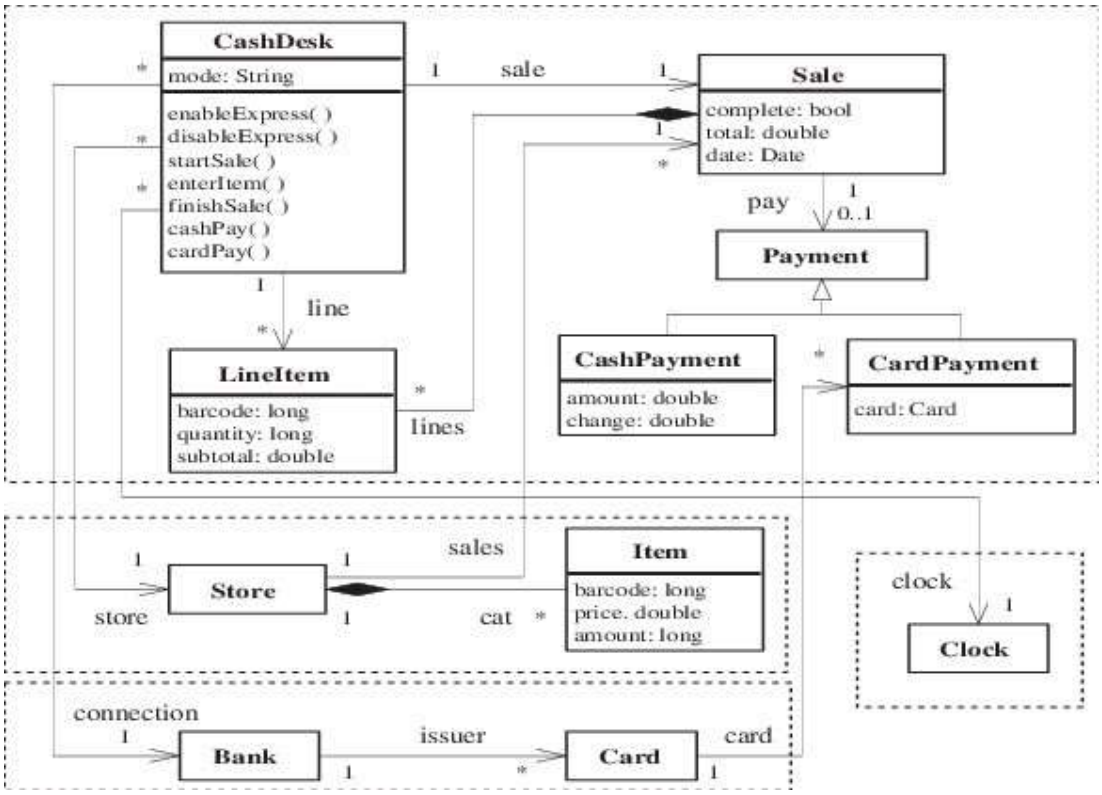
The domain classes identified for the POS terminal provide a structured representation of the system's key entities and their interactions. By defining these classes, developers can create a robust model that supports the functionalities required for efficient transaction processing in a retail environment. This analysis aids in understanding the relationships and data flow within the POS system, facilitating better design and implementation.

Event ID	Event Name	Trigger	Action	Outcome
1	Initiate Transaction	Customer approaches the register	Clerk selects "New Transaction" on the POS terminal	A new transaction session is created.
2	Scan Item	Clerk scans an item	System retrieves item details and updates transaction total	Item is added to the transaction.
3	Apply Discount	Clerk selects discount option	System applies discount to the selected item or total	Updated transaction total reflects the discount.
4	Process Payment	Clerk selects payment method	System processes payment through the chosen method	Payment is authorized, and transaction is completed.
5	Print Receipt	Payment is successful	System generates and prints a receipt	Customer receives a receipt for the transaction.
6	Handle Return/Exchange	Customer requests a return	Clerk verifies the original transaction and processes return	Inventory is updated, and customer is refunded.
7	View Sales Reports	Manager requests report	System generates sales report based on specified criteria	Manager reviews sales performance and inventory data.
8	Manage Inventory	Manager updates inventory	System updates item details or stock levels	Inventory is accurately maintained.

USE CASE DIAGRAM



DOMAIN CLASS DIAGRAM



Experiment 4

Library Management System Case Study

a) Identify and Analyze Events

In a Library Management System, key events can include:

- **User Registration**: A new user registers to access library services.
- **Book Search**: Users search for available books in the library.
- **Book Borrowing**: Users borrow books from the library.
- **Book Returning**: Users return borrowed books.
- **Fine Payment**: Users pay fines for overdue books.
- **User Login/Logout**: Users log in to or out of the system.
- **Book Reservation**: Users reserve books that are currently checked out.
- **View Borrowing History**: Users view their past borrowing records.

These events represent significant interactions between users and the system, forming the basis for further analysis.

b) Identify Use Cases

Based on the identified events, the following use cases can be defined:

1. **Register User**
2. **Login User**
3. **Search for Books**
4. **Borrow Book**
5. **Return Book**
6. **Pay Fine**
7. **Reserve Book**
8. **View Borrowing History**

Each use case describes a specific functionality that the system provides to its users.

c) Identify Event Table

An event table organizes the events and their corresponding use cases:

Event	Use Case
User Registration	Register User
User Login	Login User
Book Search	Search for Books
Book Borrowing	Borrow Book
Book Returning	Return Book
Fine Payment	Pay Fine
Book Reservation	Reserve Book
View Borrowing History	View Borrowing History

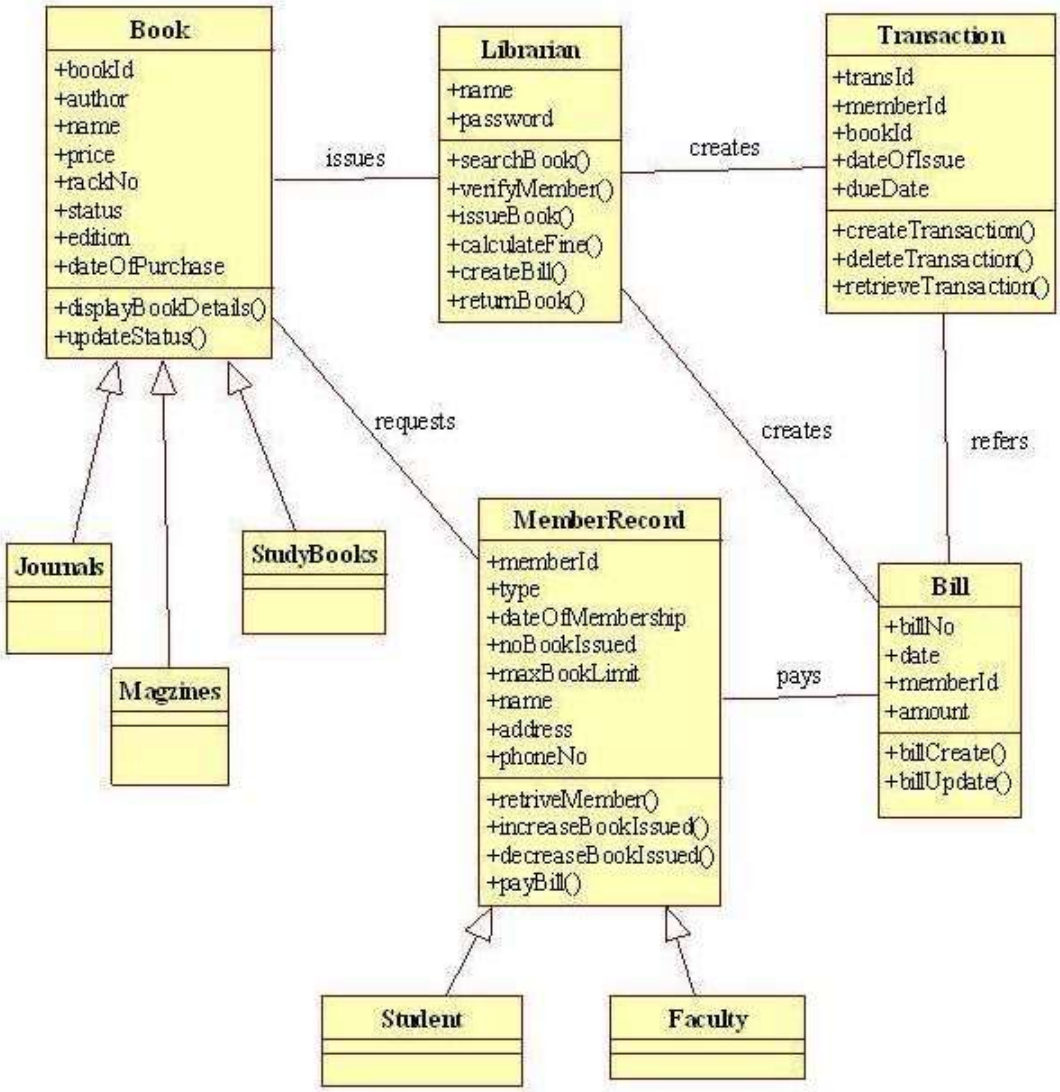
d) Identify and Analyze Domain Classes

Domain classes represent the key entities in the Library Management System. Key classes include:

- **User**: Represents library users, including attributes like userID, name, email, and methods for registration and login.
- **Book**: Represents books in the library, with attributes like bookID, title, author, availability, and methods for searching and reserving.
- **Loan**: Represents the borrowing of books, including attributes like loanID, userID, bookID, borrowDate, and returnDate.
- **Fine**: Represents fines associated with overdue books, with attributes like fineID, userID, amount, and methods for payment processing.

e) Represent Class Diagram for Library Management System

The class diagram for the Library Management System can be represented as follows:



Experiment

Railway Reservation System Case Study

Sequence Diagrams for Use Cases

For a Railway Reservation System, typical use cases might include:

1. User Registration
2. Login
3. Search Train
4. Book Ticket
5. Cancel Ticket
6. View Booking History

Each of these use cases can be represented with a sequence diagram that illustrates the interactions between the user and the system components. For example, the sequence diagram for Book Ticket would show the user interacting with the UI, the Controller processing the request, and the Model updating the database.

MVC Classes/Objects for Each Use Case

The Model-View-Controller (MVC) architecture can be broken down as follows:

- Model:
  - 'User'
  - 'Train'
  - 'Booking'
  - 'Payment'

- View:
  - 'UserRegistrationView'
  - 'LoginView'
  - 'TrainSearchView'

- ‘BookingView’

- ‘CancellationView’

- ‘BookingHistoryView’

- Controller:

- ‘UserController’

- ‘TrainController’

- ‘BookingController’

- ‘PaymentController’

Each controller handles the logic for its respective view and interacts with the model to perform operations.

Communication Diagrams for Each Use Case

Communication diagrams illustrate the interactions among objects in the system. For instance, in the Book Ticket use case, the communication diagram would show:

- ‘User’ sends a request to ‘BookingController’

- ‘BookingController’ interacts with ‘TrainController’ to check availability

- ‘BookingController’ updates the ‘Booking’ model

- ‘BookingController’ notifies the ‘BookingView’ to update the user interface

This pattern would be similar for other use cases, with variations based on the specific interactions required.

Detailed Design Class Diagram

In developing a detailed design class diagram, we can apply GRASP (General Responsibility Assignment Software Patterns) principles for responsibility assignment. Heres a breakdown of responsibilities:

- User: Responsible for managing user data and authentication.

- Train: Responsible for managing train schedules and availability.

- Booking: Responsible for handling ticket bookings and cancellations.

- Payment: Responsible for processing payments and refunds.

The class diagram would show relationships such as:

- 'User' has a one-to-many relationship with 'Booking'.

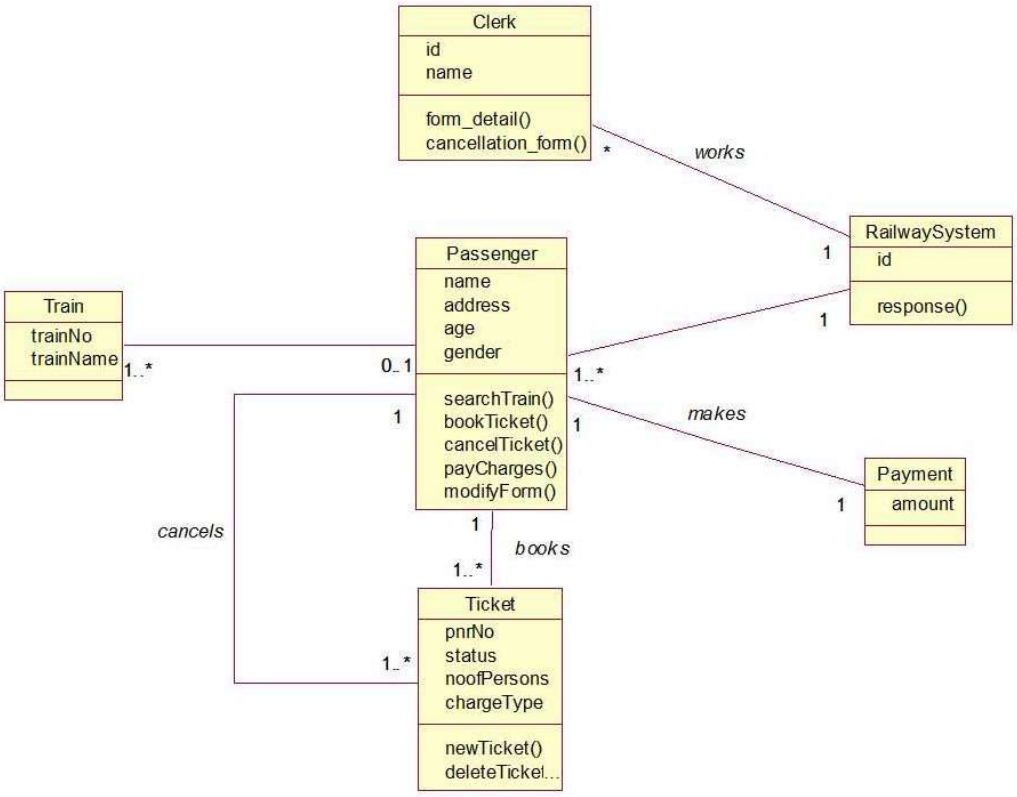
- 'Train' has a one-to-many relationship with 'Booking'.

- 'Booking' interacts with 'Payment' for transaction processing.

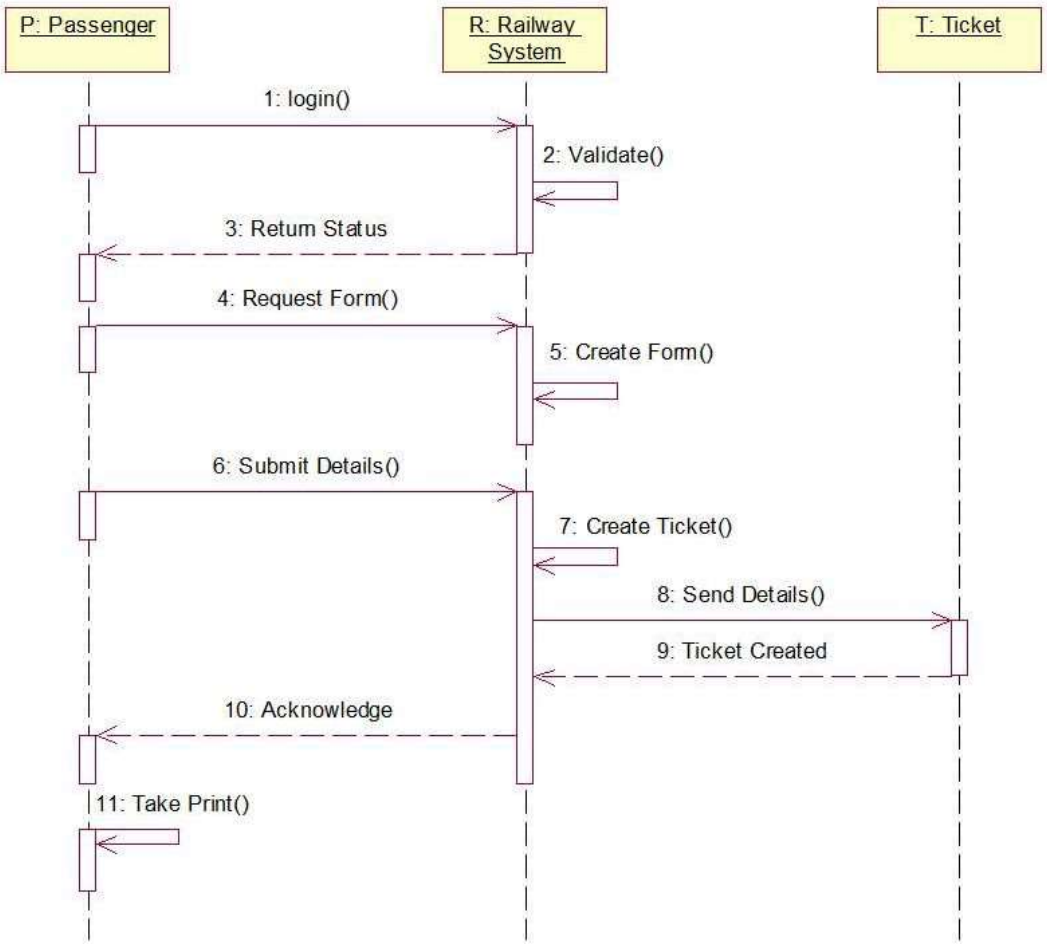
This structured approach ensures that each class has a clear responsibility, promoting maintainability and scalability in the system design.

Conclusion

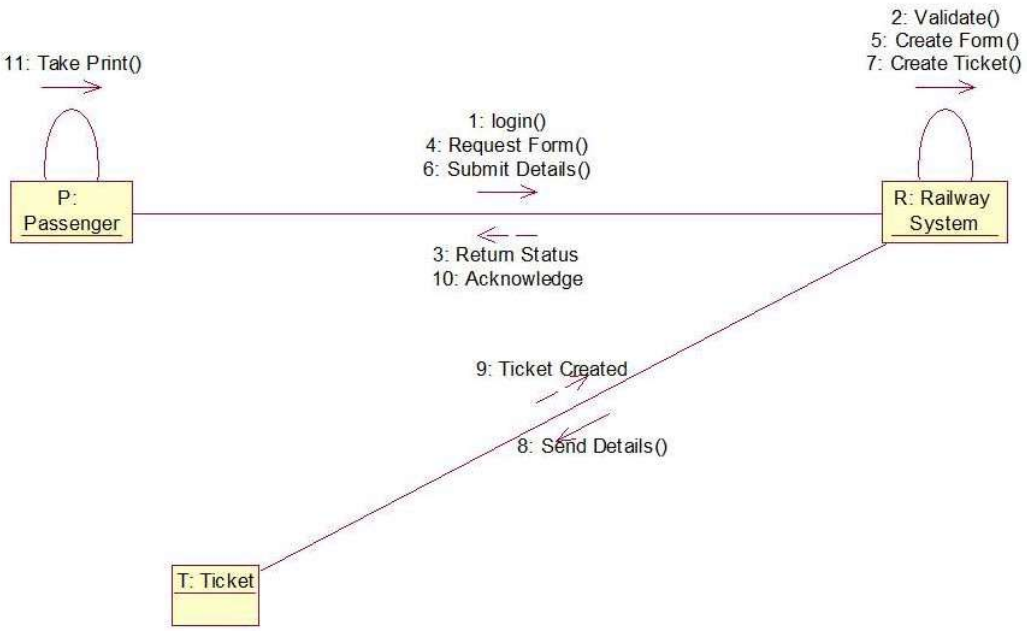
The Railway Reservation System can be effectively modeled using sequence diagrams, MVC architecture, communication diagrams, and a detailed design class diagram. This structured approach not only clarifies the interactions and responsibilities within the system but also aids in the development and maintenance of the application.



USE CASE DIAGRAM



SEQUENCE DIAGRAM



COMMUNICATION DIAGRAM

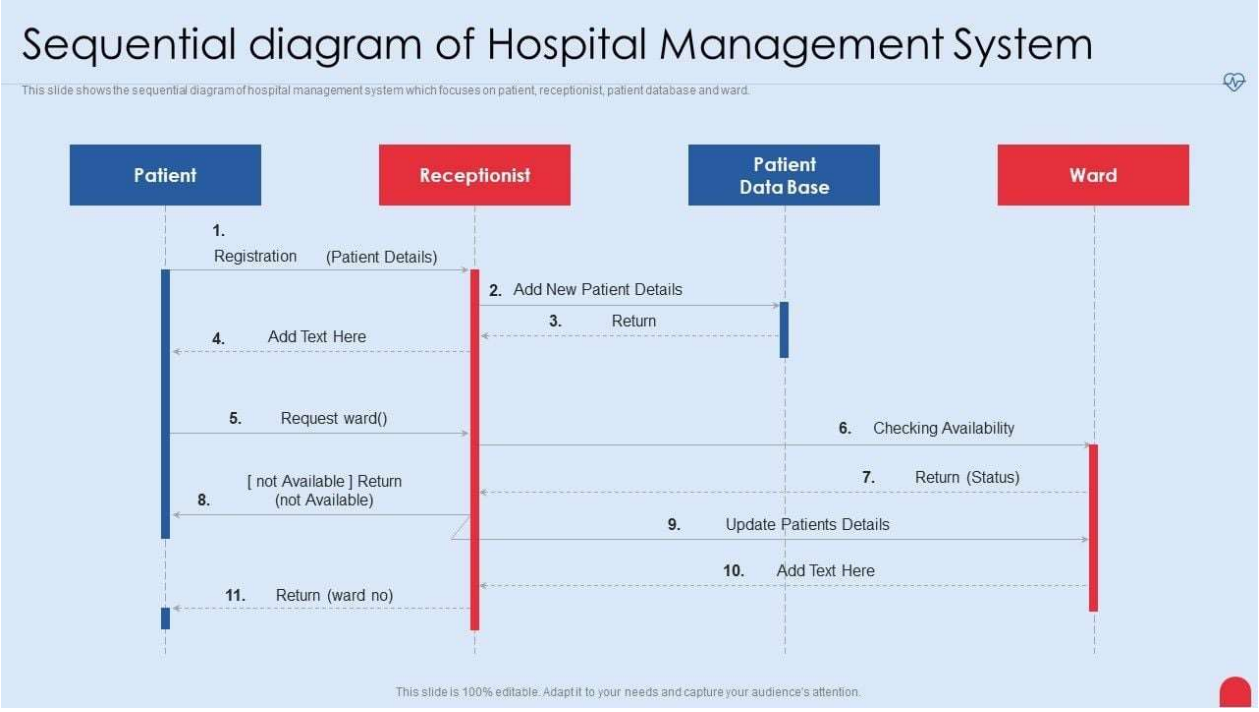
Experiment 6

Hospital Management System Case Study

a) Develop Sequence Diagram for Each Use Case

For a Hospital Management System, typical use cases might include:

1. Patient Registration
2. Doctor Appointment Booking
3. Patient Check-in
4. Medical Record Access
5. Billing and Payment Processing
6. Prescription Management



b) Develop Communication Diagrams for Each Use Case

Communication diagrams illustrate the interactions among objects in the system. For the **Doctor Appointment Booking** use case, the communication diagram would show:

- Patient sends a request to AppointmentController.
- AppointmentController interacts with DoctorModel to check availability.
- AppointmentController updates the AppointmentModel.
- AppointmentController notifies the NotificationService to send confirmation.
- NotificationService sends a confirmation to the Patient.

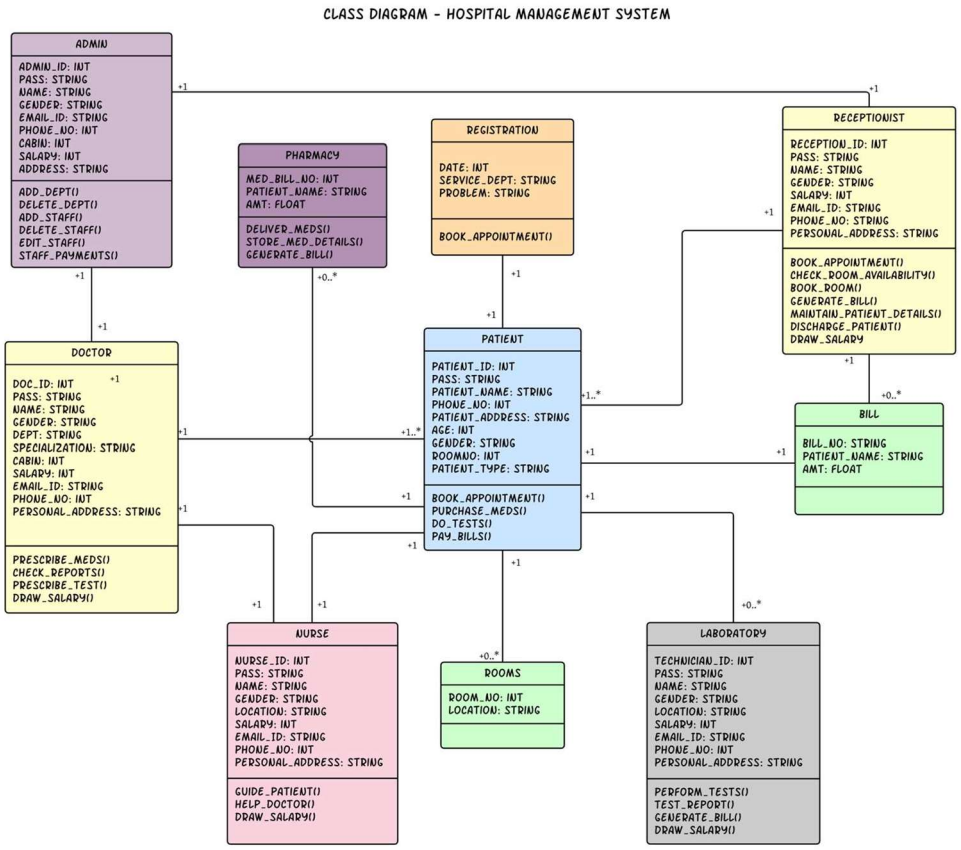
This pattern would be similar for other use cases, with variations based on the specific interactions required.

c) Develop Detailed Design Class Model (Using GRASP Patterns for Responsibility Assignment)

In developing a detailed design class model, we can apply GRASP (General Responsibility Assignment Software Patterns) principles for responsibility assignment. Here's a breakdown of responsibilities:

- **Patient**: Responsible for managing patient data and registration.
- **Doctor**: Responsible for managing doctor information and availability.
- **Appointment**: Responsible for handling appointment scheduling and notifications.
- **MedicalRecord**: Responsible for managing patient medical records.
- **Billing**: Responsible for processing payments and managing billing information.
- **NotificationService**: Responsible for sending notifications (e.g., appointment confirmations).

The class diagram can be represented as follows:



Conclusion

The Hospital Management System can be effectively modeled using sequence diagrams, communication diagrams, and a detailed design class model. This structured approach clarifies the interactions and responsibilities within the system, aiding in the development and maintenance of the application. Each component plays a crucial role in ensuring the system operates smoothly and meets user needs.

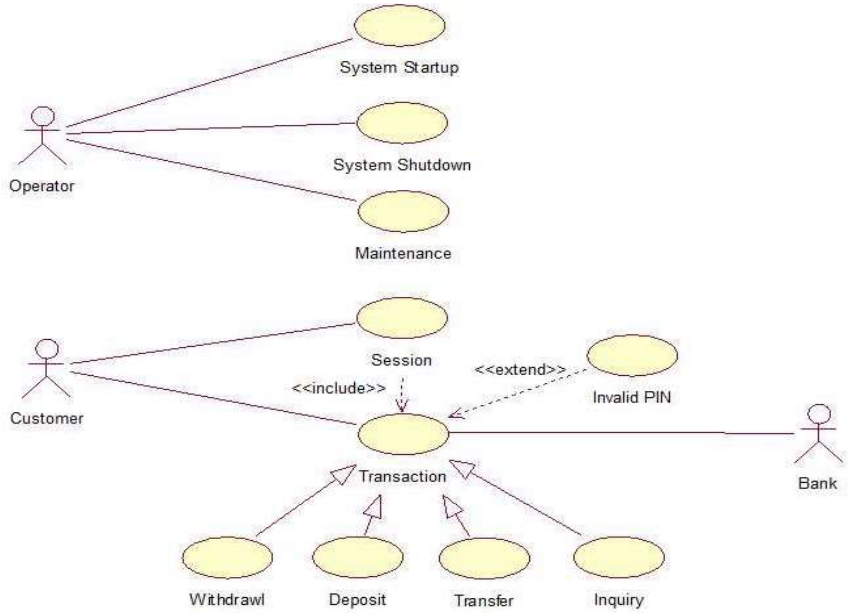
Experiment 7

ATM Application Case Study

a) Use Case Diagram

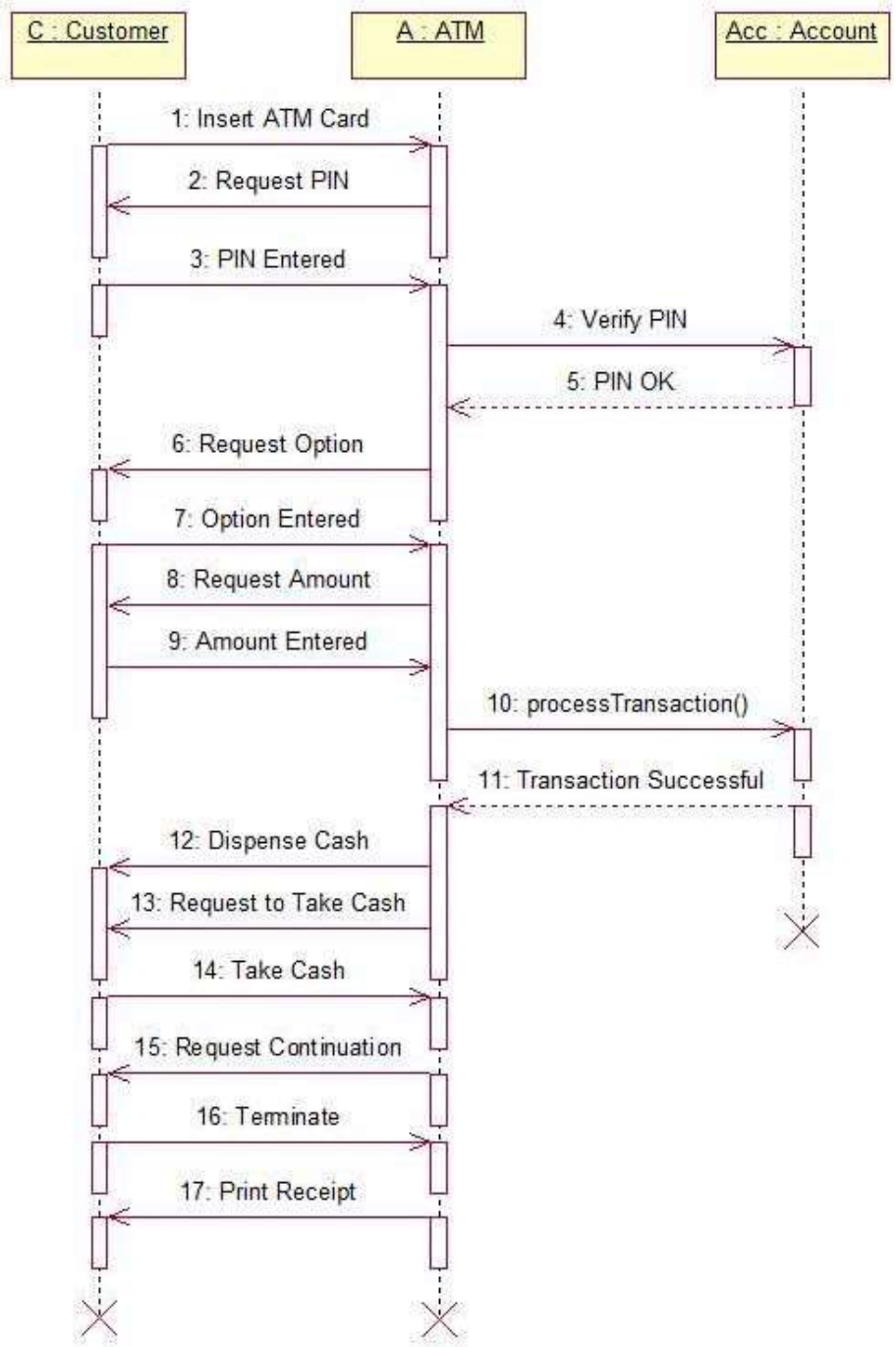
The Use Case Diagram for an ATM application typically includes the following actors and use cases:

- **Actors**:
  - Customer
  - Bank Server
  - ATM Technician
- **Use Cases**:
  - Insert Card
  - Enter PIN
  - Check Balance
  - Withdraw Cash
  - Deposit Funds
  - Transfer Funds
  - Print Receipt
  - Eject Card
  - Maintenance (for ATM Technician)
  - Here's a visual representation of the Use Case Diagram:



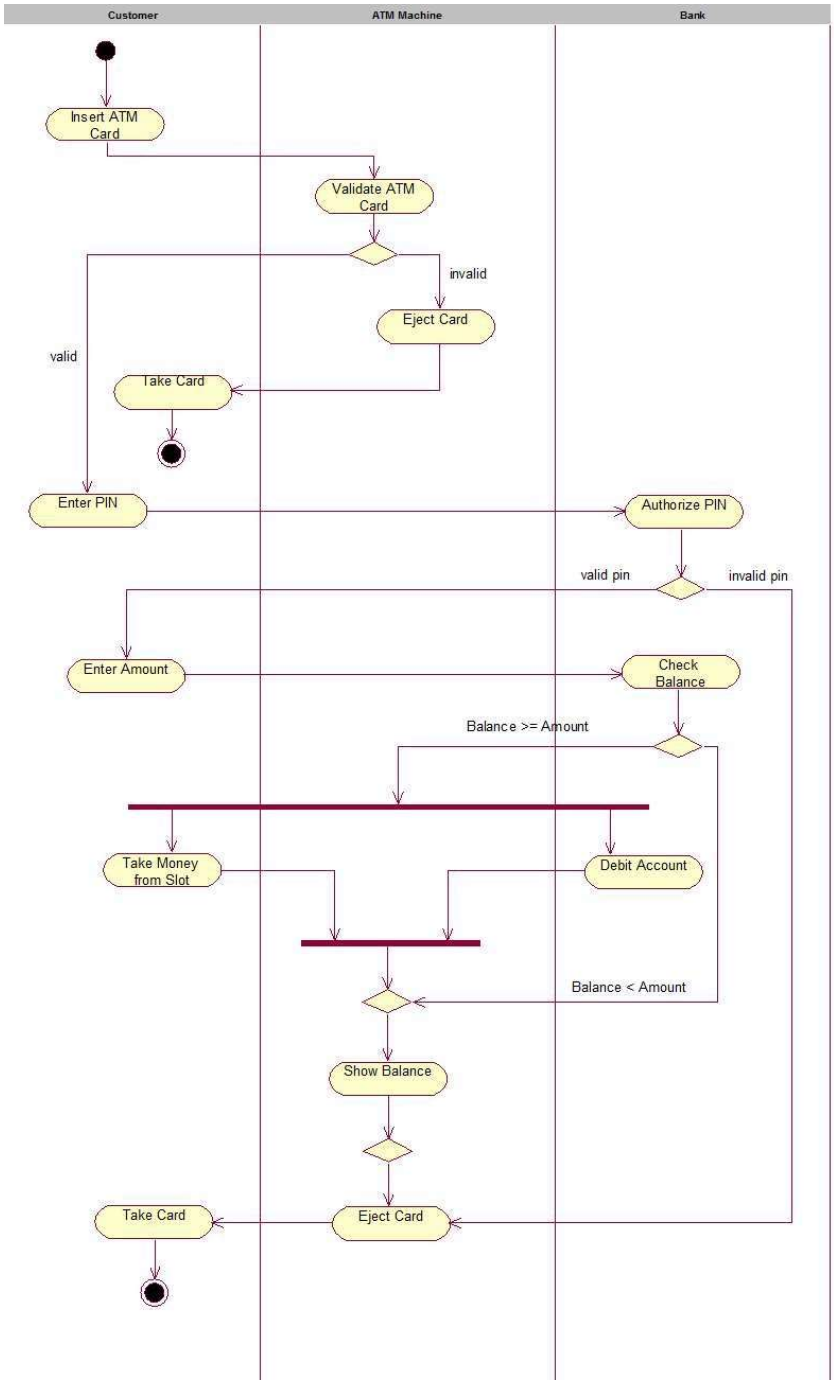
b) Sequence Diagram

For the **Withdraw Cash** use case, the sequence diagram illustrates the interactions between the customer, ATM, and bank server:



c) Activity Diagram

The Activity Diagram for the **Withdraw Cash** use case outlines the flow of activities involved in the process:



Experiment8

Auction Application Case Study

a) Use Case Diagram

The Use Case Diagram for an Auction application typically includes the following actors and use cases:

- **Actors**:
  - Buyer
  - Seller
  - Administrator
- **Use Cases**:
  - Register Account
  - Create Auction
  - Place Bid
  - View Auction Listings
  - End Auction
  - Make Payment
  - View Bidding History
  - Manage Users (for Administrator)

Here's a visual representation of the Use Case Diagram:

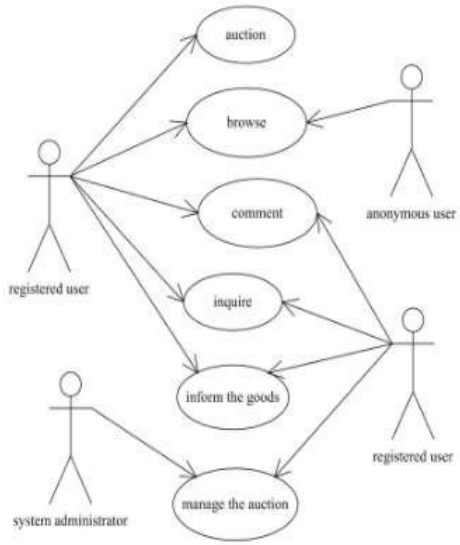
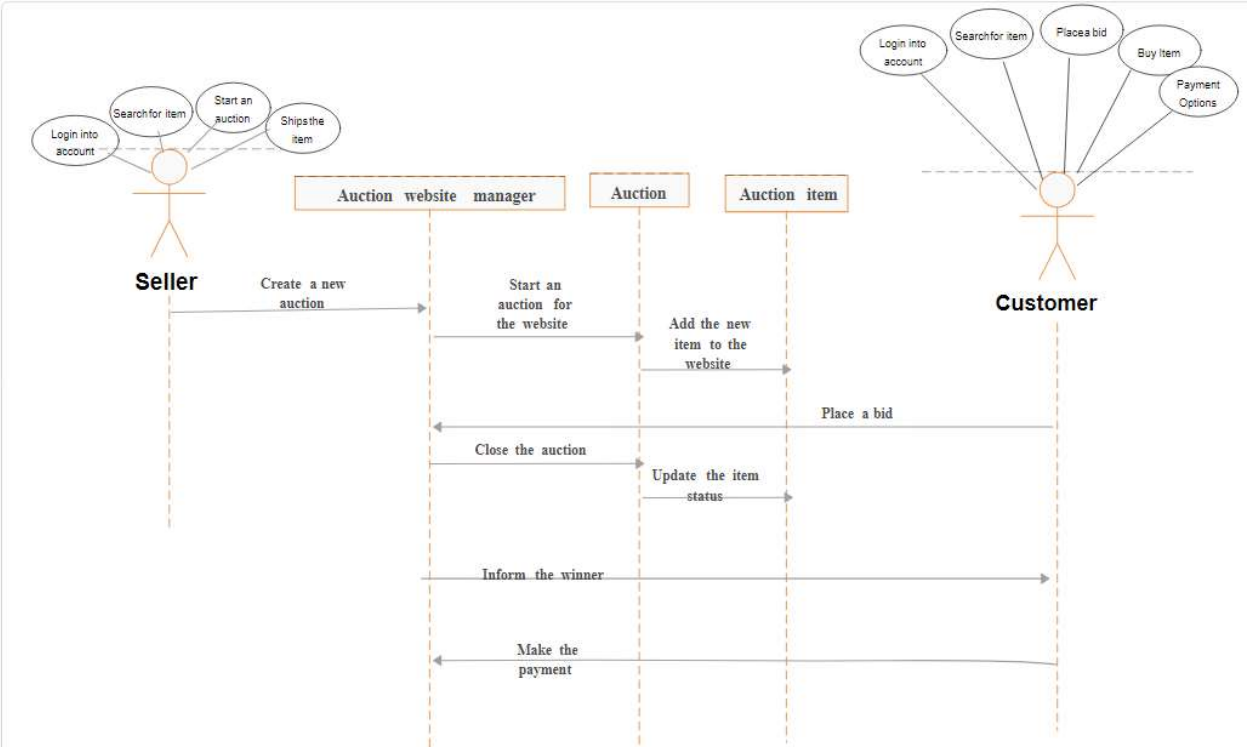


Figure 1. The use case diagram.

b) Sequence Diagram

For the **Place Bid** use case, the sequence diagram illustrates the interactions between the buyer, auction system, and bank server:



c) Activity Diagram

The Activity Diagram for the **Place Bid** use case outlines the flow of activities involved in the process:

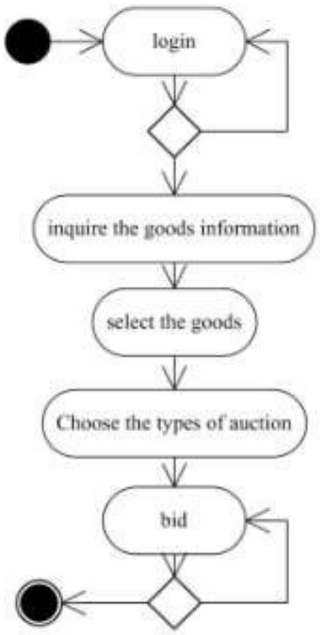


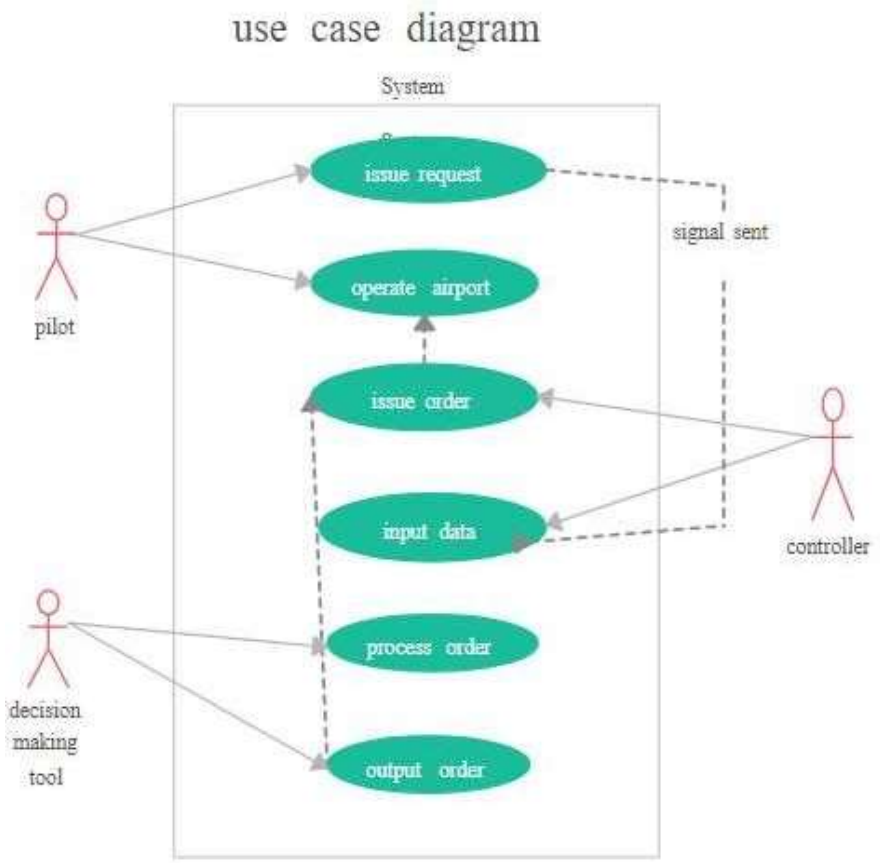
Figure 2. The auction activity diagram.

Experiment9

Multithreaded Airport Simulation Case Study

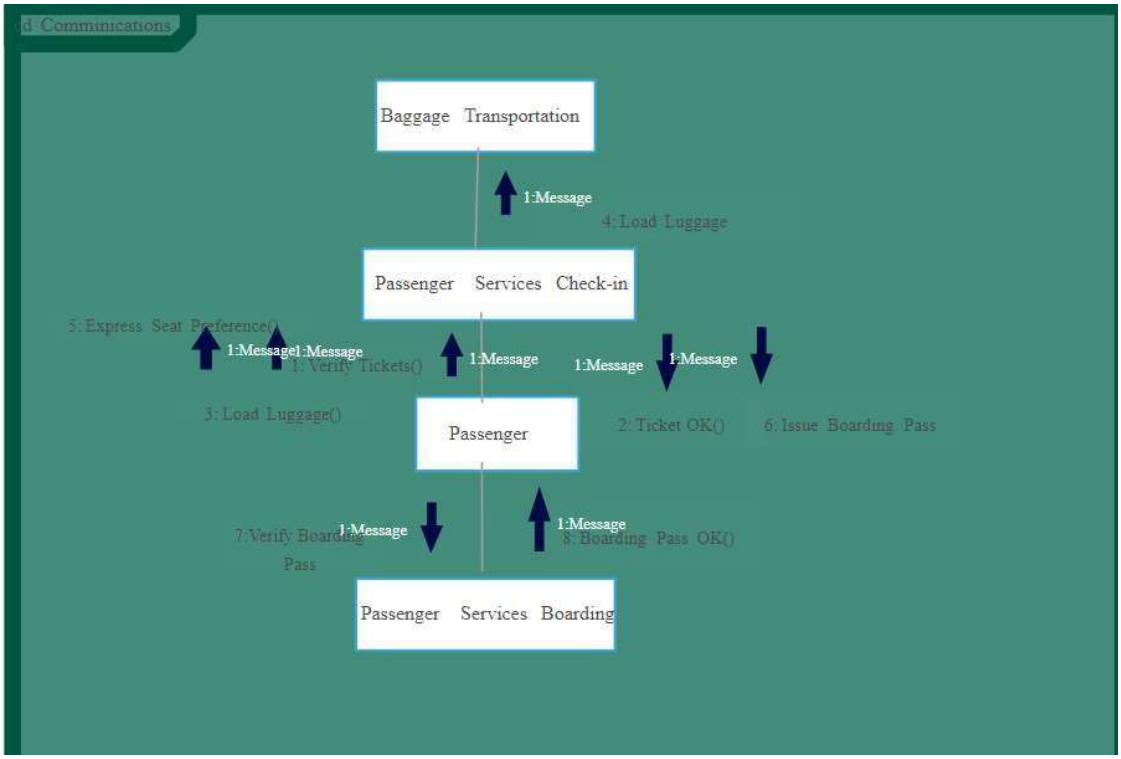
a) Sequence Diagram

In a multithreaded airport simulation, the sequence diagram can illustrate the interactions between various components such as the **Air Traffic Controller**, **Runway**, **Aircraft**, and **Ground Crew** during the landing and takeoff process. Here's an example of a sequence diagram for an aircraft landing:



b) Collaboration Diagram

The collaboration diagram for the same scenario illustrates the relationships and interactions among the objects involved in the landing process. Here's a representation of the collaboration diagram:



Experiment 10

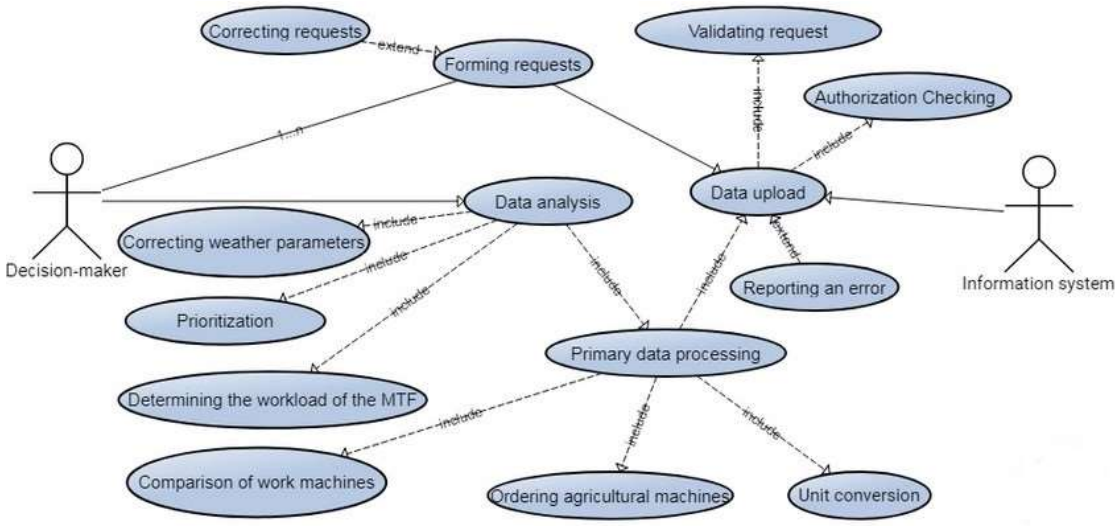
Simulated Company Case Study

a) Use Case Diagram

The Use Case Diagram for a simulated company typically includes the following actors and use cases:

- **Actors**:
  - Employee
  - Manager
  - Customer
  - Administrator
- **Use Cases**:
  - Register Account
  - Login
  - Create Project
  - Assign Task
  - Submit Report
  - View Reports
  - Manage Users (for Administrator)
  - Provide Feedback (for Customer)

Here's a visual representation of the Use Case Diagram:



b) Sequence Diagram

For the **Create Project** use case, the sequence diagram illustrates the interactions between the employee, project management system, and manager:

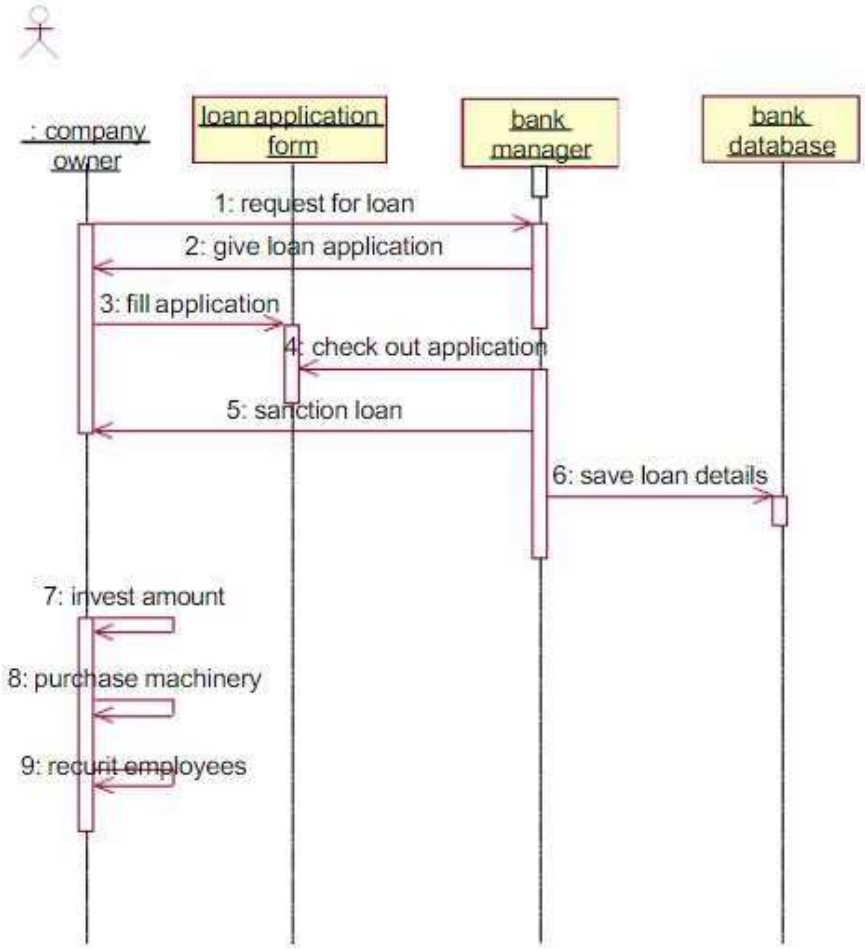


Figure 3.2.2 Sequence diagram for company owner

c) State Chart Diagram

The State Chart Diagram for a **Project** in the simulated company outlines the various states a project can be in during its lifecycle:

Figure 4.2 State chart diagram for simulated company