Structural Diagrams

- Class Diagram
- Object Diagram
- Component Diagram
- Deployment Diagram

Behavioral Diagrams

- Use Case Diagram
- Sequence Diagram
- Activity Diagram
- Diagram

Interaction Diagrams

- Communication Diagram
- Timing Diagram

ATM Case Study and Online Railway Reservation System Phases in software development project, overview and need. Understand problems in existing systems and perform system analysis: Requirement analysis, SRS

1. ATM Case Study

The **ATM (Automated Teller Machine)** system is designed to allow users to interact with their bank accounts for transactions such as withdrawals, deposits, and balance checks.

Key Points for ATM Case Study

- **Actors**: Customer, Bank, ATM Technician
- **Use Cases**: Withdraw Cash, Deposit Cash, Check Balance, Transfer Funds, Change PIN, View Transaction History
- **System Requirements**:
  - **Functional Requirements**: Authentication of user, display of account balance, withdrawal/deposit operations.
  - **Non-functional Requirements**: Security, usability, performance, reliability.

2. Online Railway Reservation System

The **Online Railway Reservation System** allows users to book train tickets, check schedules, and manage reservations online.

Key Points for Online Railway Reservation System

- **Actors**: Passenger, Ticketing Agent, Administrator

- **Use Cases**: Book Ticket, Cancel Ticket, Check Train Status, Make Payment, Print Ticket
- **System Requirements**:
  - **Functional Requirements**: User login, search and select train, reservation/cancellation, ticket pricing, payment gateway integration.
  - **Non-functional Requirements**: High availability, data integrity, user-friendly interface, security.

3. Phases in Software Development Project

Understanding the phases is essential in building a structured project. Common phases in the **Software Development Life Cycle (SDLC)** include:

- **Requirement Analysis**: Gather and define requirements. Tools include interviews, surveys, and studying existing systems.
- **System Design**: Outline architecture and components to meet requirements, defining the system's structure.
- **Implementation**: Coding and integrating different modules based on design specifications.
- **Testing**: Verify functionality, security, and performance through various test cases.
- **Deployment**: Deploy the final system to the user's environment, with training if necessary.
- **Maintenance**: Resolve any issues and update the system based on feedback.

Need for a Phased Approach

- **Structured Development**: Ensures each aspect of the project is well-defined and organized.
- **Risk Mitigation**: Early stages can identify potential risks, reducing the likelihood of issues in later phases.
- **Improved Quality**: Detailed requirements, design, and testing phases lead to higher quality and a more reliable system.

4. System Analysis

System Analysis is crucial for understanding the problems in existing systems and determining solutions.

Problems in Existing Systems

- **ATM System**: Limited accessibility, lack of multi-language support, security vulnerabilities.
- **Online Railway Reservation**: Slow response times during high traffic, issues with real-time seat availability updates, and payment failures.

System Analysis Steps:

- **Requirement Analysis**:
  - Gather and analyze user needs and existing system limitations.
  - Involve stakeholders to understand functional and non-functional requirements.
- **Software Requirements Specification (SRS)**:
  - A comprehensive document outlining system requirements in detail.
  - Divided into **functional** and **non-functional** requirements.
  - Defines system scope, constraints, interfaces, and overall expectations.

SRS in Detail:

An effective **SRS** includes:

1. **Introduction**: Purpose, scope, and overview.
2. **Overall Description**: High-level system perspective.
3. **System Features**: Functional requirements for each feature.
4. **External Interface Requirements**: User interfaces, APIs, etc.
5. **Non-functional Requirements**: Performance, security, reliability.
6. **Assumptions and Dependencies**: Assumptions about the system environment.

Requirement analysis, SRS of both case studies

Requirement Analysis

Requirement Analysis is the process of figuring out what a software system should do by collecting information from people involved, like clients, end users, and managers. The goal is to understand and document all needs and expectations for the system.

Software Requirements Specification (SRS)

An SRS document is a detailed description of a software system's requirements. It serves as a key guide that ensures both clients and developers have a clear, shared understanding of what the system should achieve.

1. ATM Case Study: Requirement Analysis and SRS

Requirement Analysis

In the ATM system, requirement analysis involves identifying the needs of users and defining the scope of functionalities for smooth, secure transactions.

- **Functional Requirements**:
  - **User Authentication**: Ensure user identity verification through card insertion, PIN entry, or biometric authentication.
  - **Account Balance Inquiry**: Display the current account balance after user login.
  - **Cash Withdrawal**: Allow users to withdraw cash up to a daily limit.
  - **Deposit**: Accept deposits, either in cash or check form, and update the user's account balance.
  - **Funds Transfer**: Enable transfer of funds between accounts within the same bank.
  - **PIN Change**: Allow users to change their PIN securely.
  - **Transaction History**: Show recent transactions on the account for review.
- **Non-functional Requirements**:
  - **Security**: Secure PIN and transaction data using encryption, and ensure tamper-resistant hardware.
  - **Usability**: Simple and intuitive interface for users of all ages.
  - **Reliability**: Ensure system availability at least 99.9% of the time.
  - **Performance**: Transactions should process within 2 seconds, and the interface should respond quickly.

2. Online Railway Reservation System: Requirement Analysis and SRS

Requirement Analysis

For the Online Railway Reservation System, requirement analysis identifies the essential features for users to book tickets, view train schedules, and manage reservations.

- **Functional Requirements**:
  - **User Registration and Login**: Allow new users to create accounts and existing users to log in securely.
  - **Search for Trains**: Enable users to search for trains based on destination, date, and class of travel.
  - **Book and Cancel Tickets**: Allow users to book tickets and view booking history. Include a cancellation option with refund rules.

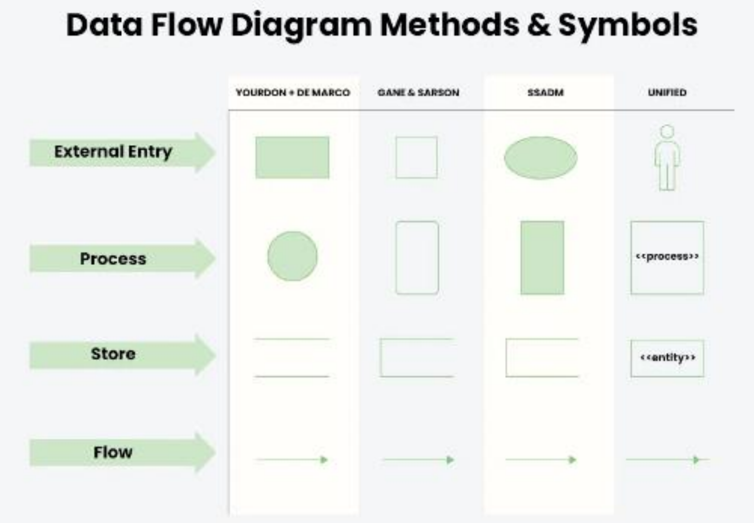
    - **View Train Status**: Real-time updates on train status, including delays and cancellations.
    - **Make Payments**: Secure payment gateway integration for booking tickets.
    - **Notifications**: Send notifications for booking confirmations, cancellations, and updates on train status.
- **Non-functional Requirements**:
    - **Scalability**: Support a large number of concurrent users, especially during peak booking hours.
    - **Reliability**: Ensure data consistency and availability during booking.
    - **Usability**: Simple navigation for users to find information quickly.
    - **Performance**: Quick response time, with no page load exceeding 3 seconds.

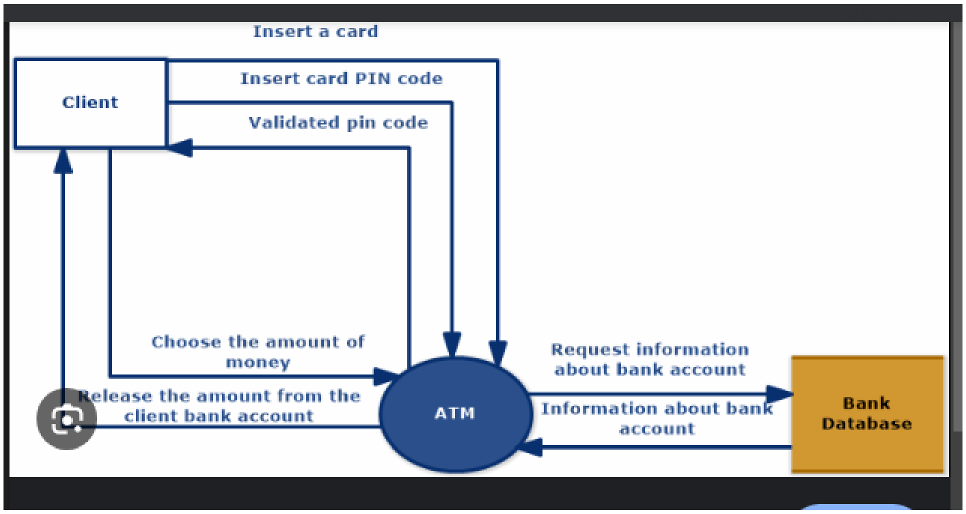
To perform the function-oriented design: Data flow diagrams

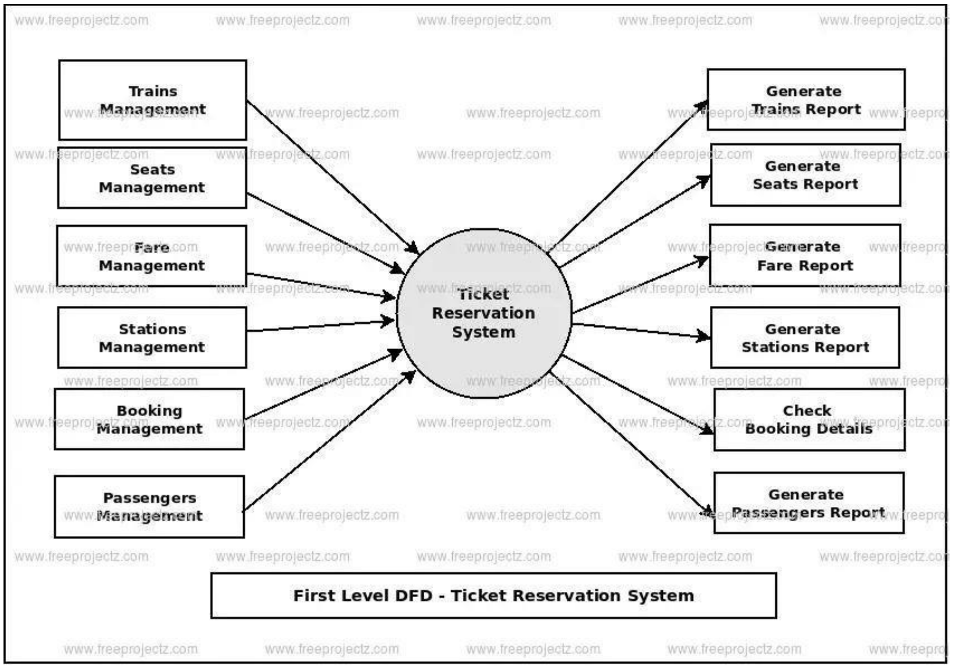
A **Data Flow Diagram (DFD)** shows how data moves through an information system. It's a visual tool that makes it easy for both technical and non-technical users to understand the system's data flow and processes.

Components of a Data Flow Diagram

1. **Process**:
    - Represents actions or functions performed on data (like calculations or data transformations).
    - Usually depicted by a circle or rounded rectangle.
2. **Data Flow**:
    - Represents the movement of data between different parts of the system.
    - Shown as arrows, indicating the direction of data movement.
3. **Data Store**:
    - Represents storage locations for data within the system.
    - Drawn as an open-ended rectangle or parallel lines.
4. **External Entity**:
    - Represents outside sources or destinations of data, such as users, other systems, or external organizations.
    - Usually shown as a rectangle.

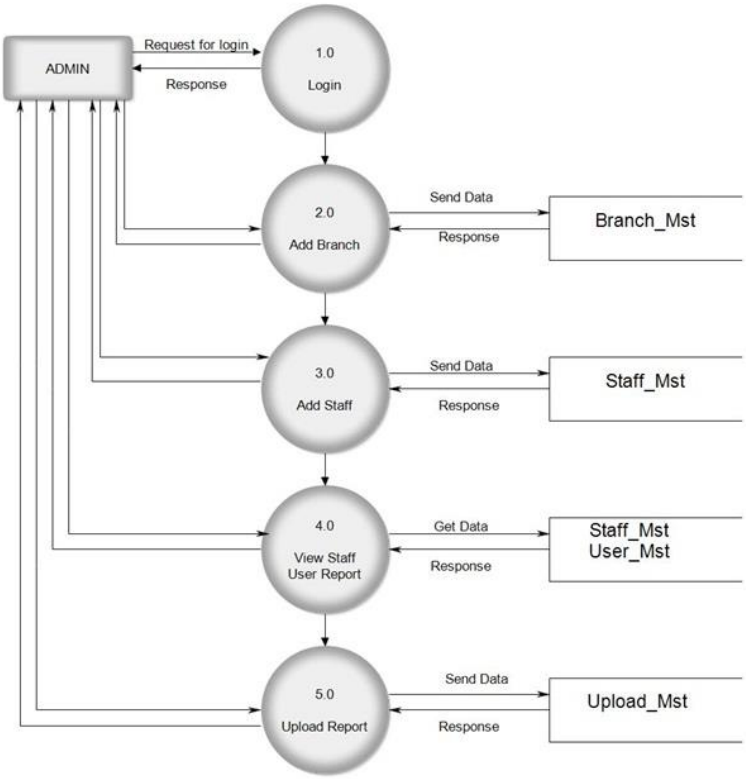

Data Flow Diagram Methods & Symbols

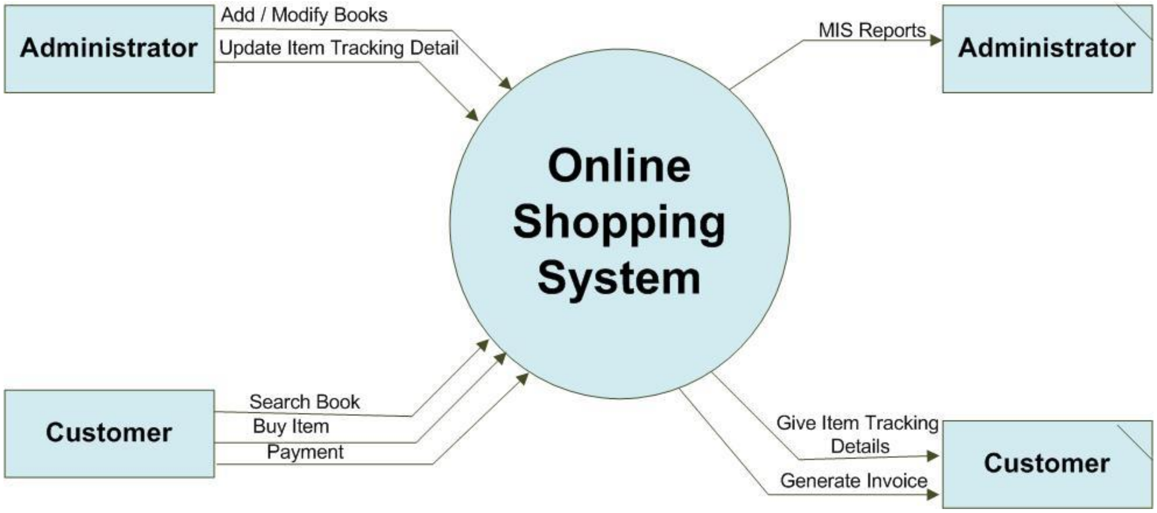

Client
Insert a card
Insert card PIN code
Validated pin code
Choose the amount of money
Release the amount from the client bank account
ATM
Request information about bank account
Information about bank account
Bank Database



First Level DFD - Ticket Reservation System

College management system dfd:



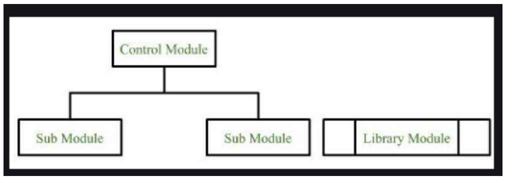


To perform the function-oriented design: Structured chart

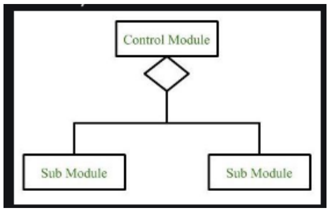
A **Structure Chart** is a diagram that divides a system into smaller parts called modules. Each module shows a specific function, with only its inputs and outputs visible, making it easier to understand how the system works together without revealing internal details.

Symbols in a Structure Chart

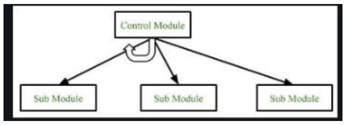
1. **Module**:
   - Represents a specific function or task in the system.
   - **Types**:
     - **Control Module**: Calls multiple submodules.
     - **Submodule**: Part of a parent module.
     - **Library Module**: Reusable, can be called by any module.



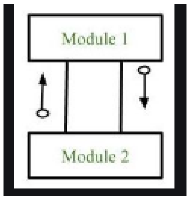
2. **Conditional Call**:
   - Shows a choice between submodules based on a condition. The control module decides which submodule to use.



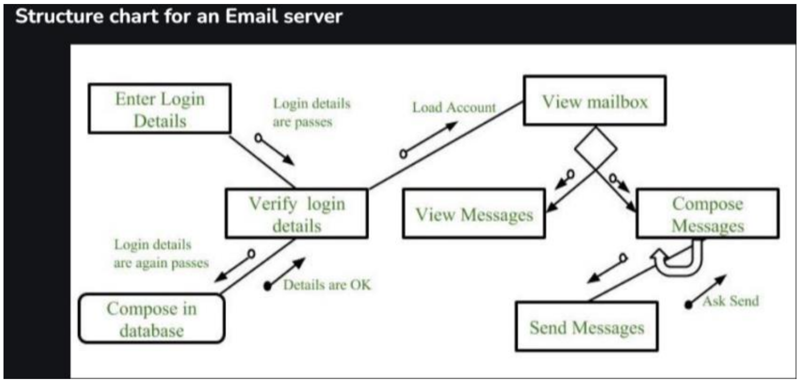
3. **Loop (Repetitive Call)**:
   - Shows repeated execution of a module, represented by a curved arrow around the modules being repeated.

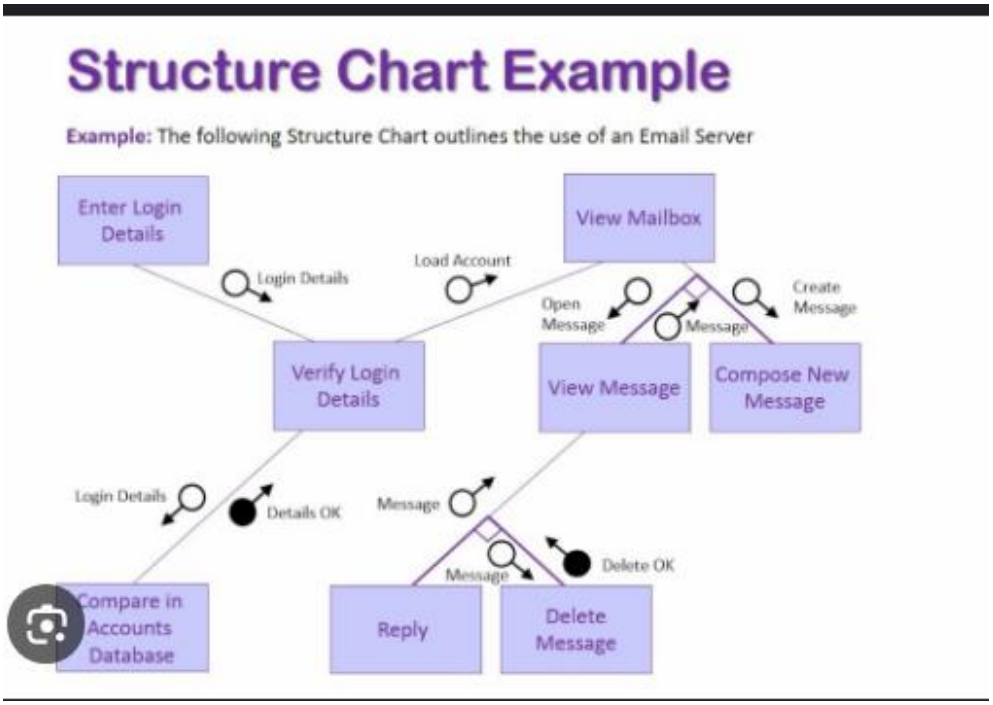


4. **Data Flow**:
   - Represents data passing between modules, shown as an arrow with an open circle at the end.



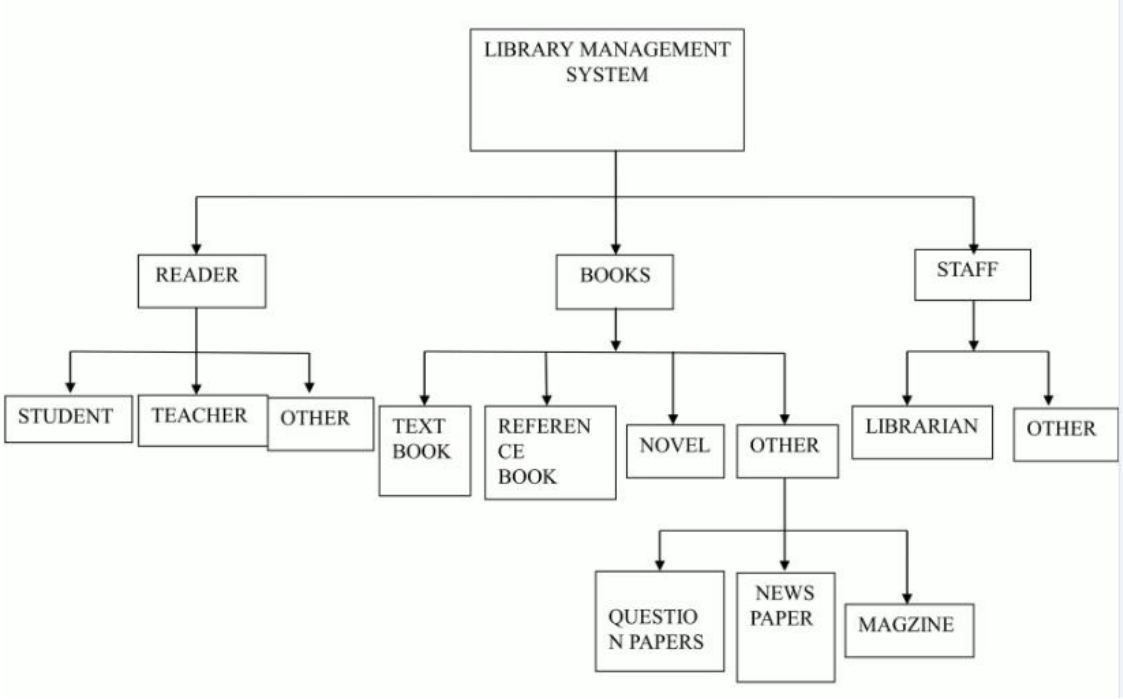
5. **Control Flow**:
   - Represents control passing between modules, shown as an arrow with a filled circle at the end.
6. **Physical Storage**:
   - Shows where data is stored in the system.

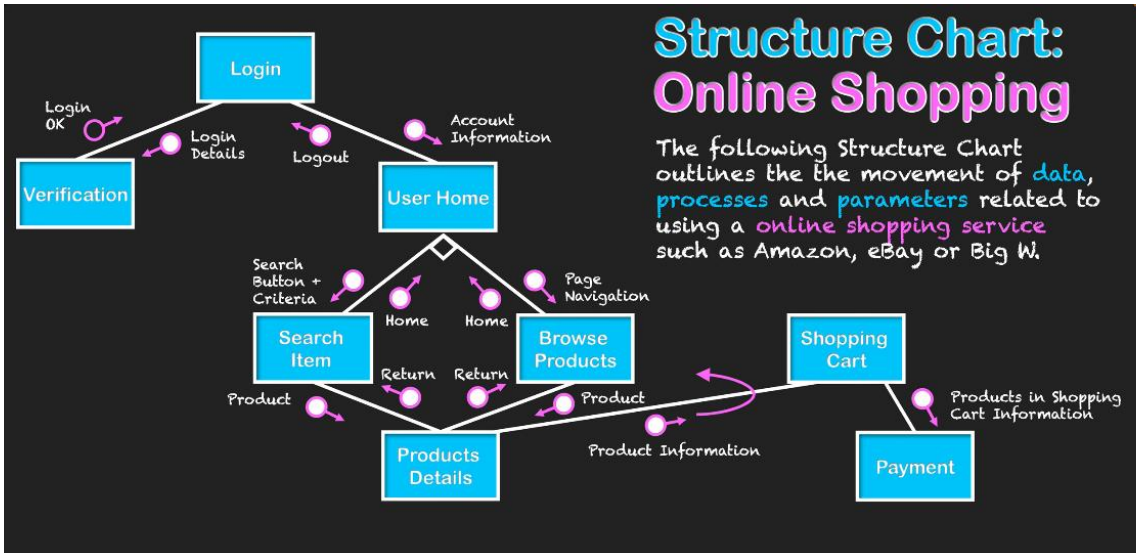




Structure Chart Example

Example: The following Structure Chart outlines the use of an Email Server





Structure Chart: Online Shopping

The following Structure Chart outlines the the movement of data, processes and parameters related to using a online shopping service such as Amazon, eBay or Big W.

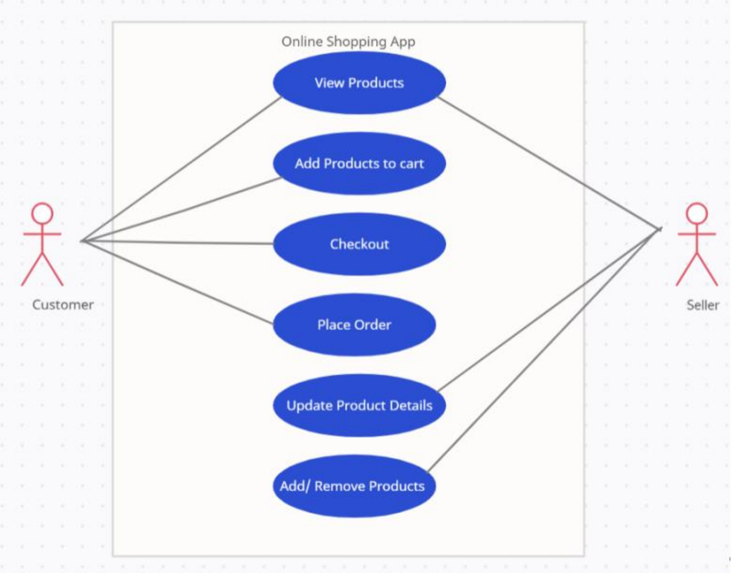
To perform the user’s view analysis: Use case diagram

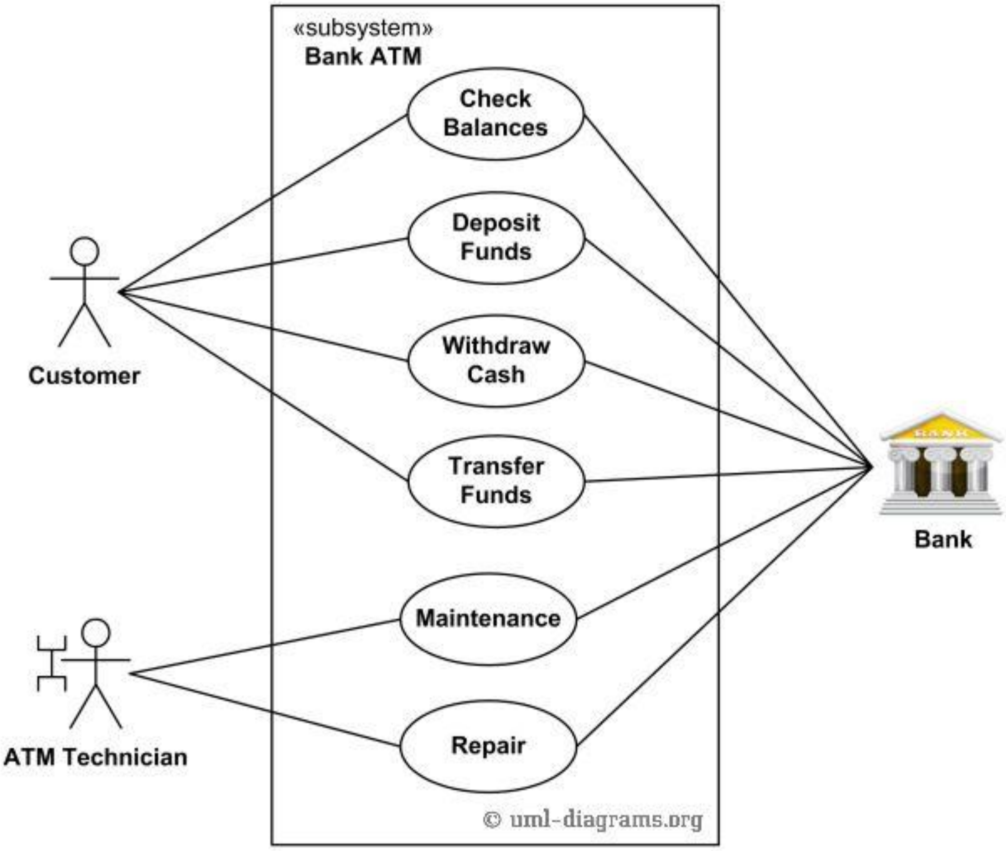
A **Use Case Diagram** in UML shows how users or other systems interact with a system to achieve specific goals. It gives an overview of the system's functions from the user's perspective, highlighting what the system can do.

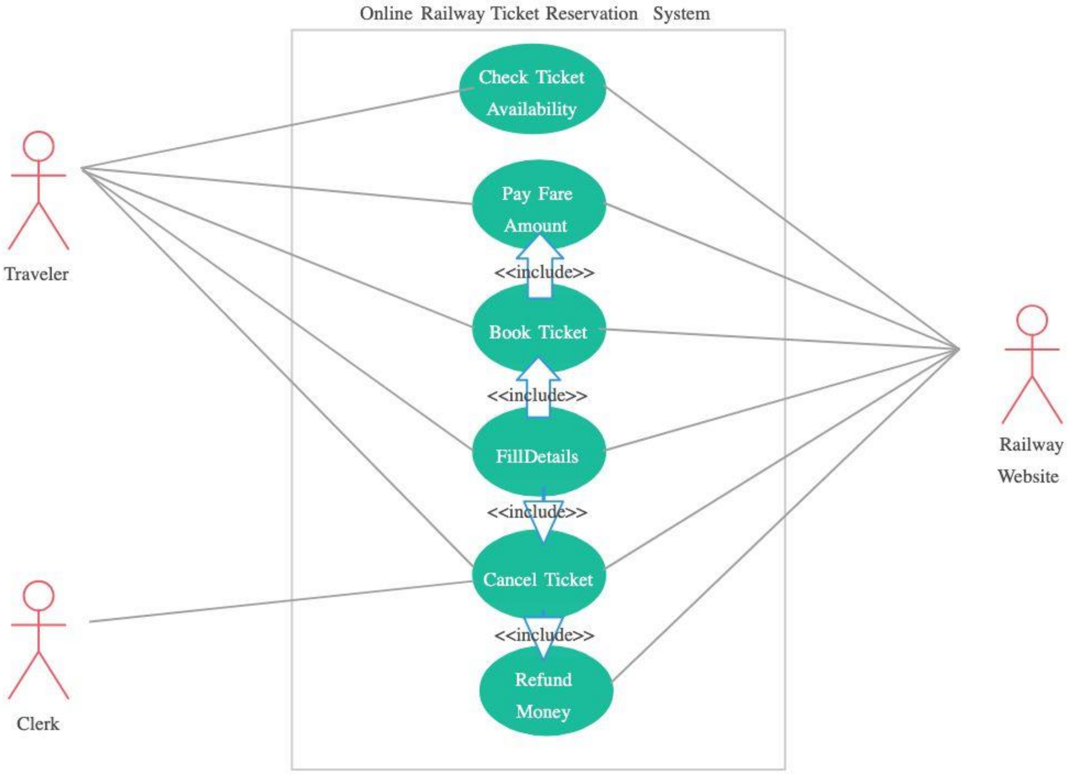
Actors: Represent people, other systems, or devices interacting with the system.Shown as stick figures, actors initiate or receive results from use cases.

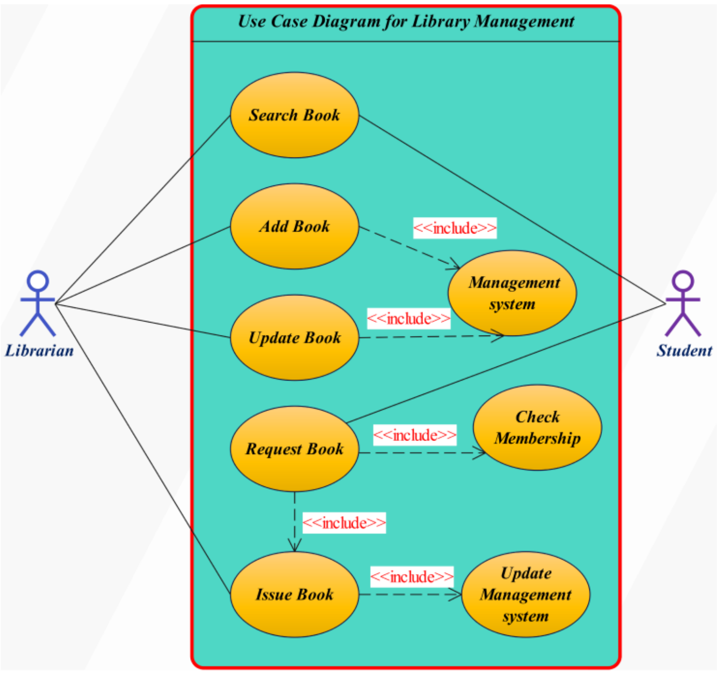
Use Cases: Represent actions or functions the system can perform, like "Login" or "Place Order."Depicted as ovals, use cases describe specific system functionalities.

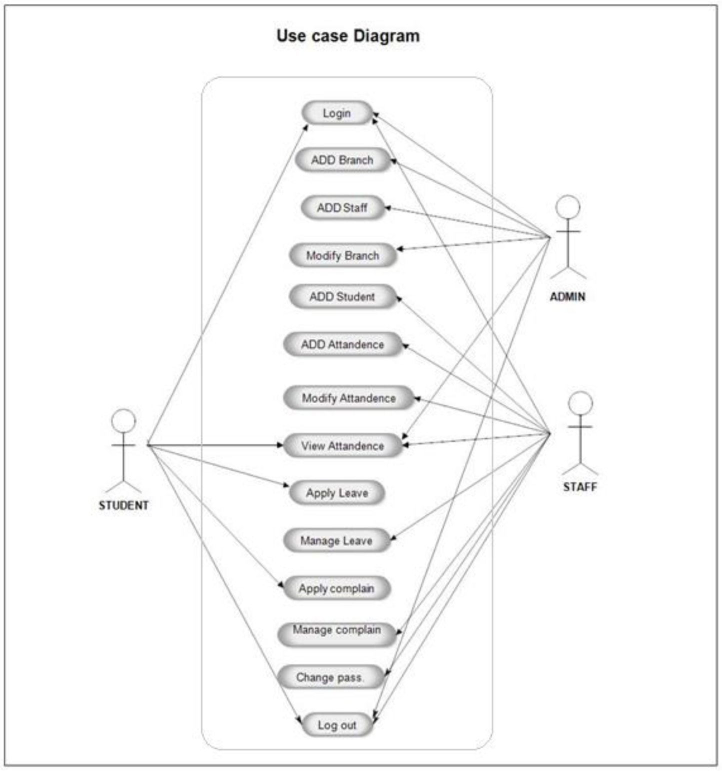
System Boundary: A box that surrounds all use cases, showing the scope of the system.Everything inside the box is part of the system, and everything outside is an external entity or actor.



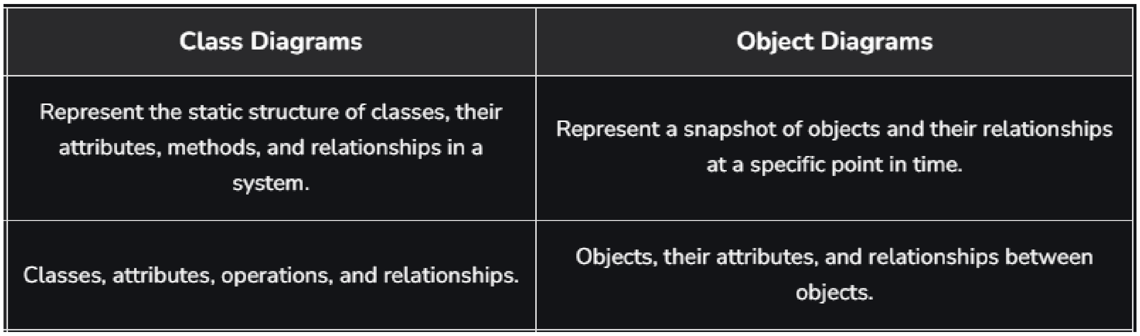

«subsystem»
Bank ATM
Check Balances
Deposit Funds
Withdraw Cash
Transfer Funds
Maintenance
Repair
Customer
ATM Technician
BANK
Bank
© uml-diagrams.org


Online Railway Ticket Reservation System
Check Ticket Availability
Pay Fare Amount
<<include>>
Book Ticket
<<include>>
FillDetails
<<include>>
Cancel Ticket
<<include>>
Refund Money
Traveler
Clerk
Railway Website


Use Case Diagram for Library Management
Search Book
Add Book
<<include>>
Management system
Update Book
<<include>>
Request Book
<<include>>
Check Membership
<<include>>
Issue Book
<<include>>
Update Management system
Librarian
Student


Use case Diagram
Login
ADD Branch
ADD Staff
Modify Branch
ADD Student
ADD Attandence
Modify Attandence
View Attandence
Apply Leave
Manage Leave
Apply complain
Manage complain
Change pass.
Log out
STUDENT
ADMIN
STAFF

To draw the structural view diagram: Class diagram, object diagram.



Class Diagrams	Object Diagrams
Represent the static structure of classes, their attributes, methods, and relationships in a system.	Represent a snapshot of objects and their relationships at a specific point in time.
Classes, attributes, operations, and relationships.	Objects, their attributes, and relationships between objects.

Class Diagrams: Class diagrams visually represent the structure and relationships of classes in a system, highlighting attributes and methods.

Object Diagrams: Object diagrams show specific instances of classes and their relationships at a particular point in time, providing a snapshot of the system's structure.

Class Diagram Components

1. **Classes**: Represented as boxes with three compartments:
    - **Class Name**: The name of the class.
    - **Attributes**: Properties or characteristics of the class (e.g., variables)**.**
    - **Methods**: Functions or operations the class can perform.
2. **Associations**: Lines connecting classes that indicate relationships between them (e.g., one-to-one, one-to-many).
3. **Multiplicity**: Notations on association lines that define how many instances of one class relate to another (e.g., 1, 0..*, 1..n).
4. **Generalization**: A line with a hollow triangle pointing to the parent class, showing inheritance relationships.
5. **Interfaces**: Represented as circles or rectangles with a «interface» label, indicating classes that implement specific contracts.
6. **Dependencies**: Dotted lines that indicate a class depends on another class, usually for method parameters or return types.

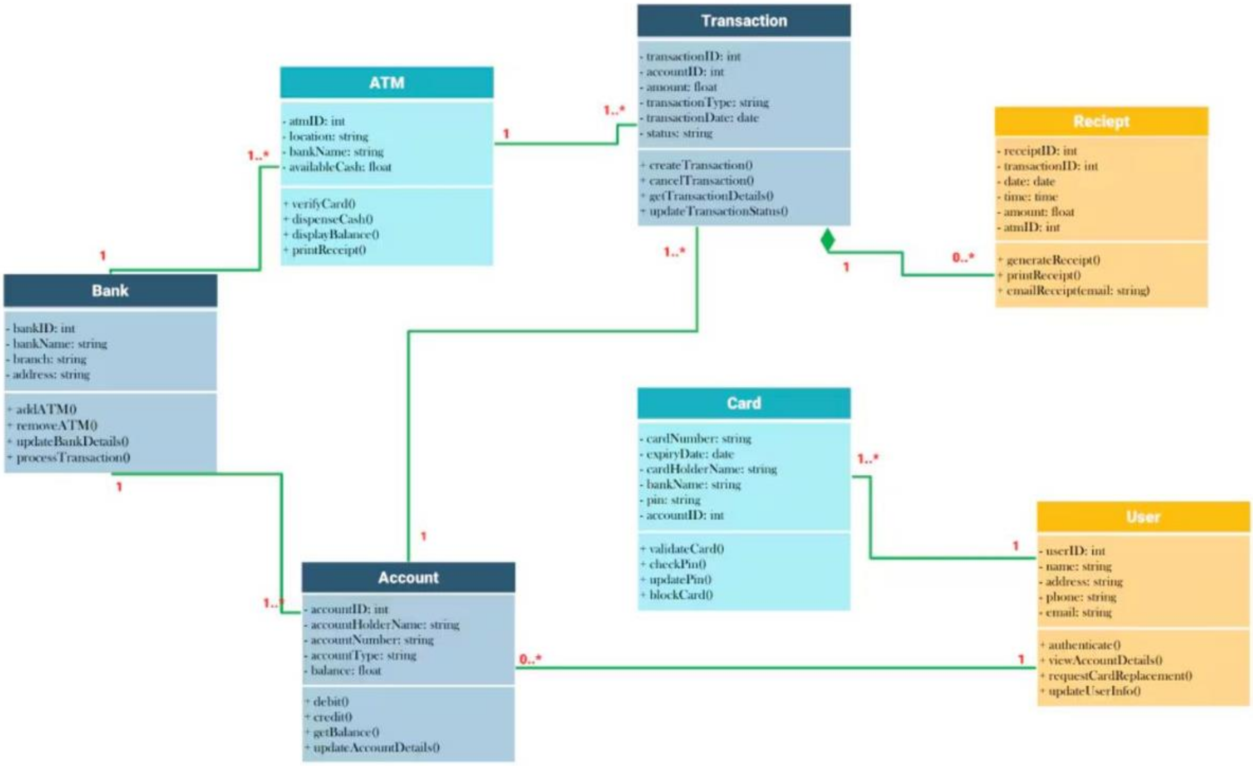
Object Diagram Components

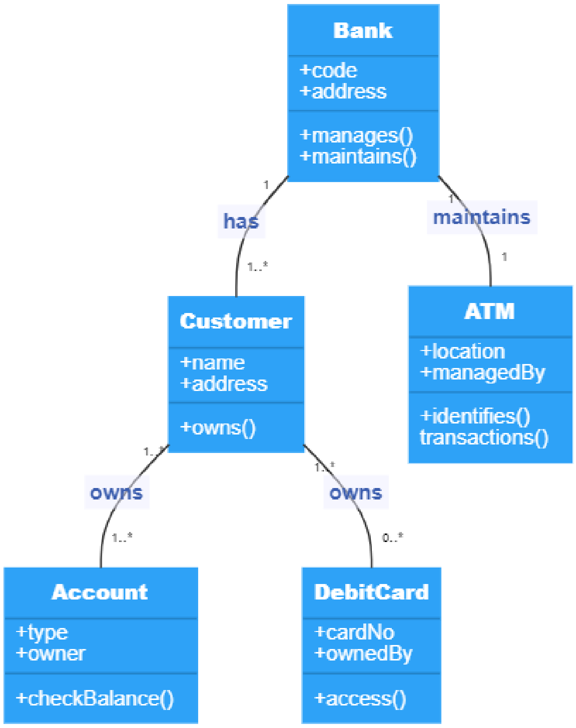
1. **Objects**: Represented as rectangles showing instances of classes, with the format:
    - **Object Name**: The name of the object.
    - **Class Name**: The name of the class it belongs to (often underlined).
2. **Attributes**: Listed within the object box, showing specific values for the instance's attributes (e.g., name: "John", age: 30).

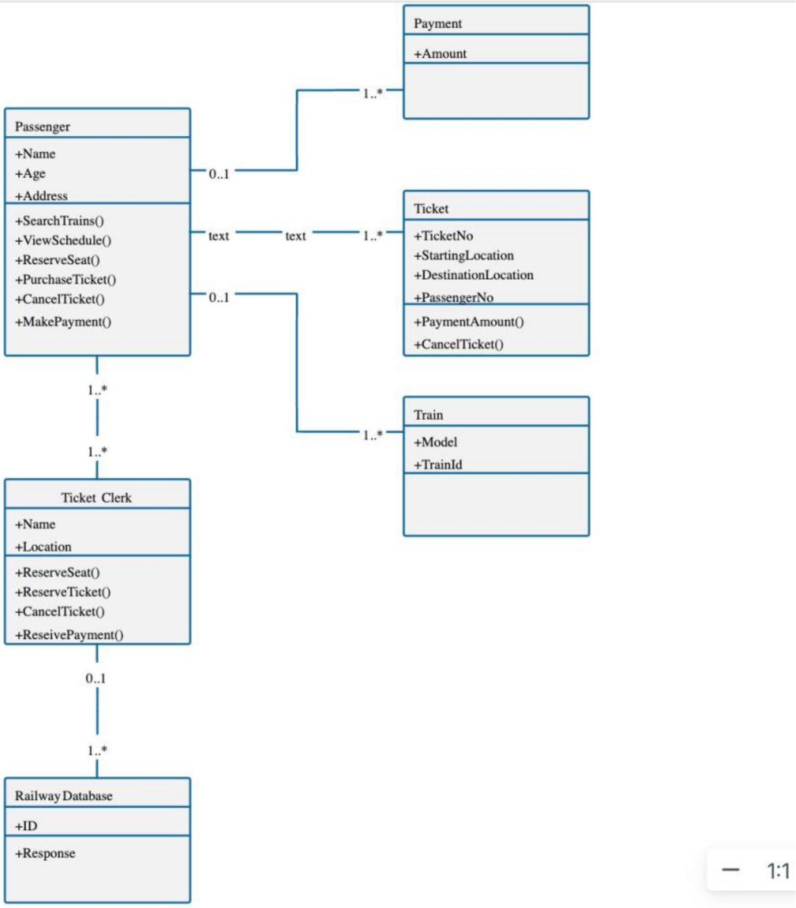
3. **Links**: Lines connecting objects that represent their relationships or associations, similar to class associations.

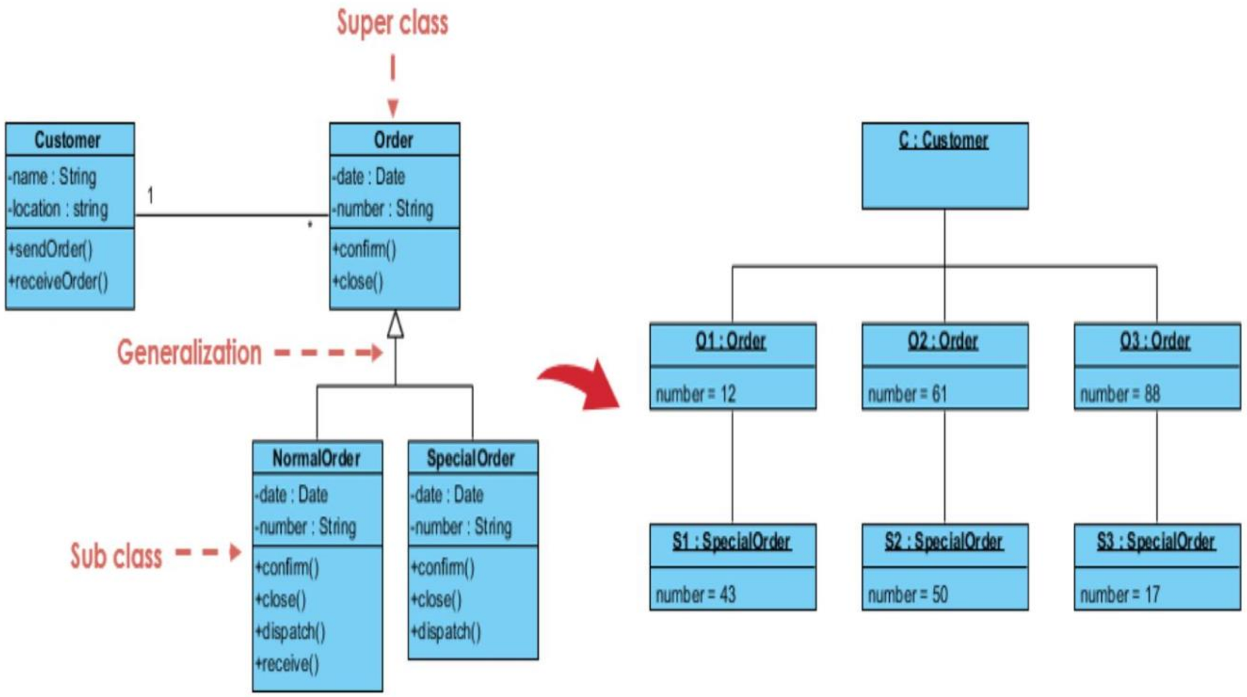
4. **Multiplicity**: Notations on links that indicate the number of instances involved in the relationship.

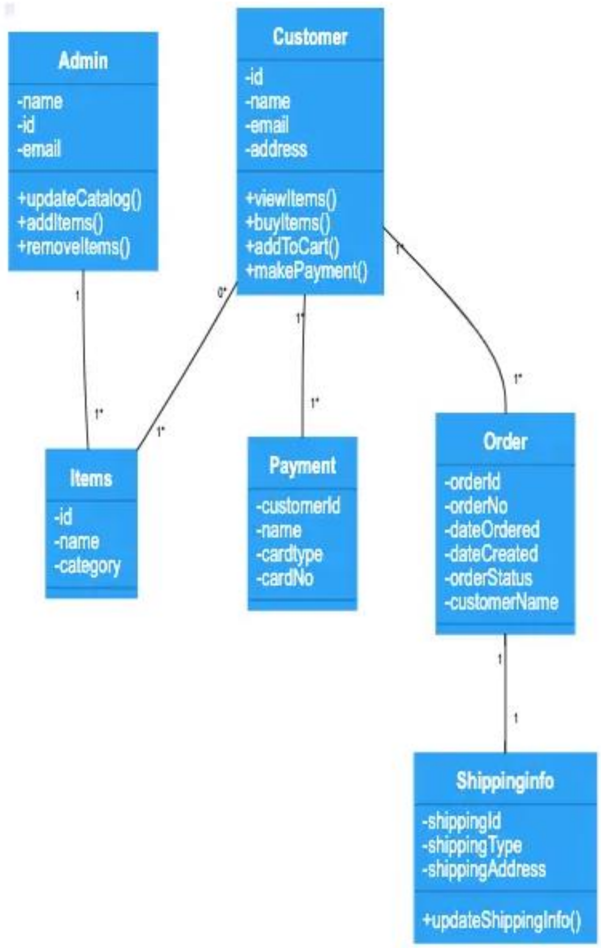
5. **State**: Indicates the current state of an object, often shown next to the object name.




Bank
+code
+address
+manages()
+maintains()
1
has
1..*
1
maintains
1
Customer
+name
+address
+owns()
ATM
+location
+managedBy
+identifies()
transactions()
1..*
owns
1..*
1..*
owns
0..*
Account
+type
+owner
+checkBalance()
DebitCard
+cardNo
+ownedBy
+access()


Payment
+Amount
1..*
0..1
Passenger
+Name
+Age
+Address
+SearchTrains()
+ViewSchedule()
+ReserveSeat()
+PurchaseTicket()
+CancelTicket()
+MakePayment()
text
text
1..*
Ticket
+TicketNo
+StartingLocation
+DestinationLocation
+PassengerNo
+PaymentAmount()
+CancelTicket()
0..1
1..*
Train
+Model
+TrainId
1..*
1..*
Ticket Clerk
+Name
+Location
+ReserveSeat()
+ReserveTicket()
+CancelTicket()
+ReseivePayment()
0..1
1..*
Railway Database
+ID
+Response
1:1


Super class
Customer
-name : String
-location : string
+sendOrder()
+receiveOrder()
1
*
Order
-date : Date
-number : String
+confirm()
+close()
Generalization
NormalOrder
-date : Date
-number : String
+confirm()
+close()
+dispatch()
+receive()
SpecialOrder
-date : Date
-number : String
+confirm()
+close()
+dispatch()
Sub class
C : Customer
O1 : Order
number = 12
O2 : Order
number = 61
O3 : Order
number = 88
S1 : SpecialOrder
number = 43
S2 : SpecialOrder
number = 50
S3 : SpecialOrder
number = 17


Admin
-name
-id
-email
+updateCatalog()
+addItems()
+removeItems()
Customer
-id
-name
-email
-address
+viewItems()
+buyItems()
+addToCart()
+makePayment()
1
1*
0*
1*
1*
1*
1*
1*
Items
-id
-name
-category
Payment
-customerId
-name
-cardtype
-cardNo
Order
-orderId
-orderNo
-dateOrdered
-dateCreated
-orderStatus
-customerName
1
1
ShippingInfo
-shippingId
-shippingType
-shippingAddress
+updateShippingInfo()

To draw the behavioral view diagram: Sequence diagram, Collaboration diagram

A **Sequence Diagram** shows how objects communicate over time in a specific scenario.

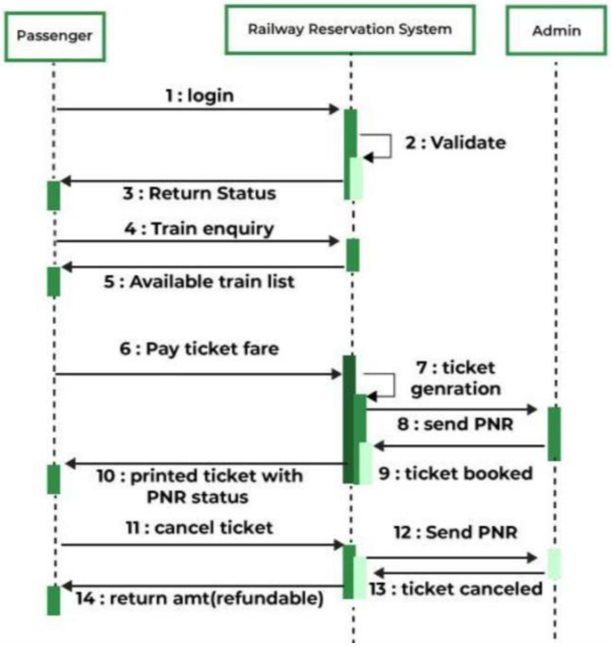
Components:

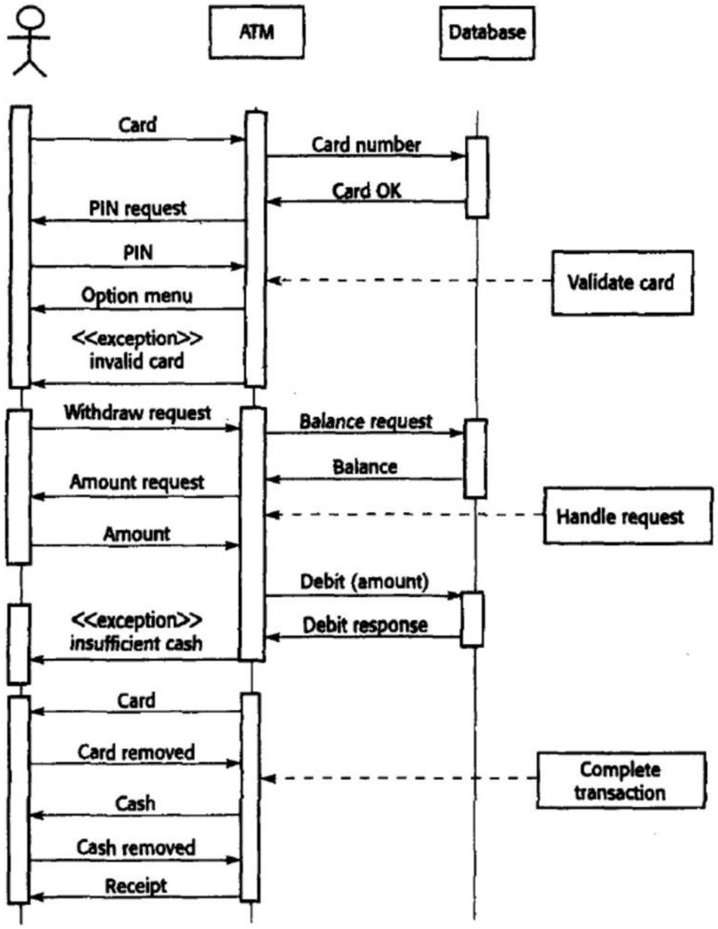
1. **Lifelines**: Vertical dashed lines for each object or actor, with a rectangle at the top for the object's name.
2. **Messages**: Horizontal arrows between lifelines showing the communication flow, with the message name above the arrow.
3. **Activation Bars**: Rectangles on lifelines indicating when an object is active in the process.
4. **Return Messages**: Dashed arrows that show responses from one object to another.
5. **Notes**: Extra comments that provide additional context.

A **Collaboration Diagram** (or Communication Diagram) illustrates how objects interact to achieve a goal, focusing on their relationships.

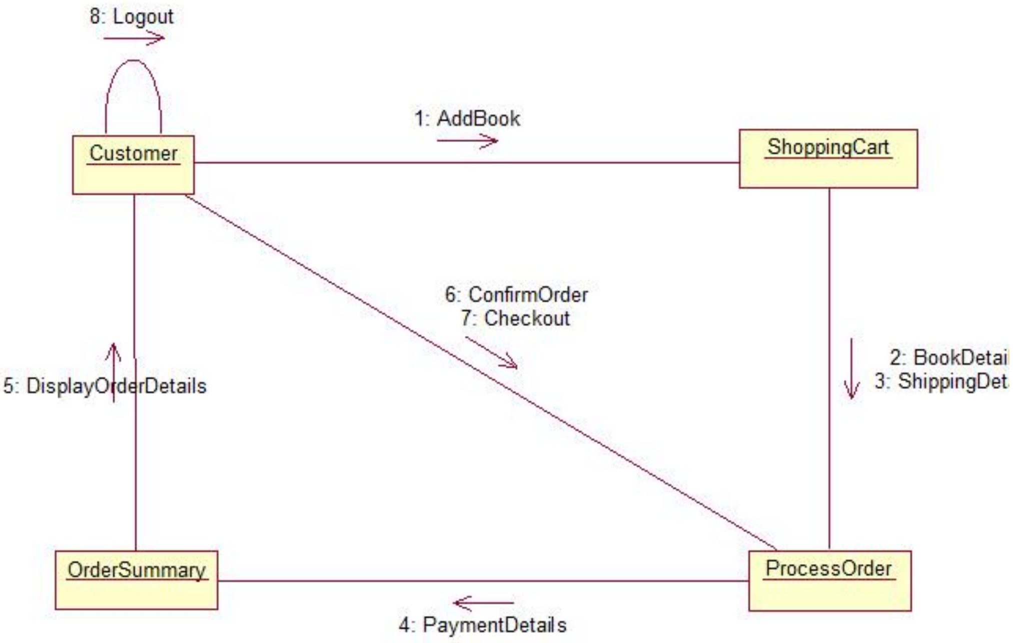
Components:

1. **Objects**: Labeled boxes representing the objects involved in the interaction.
2. **Links**: Lines connecting the objects to show their relationships.
3. **Messages**: Numbered arrows or labels next to the links that indicate the order of messages exchanged.
4. **Roles**: Labels for the roles of the objects involved in the interaction.
5. **Notes**: Additional comments for clarification.

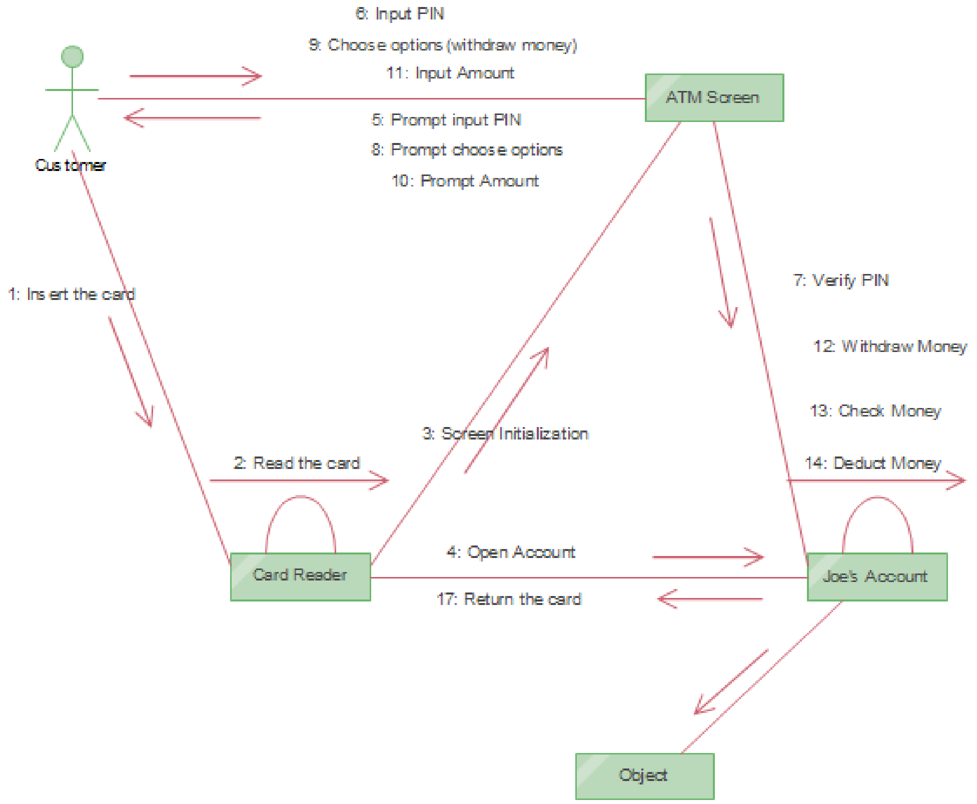


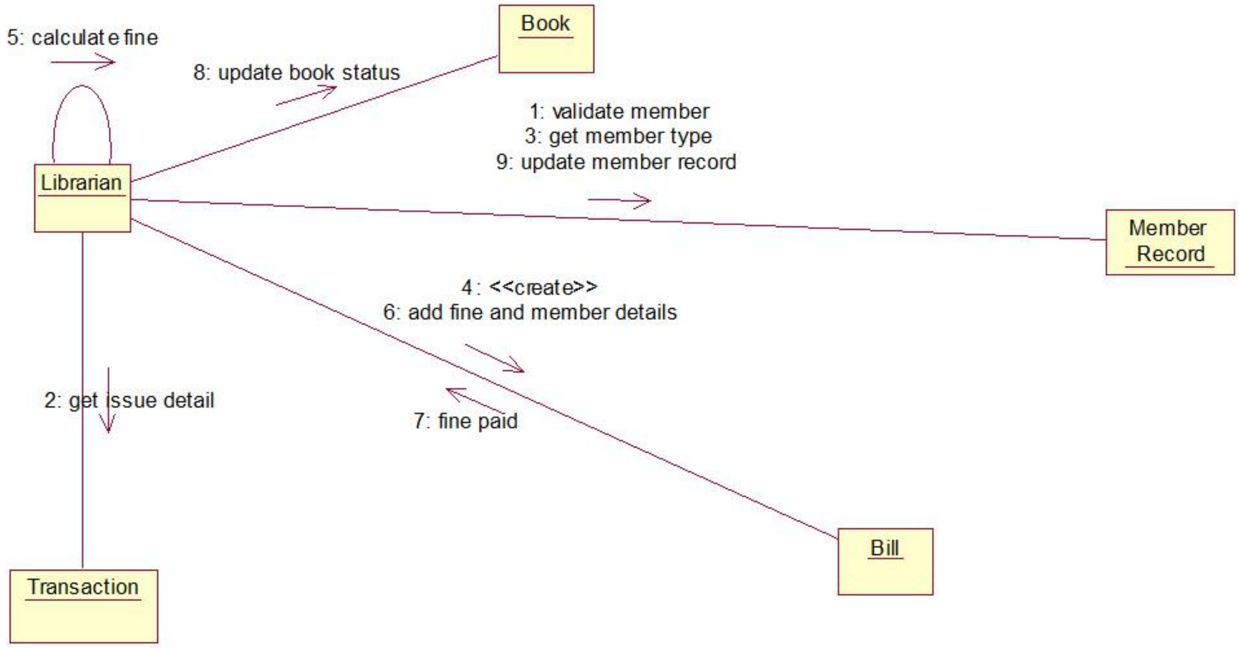


Online shopping – collaboration diag



ATM UML Collaboration Diagram

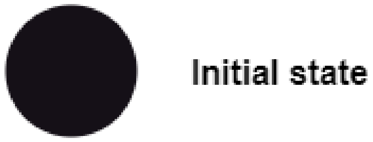


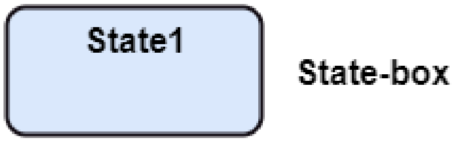


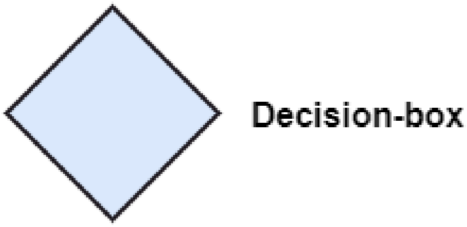
To draw the behavioral view diagram: State-chart diagram

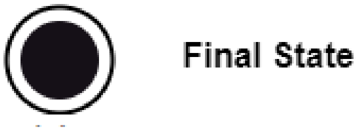
A **State-Chart Diagram** shows the different states an object can be in throughout its life cycle and how it transitions from one state to another in response to events.

1. **States**: Represented as rounded rectangles or ovals. Each state describes a specific condition or situation of the object.
2. **Initial State**: Shown as a filled black circle, indicating where the state transitions begin.
3. **Final State**: Represented as a circle surrounding a filled black circle, indicating where the state transitions end.
4. **Transitions**: Arrows connecting states that indicate how an object moves from one state to another. Each arrow can be labeled with the event that triggers the transition.
5. **Events**: Actions or occurrences that cause a transition between states. They can be external (like user actions) or internal (like time events).
6. **Conditions**: Optional guards placed on transitions, expressed in brackets (e.g., [condition]) to specify when the transition is valid.
7. **Actions**: Optional activities that occur during a transition, usually written next to the transition arrow.

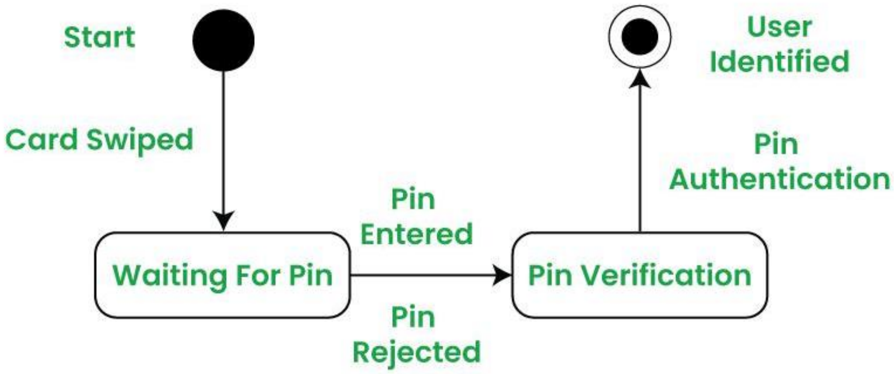






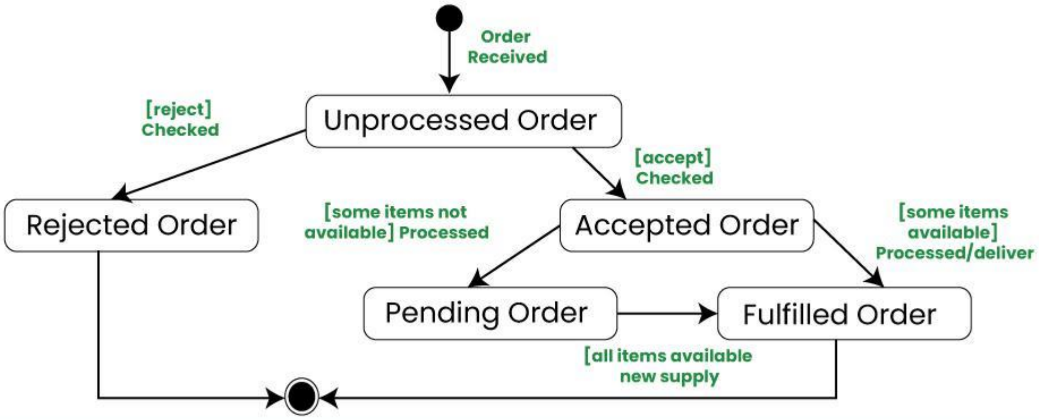


A State Machine Diagram for user verification

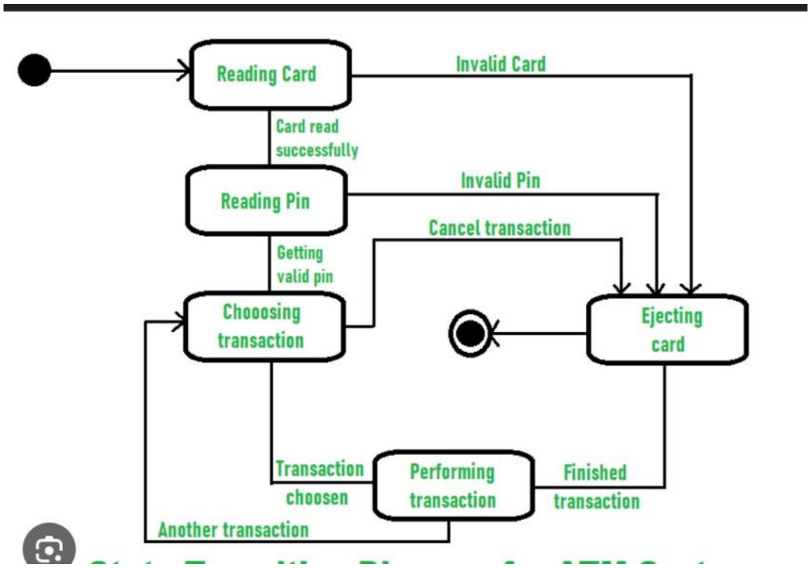


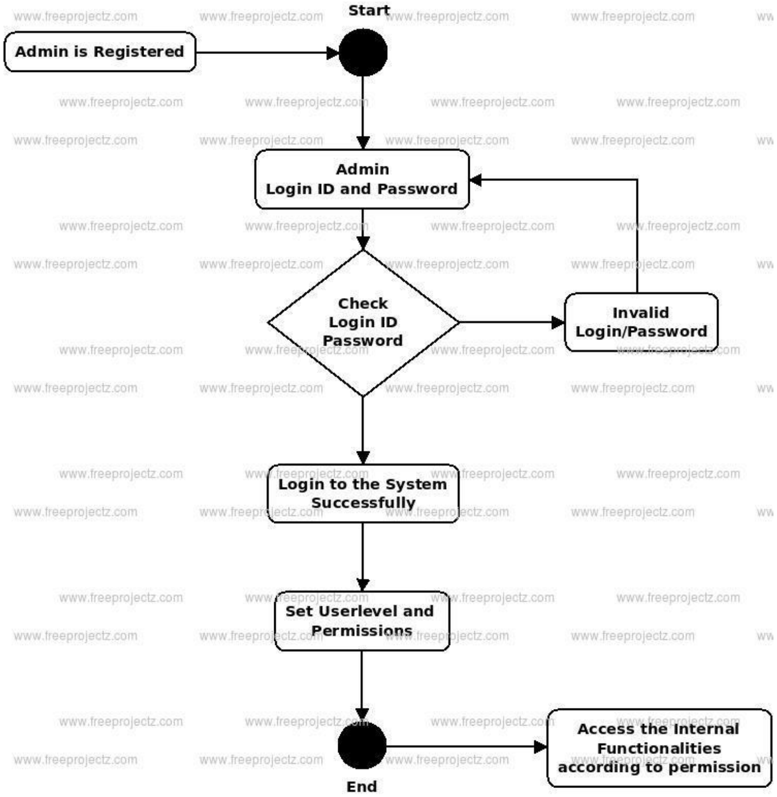
State Machine Diagrams | Unified Modeling Language (UML)

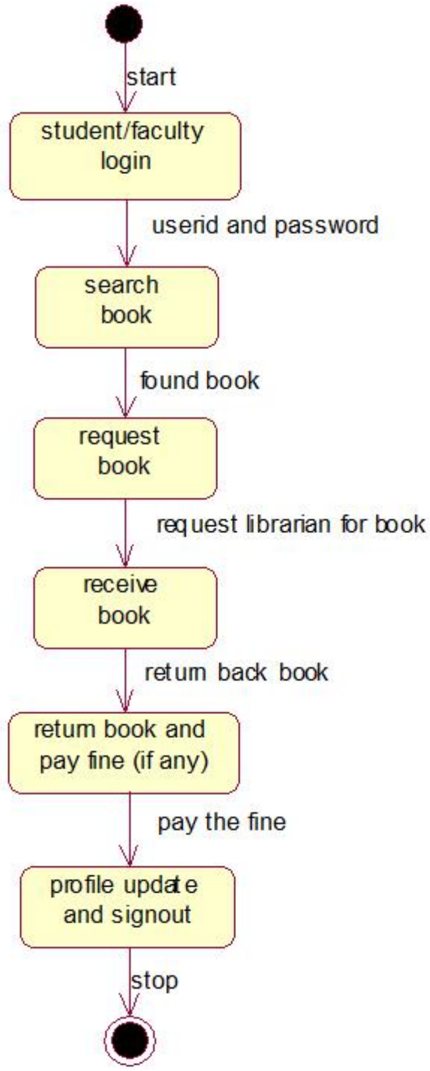
State machine diagram for an online order



State Machine Diagrams | Unified Modeling Language (UML)

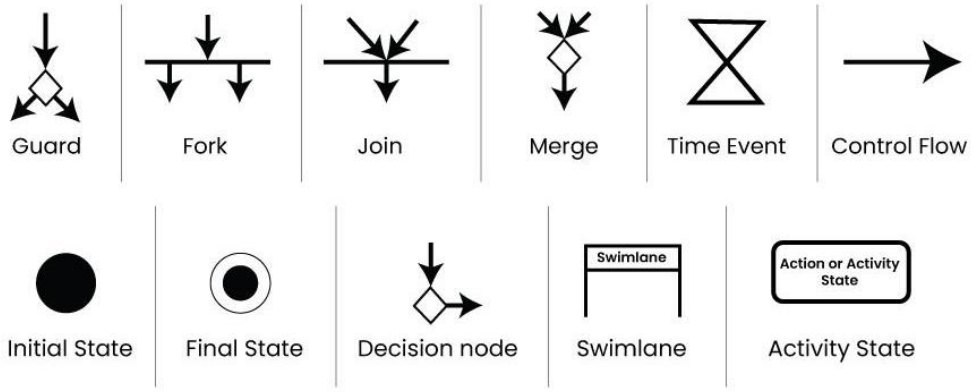

Reading Card
Invalid Card
Card read successfully
Reading Pin
Invalid Pin
Cancel transaction
Getting valid pin
Chooosing transaction
Ejecting card
Transaction choosen
Performing transaction
Finished transaction
Another transaction


Start
Admin is Registered
Admin Login ID and Password
Check Login ID Password
Invalid Login/Password
Login to the System Successfully
Set Userlevel and Permissions
End
Access the Internal Functionalities according to permission

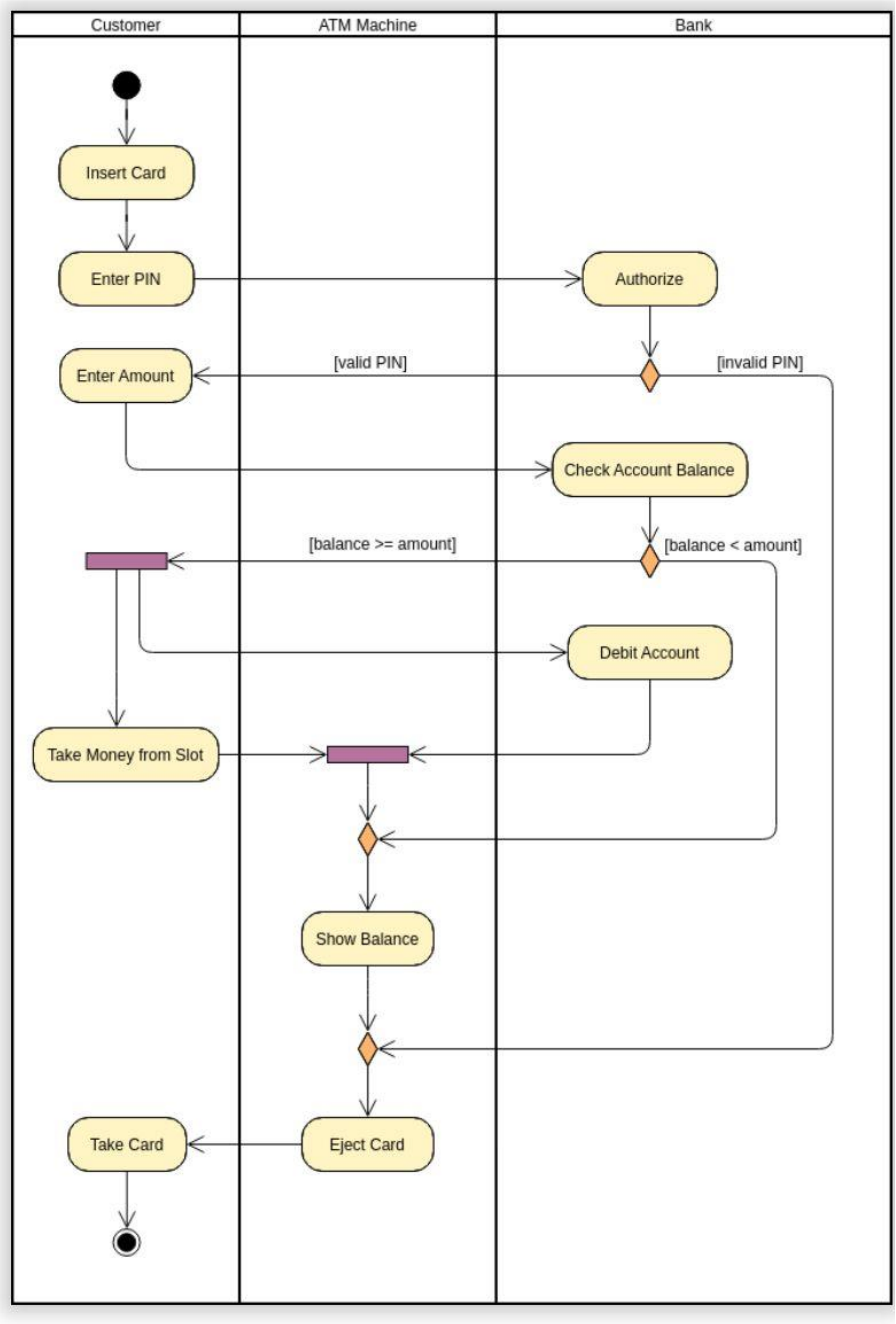

start
student/faculty login
userid and password
search book
found book
request book
request librarian for book
receive book
return back book
return book and pay fine (if any)
pay the fine
profile update and signout
stop

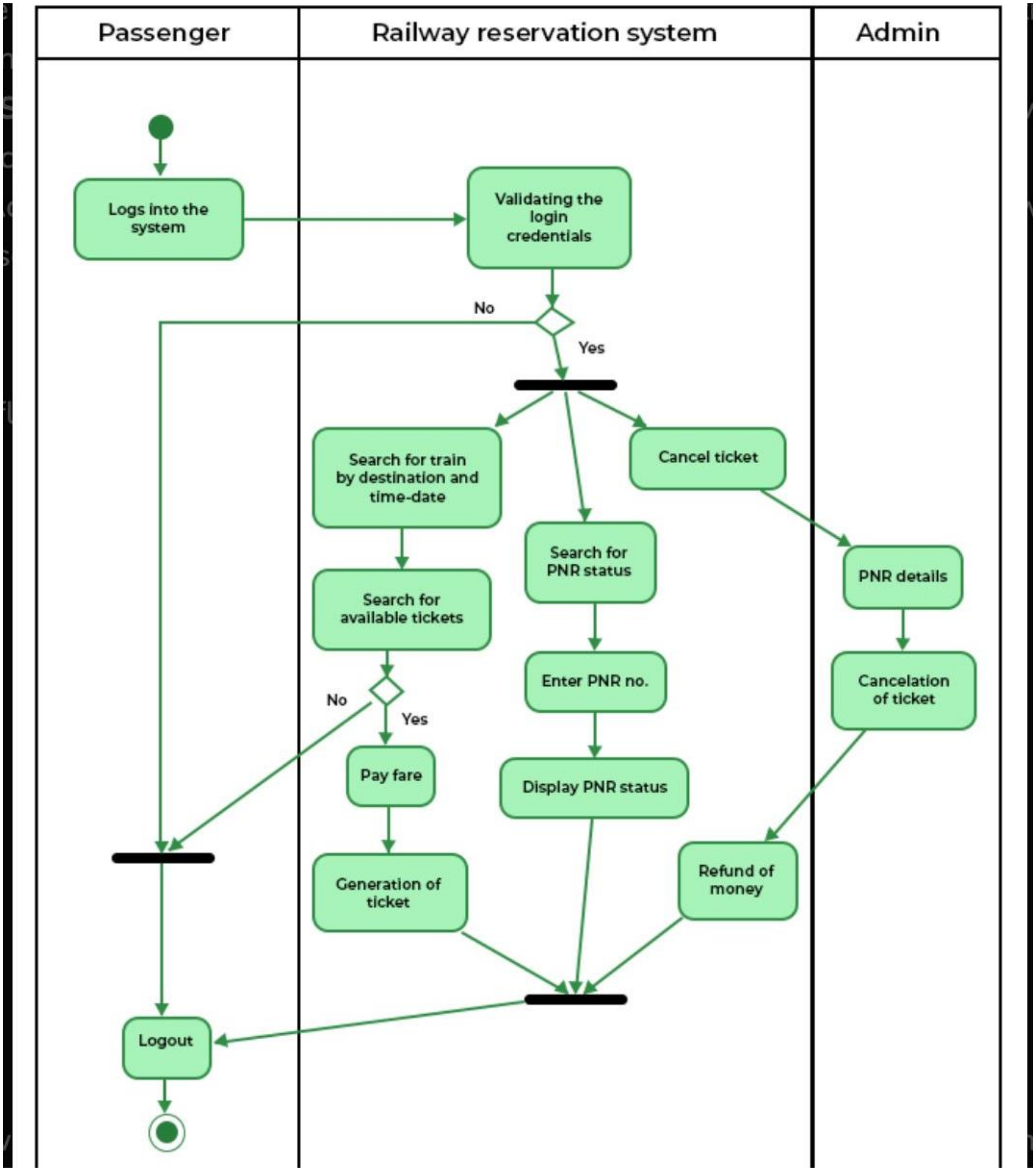
To draw the behavioral view diagram: Activity diagram

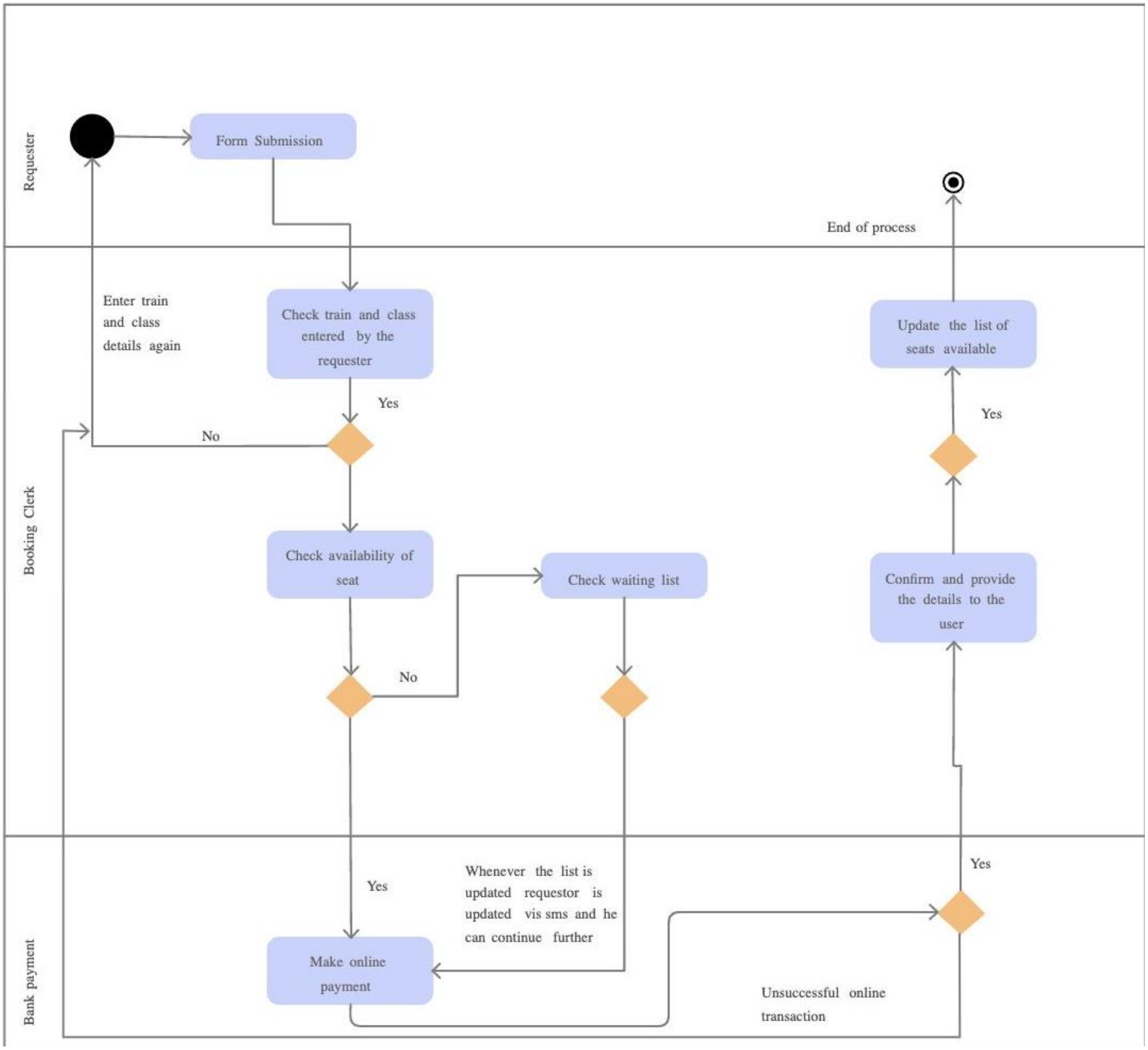
An **Activity Diagram** shows the workflow of activities and actions in a system. It visually represents the sequence of actions and decisions involved in a process, helping to model dynamic aspects of a system.



1. **Initial State**: Represents the starting point of a process, shown as a filled black circle.
2. **Action or Activity State**: Depicts the execution of an action, represented by a rectangle with rounded corners.
3. **Action Flow or Control Flows**: Arrows indicating the transition between activity states, showing the flow of control.
4. **Decision Node and Branching**: A diamond that represents a point where a decision is made, leading to different paths based on conditions.
5. **Guard**: A condition written on an arrow near a decision node that must be true for the flow to follow that path.
6. **Fork**: A node that splits an activity into two or more concurrent actions, represented by a solid rectangular bar.
7. **Join**: A node that converges multiple incoming activities into one outgoing activity, indicated by two or more incoming arrows and one outgoing arrow.
8. **Merge or Merge Event**: A node that combines multiple paths into one, allowing control to proceed regardless of the path taken.


Customer
ATM Machine
Bank
Insert Card
Enter PIN
Authorize
[valid PIN]
[invalid PIN]
Enter Amount
Check Account Balance
[balance >= amount]
[balance < amount]
Debit Account
Take Money from Slot
Show Balance
Take Card
Eject Card


Passenger
Railway reservation system
Admin
Logs into the system
Validating the login credentials
No
Yes
Search for train by destination and time-date
Search for PNR status
Cancel ticket
PNR details
Search for available tickets
Enter PNR no.
Cancelation of ticket
No
Yes
Pay fare
Display PNR status
Refund of money
Generation of ticket
Logout


Requester
Booking Clerk
Bank payment
Form Submission
Check train and class entered by the requester
Enter train and class details again
No
Yes
Check availability of seat
No
Check waiting list
Yes
Whenever the list is updated requestor is updated vis sms and he can continue further
Make online payment
Unsuccessful online transaction
Yes
Confirm and provide the details to the user
Yes
Update the list of seats available
End of process

To draw the implementation view diagram: Component diagram.

A **Component Diagram** shows the physical components of a system, including the relationships and dependencies between them. It helps visualize how different parts of a system work together and how they are connected.

Component: Modular parts encapsulating specific functionalities, represented by rectangles with the stereotype «component».

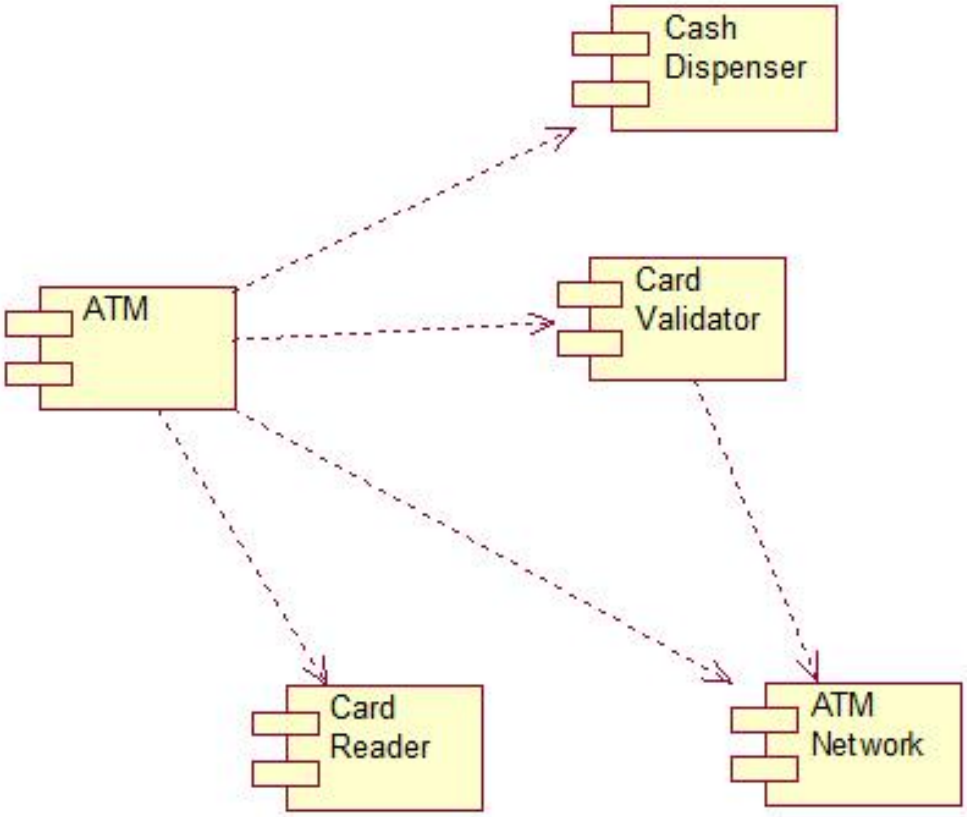
Interfaces: Operations offered or required by components, shown as circles (provided) and half-circles (required).

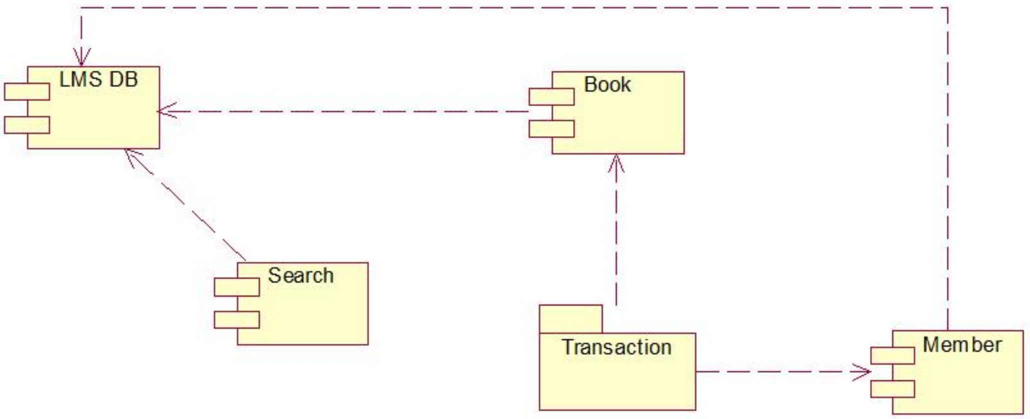
Relationships: Connections between components, depicted with lines; dashed arrows for dependencies and solid lines for associations.

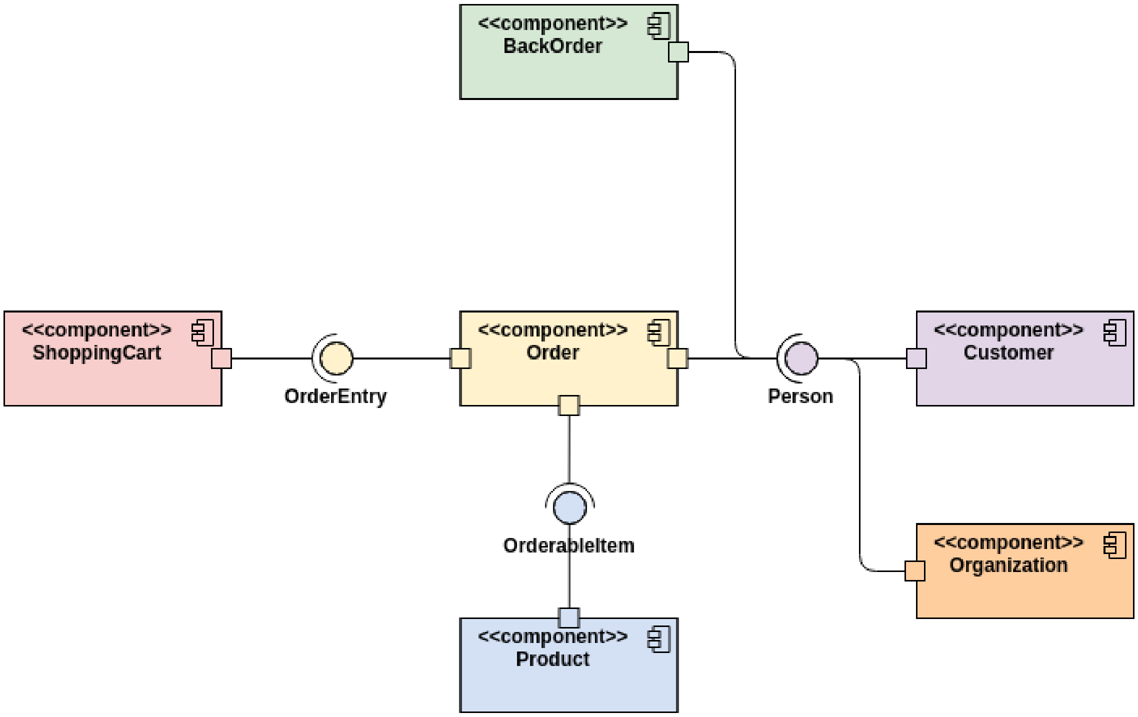
Ports: Interaction points on a component's boundary, represented by small squares.

Artifacts: Physical files or data related to components, shown as rectangles with the stereotype «artifact».

Nodes: Physical or virtual environments where components are deployed, depicted as 3D boxes

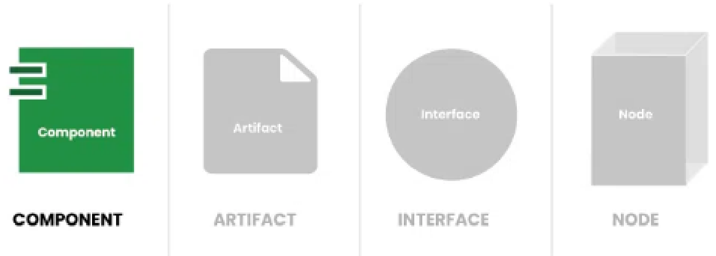






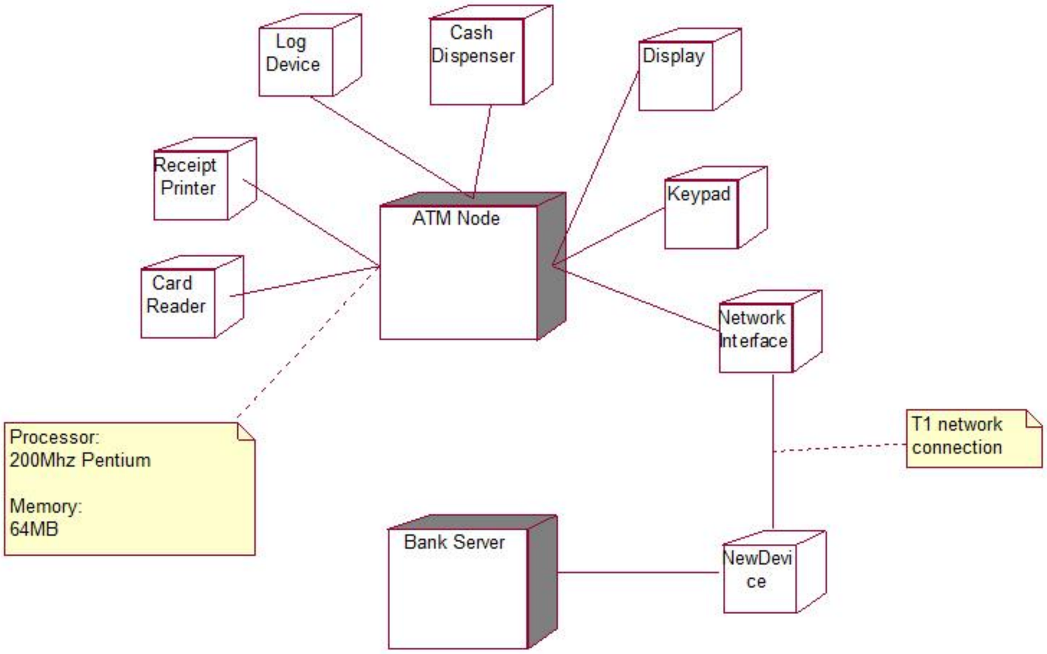
To draw the implementation view diagram: deployment diagram

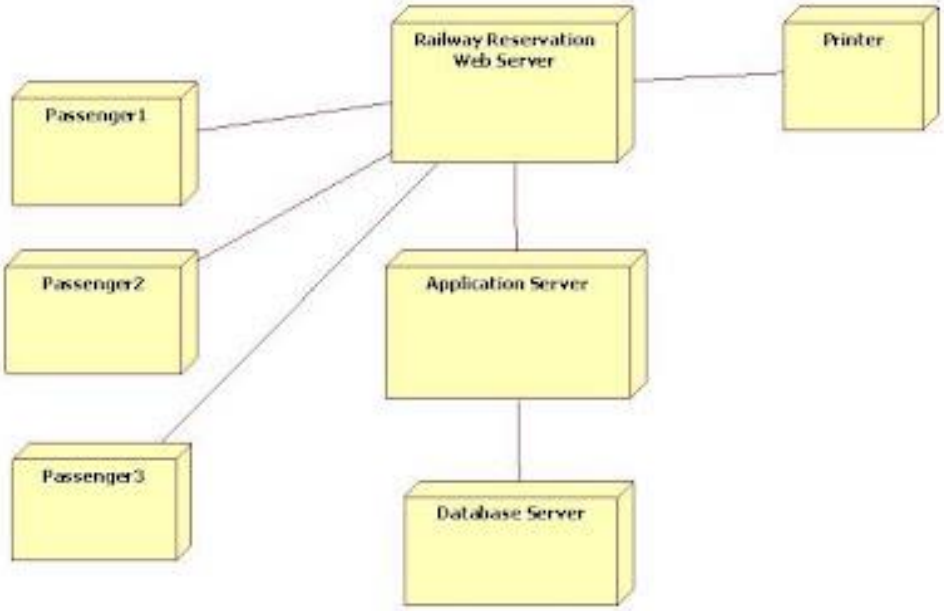
Component | Unified Modeling Language(UML)

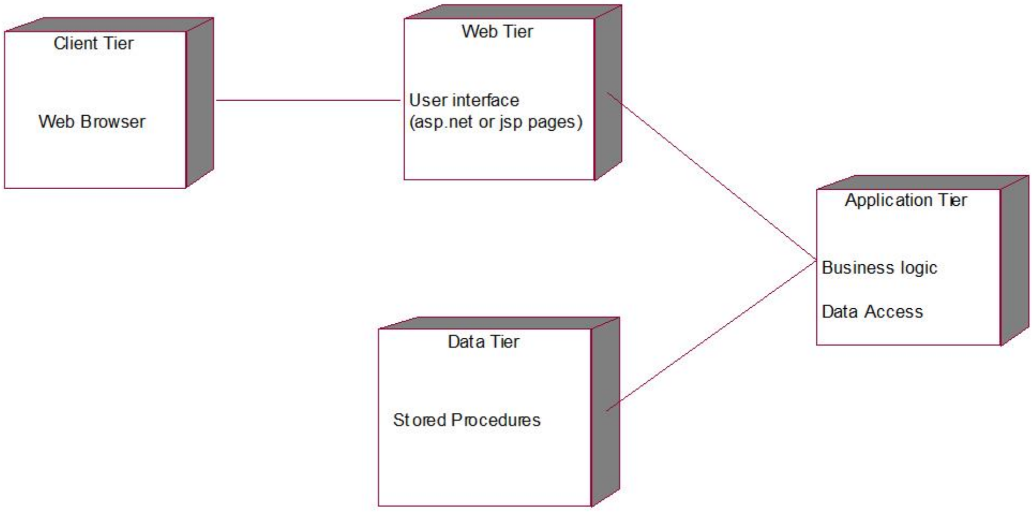


A **Deployment Diagram** shows the physical arrangement of software artifacts on hardware nodes, illustrating how components of a system are distributed across the hardware.

1. **Communication Paths**: Lines connecting nodes, showing network links and protocols.
2. **Artifacts**: Software elements (rectangles with «artifact») that reside on nodes.
3. **Nodes**: Physical or virtual devices (3D boxes) where components are deployed.
4. **Associations**: Lines indicating which artifacts are deployed on which nodes.






Client Tier
Web Browser
Web Tier
User interface
(asp.net or jsp pages)
Application Tier
Business logic
Data Access
Data Tier
Stored Procedures