**GURU TEGH BAHADUR INSTITUTE OF TECHNOLOGY**



**SOFTWARE ENGINEERING PRACTICAL FILE**

**SUBJECT CODE: CIC-357**

**SEMESTER – V**

**Submitted to, Submitted by,**

**Dr. Gurpreet Kaur Name: Abhay Raj**

**Enrolment No: 00976803122**

**Branch: IT – 3**

**Batch: 2022-2026**

**LIST OF EXPERIMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| S. No | Experiment | Experiment Date | Sign |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**EXPERIMENT – 1A**

**AIM:-**

To Write PROBLEM STATEMENT for Library Management System

**REQUIREMENT:-**

Document Editor (MS Word, Libera Writer, etc.).

**THEORY:-**

Library Management System: A Library Management System (LMS) is designed to streamline and automate the processes involved in managing a library. Its primary goal is to facilitate efficient management of library resources, services, and user interactions. An LMS helps librarians and staff efficiently handle book cataloguing, circulation, user accounts, and other related tasks, ultimately enhancing the overall user experience and operational efficiency of the library.

PROBLEM STATEMENT:

A student enters the library. To enter the library he should have his college Id card or library card with him. To issue a book he has to go to the clerk. There he show his library card to get his book issued. He can avail multiple books for a set duration. Within that duration he has to return it. If not returned, an amount will be fined per day until returned. A student can avail facility like magazines, news paper, computer system, PYQ papers. A student can also issue books for a given semester. He can get books based on the curriculum for the time period of the entire semester. After that he has to return the issued books. A student will be making the library card after his admission to the institution. To make the library card he has to fill a form provided in the library and submit it. In a week’s time library card will be issued for the student. The librarian do the ordering of books from publishers, accounting of books, organizing book displays. Librarian also manage budgeting, staff management, policy development and ensuring smooth operations of the library and meets its goals. The procurement of the books is done with publishers, distributors or book vendors. Before the procurement, the books are selected which are to be included in the library. This is done based on the curriculum, faculty needs, students interest and emerging academic trends. After this the librarian allocate funds for the book purchase and then the procurement is done. The library staff ensures discipline is maintain in the library. They also help students to find books in the library

**CONCLUSION:-**

The PROBLEM STATEMENT is Written for Library Management System.

**EXPERIMENT – 1B**

**AIM:-**

To Write PROBLEM STATEMENT for Hotel Management System

**REQUIREMENT:-**

Document Editor (MS Word, Libera Writer, etc.).

**THEORY:-**

Hotel Management System: A Hotel Management System (HMS) is designed to streamline and automate the various processes involved in running a hotel. Its primary goal is to facilitate efficient management of hotel operations, including reservations, guest services, billing, and resource management. An HMS helps hotel staff provide a smooth and seamless experience for guests, improving operational efficiency and enhancing customer satisfaction.

PROBLEM STATEMENT:

A guest arrives at the hotel and is greeted with refreshments before checking in using a reservation made through the hotel’s online booking platform or at the front desk. The receptionist manages the booking process, including handling guest details, room availability, and payment information, and assigns a room based on the guest's preferences and booking status. During their stay, guests can enjoy various amenities: the restaurant, managed by the chef, offers a range of dining options, while room service delivers food and beverages directly to the guest’s room. The cleaner, or housekeeping staff, ensures rooms are kept clean and well-maintained, addressing any special requests from the guest. Additionally, the hotel provides a convenient car service for pickups and drop-offs and ample parking for guests. A gardener maintains the hotel’s lawns and surrounding outdoor areas, ensuring a pleasant and well-kept environment. The hotel also has a dedicated building for staff accommodations, providing on-site housing for employees. The manager oversees daily operations, coordinating between the front desk, restaurant, room service, housekeeping, car service, and gardening to ensure smooth and efficient service. They are also responsible for staff scheduling, inventory management, and financial reporting. At check-out, the receptionist processes the final bill, including charges for room service, dining, car service, and any other amenities used.

**CONCLUSION:-**

The PROBLEM STATEMENT is Written for Hotel Management System.

**EXPERIMENT – 2A**

**AIM:-**

To Prepare FLOW CHART for Library Management System

**REQUIREMENT:-**

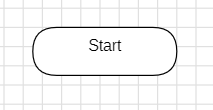
StarUML.

**THEORY:-**

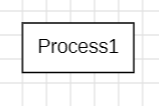
A flowchart is a visual representation of a process, workflow, or system, using standardized symbols and shapes to depict steps and decisions. Flowcharts are used to illustrate how different elements of a process interact and how data flows from one step to another. The primary purpose of a flowchart is to provide a clear and easy-to-understand depiction of complex processes, making it easier to analyse, design, and communicate workflows.

Basic Components of a Flowchart

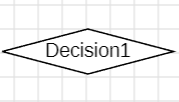
* **Start/End**: Represented by oval shapes, these symbols indicate the beginning and end points of the process.



* **Process**: Depicted by rectangular shapes, this symbol represents a step or action in the process, such as a task or operation.



* **Decision**: Shown as diamond shapes, these symbols represent decision points where the process can branch based on a yes/no or true/false condition.

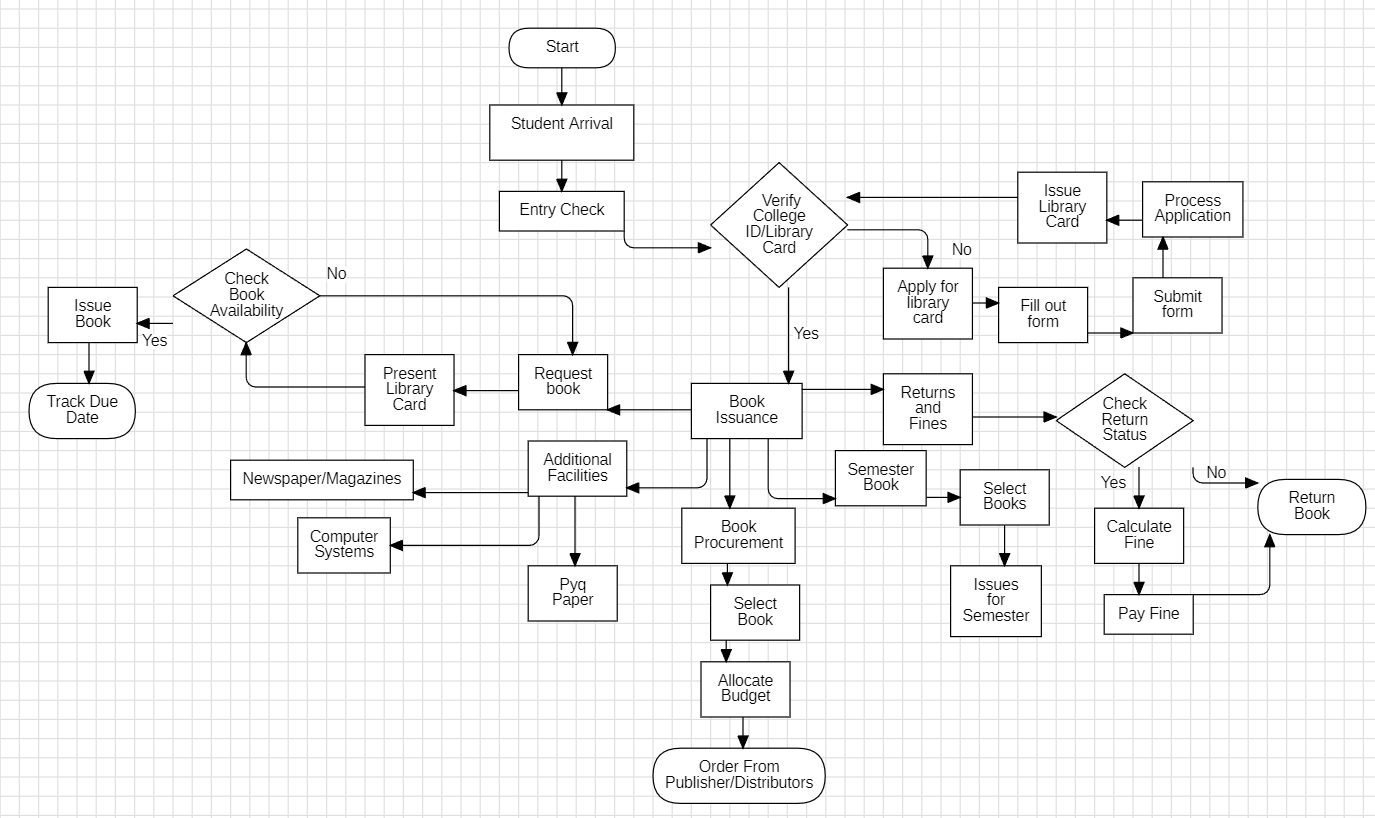


* **Flow Lines**: Arrows connecting the symbols, showing the direction of the process flow and the sequence of steps.



**FLOW CHART:-**

Library Management System:



**CONCLUSION:-**

The FLOW CHART is Made for Library Management System.

**EXPERIMENT – 2B**

**AIM:-**

To Prepare FLOW CHART for Hotel Management System

**REQUIREMENT:-**

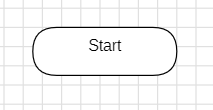
StarUML.

**THEORY:-**

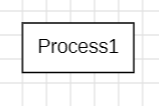
A flowchart is a visual representation of a process, workflow, or system, using standardized symbols and shapes to depict steps and decisions. Flowcharts are used to illustrate how different elements of a process interact and how data flows from one step to another. The primary purpose of a flowchart is to provide a clear and easy-to-understand depiction of complex processes, making it easier to analyse, design, and communicate workflows.

Basic Components of a Flowchart

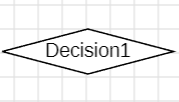
* **Start/End**: Represented by oval shapes, these symbols indicate the beginning and end points of the process.



* **Process**: Depicted by rectangular shapes, this symbol represents a step or action in the process, such as a task or operation.



* **Decision**: Shown as diamond shapes, these symbols represent decision points where the process can branch based on a yes/no or true/false condition.

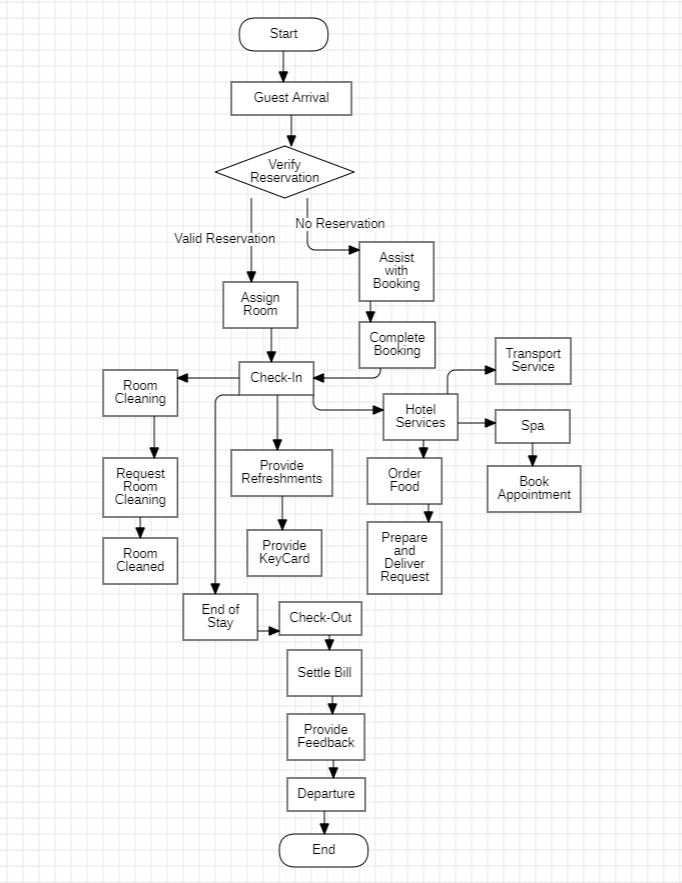


* **Flow Lines**: Arrows connecting the symbols, showing the direction of the process flow and the sequence of steps.



**FLOW CHART:-**

Hotel Management System:



**CONCLUSION:-**

The FLOW CHART is Made for Hotel Management System.

**EXPERIMENT – 3A**

**AIM:-**

To Prepare ER Diagram for Library Management System

**REQUIREMENT:-**

StarUML.

**THEORY:-**

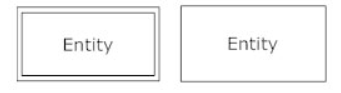
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a

database. An entity in this context is an object, a component of data. An entity set is a

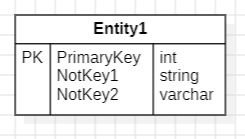
Collection of similar entities. These entities can have attributes that define its properties.

By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. ER diagrams are used to sketch out the design of a database. An ER diagram is a means of visualizing how the information a system produces is related.

* **Entity**: Represents a real-world object or concept that is distinguishable from other objects. Entities are usually represented as rectangles in an ER diagram. Each entity has a name and may have attributes associated with it. A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone



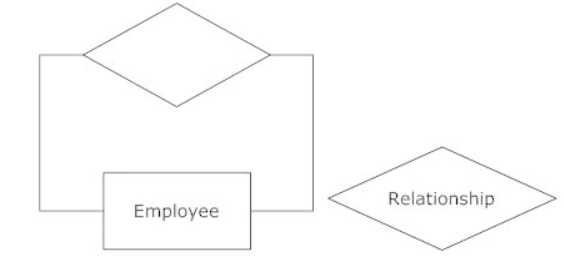
Entity In Star UML with attributes.



* **Attribute**: Describes the properties or characteristics of an entity. Attributes are represented by ovals connected to their respective entities. There are different types of attributes:

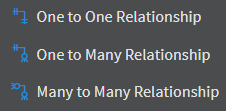


* **Relationship**: Represents the associations between entities. Relationships are depicted as diamonds connected to the entities involved. Relationships can have their own attributes, known as relationship attributes.



* **Cardinality**: Describes the number of instances of one entity that can or must be associated with each instance of another entity. Cardinality is usually depicted near the relationship lines, and includes:
  + - **One-to-One (1:1)**: Each instance of Entity A is related to exactly one instance of Entity B and vice versa.
    - **One-to-Many (1:M)**: Each instance of Entity A can be related to multiple instances of Entity B, but each instance of Entity B is related to exactly one instance of Entity A.
    - **Many-to-Many (M:M)**: Each instance of Entity A can be related to multiple instances of Entity B, and each instance of Entity B can be related to multiple instances of Entity A.

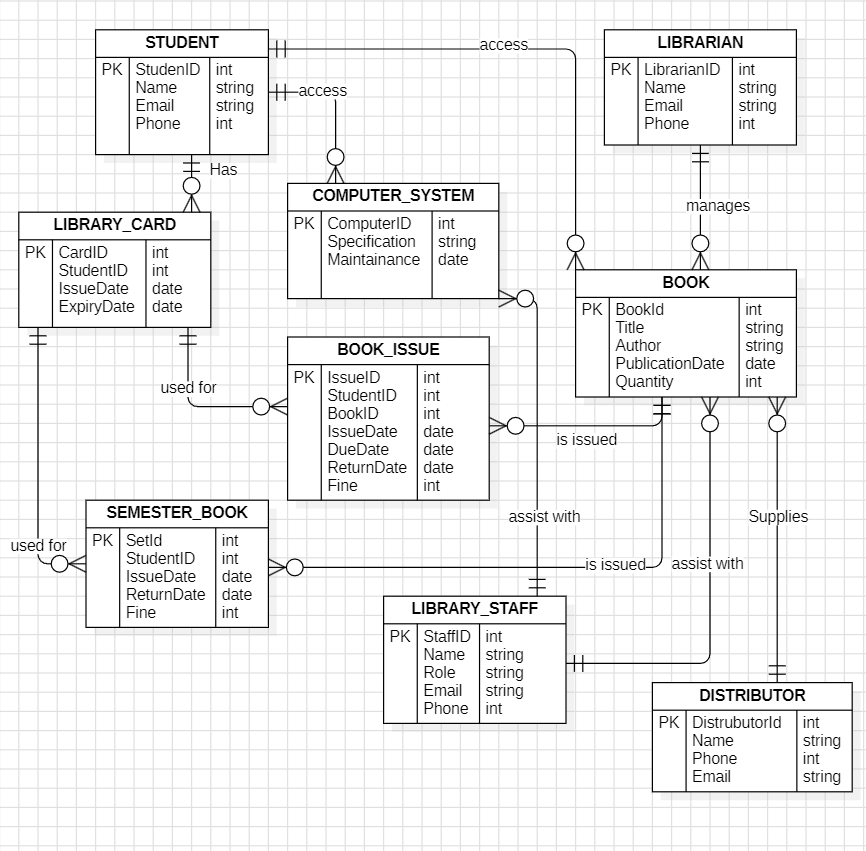
Cardinality In Star UML.



**Primary Key**: An attribute or a set of attributes that uniquely identifies each instance of an entity. It is underlined in the diagram.

**ER DIAGRAM:-**

Library Management System:



Entities of ER Diagram:

**STUDENT**  
Represents students registered in the library system.  
Attributes:

* StudentID (Primary Key)
* Name
* Email
* Phone

**LIBRARIAN**  
Represents the librarian responsible for library operations.  
Attributes:

* LibrarianID (Primary Key)
* Name
* Email
* Phone

**COMPUTER\_SYSTEM**  
Represents computer systems used within the library.  
Attributes:

* ComputerID (Primary Key)
* Specification
* Maintenance

**LIBRARY\_CARD**  
Represents the library card issued to students for book borrowing.  
Attributes:

* CardID (Primary Key)
* StudentID (Foreign Key)
* IssueDate
* ExpiryDate

**BOOK**  
Represents the books available in the library for borrowing.  
Attributes:

* BookID (Primary Key)
* Title
* Author
* PublicationDate
* Quantity

**BOOK\_ISSUE**  
Represents the record of books issued to students.  
Attributes:

* IssueID (Primary Key)
* StudentID (Foreign Key)
* BookID (Foreign Key)
* IssueDate
* DueDate
* ReturnDate
* Fine

**SEMESTER\_BOOK**  
Represents books issued for a semester to students.  
Attributes:

* SetID (Primary Key)
* StudentID (Foreign Key)
* IssueDate
* ReturnDate
* Fine

**LIBRARY\_STAFF**  
Represents staff members working in the library.  
Attributes:

* StaffID (Primary Key)
* Name
* Role
* Email
* Phone

**DISTRIBUTOR**  
Represents book distributors who supply books to the library.  
Attributes:

* DistributorID (Primary Key)
* Name
* Phone
* Email

### Relationships

1. **STUDENT has a LIBRARY\_CARD** (One-to-One)
2. **LIBRARIAN manages COMPUTER\_SYSTEM** (One-to-Many)
3. **LIBRARY\_CARD is used for BOOK\_ISSUE** (One-to-Many)
4. **BOOK is issued for BOOK\_ISSUE** (One-to-Many)
5. **BOOK\_ISSUE is used for SEMESTER\_BOOK** (One-to-Many)
6. **LIBRARY\_STAFF assists with BOOK\_ISSUE** (Many-to-Many)
7. **LIBRARY\_STAFF assists with SEMESTER\_BOOK** (Many-to-Many)
8. **DISTRIBUTOR supplies BOOK** (One-to-Many)

**CONCLUSION:-**

The ER DIAGRAM is Made for Library Management System.

**EXPERIMENT – 3B**

**AIM:-**

To Prepare ER Diagram for Hotel Management System

**REQUIREMENT:-**

StarUML.

**THEORY:-**

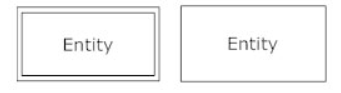
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a

database. An entity in this context is an object, a component of data. An entity set is a

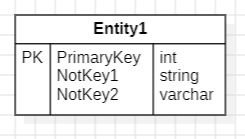
Collection of similar entities. These entities can have attributes that define its properties.

By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. ER diagrams are used to sketch out the design of a database. An ER diagram is a means of visualizing how the information a system produces is related.

* **Entity**: Represents a real-world object or concept that is distinguishable from other objects. Entities are usually represented as rectangles in an ER diagram. Each entity has a name and may have attributes associated with it. A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone



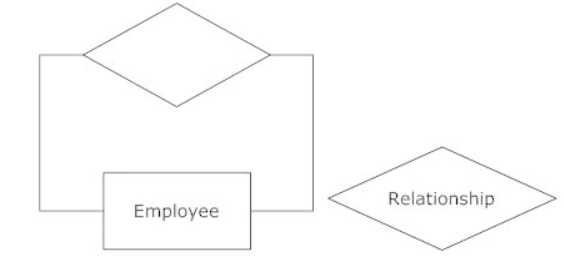
Entity In Star UML with attributes.



* **Attribute**: Describes the properties or characteristics of an entity. Attributes are represented by ovals connected to their respective entities. There are different types of attributes:

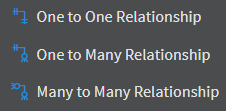


* **Relationship**: Represents the associations between entities. Relationships are depicted as diamonds connected to the entities involved. Relationships can have their own attributes, known as relationship attributes.



* **Cardinality**: Describes the number of instances of one entity that can or must be associated with each instance of another entity. Cardinality is usually depicted near the relationship lines, and includes:
  + - **One-to-One (1:1)**: Each instance of Entity A is related to exactly one instance of Entity B and vice versa.
    - **One-to-Many (1:M)**: Each instance of Entity A can be related to multiple instances of Entity B, but each instance of Entity B is related to exactly one instance of Entity A.
    - **Many-to-Many (M:M)**: Each instance of Entity A can be related to multiple instances of Entity B, and each instance of Entity B can be related to multiple instances of Entity A.

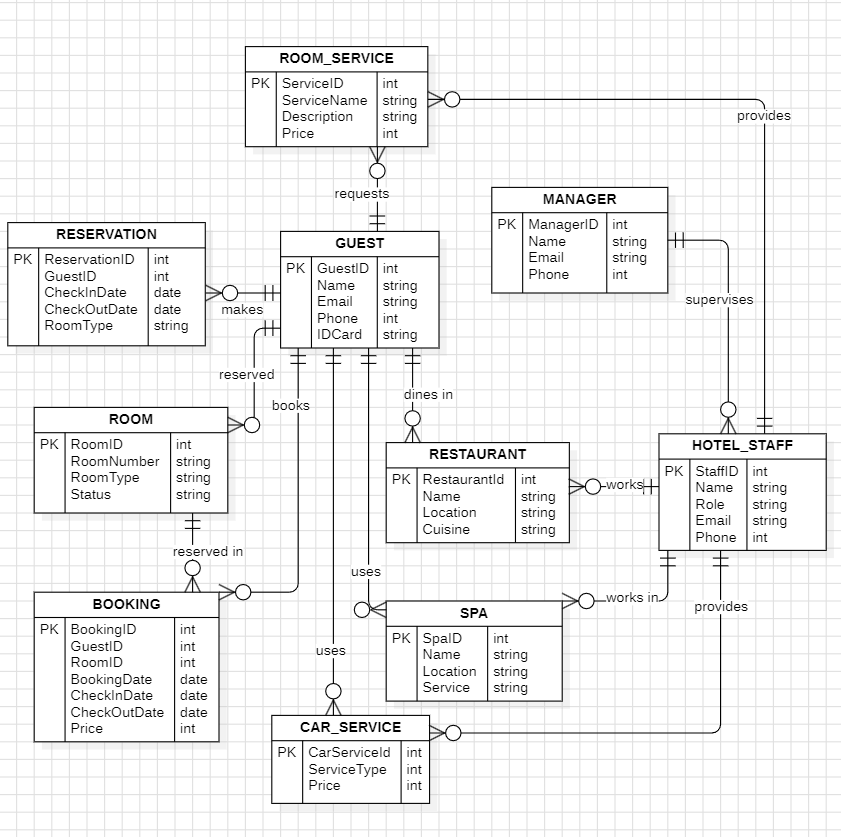
Cardinality In Star UML.



* **Primary Key**: An attribute or a set of attributes that uniquely identifies each instance of an entity. It is underlined in the diagram.

**ER DIAGRAM:-**

Hotel Management System:



Entities of ER Diagram:

**ROOM\_SERVICE**  
Represents the services provided to guests in their rooms.  
Attributes:

* ServiceID (Primary Key)
* ServiceName
* Description
* Price

**MANAGER**  
Represents the managers overseeing hotel operations and services.  
Attributes:

* ManagerID (Primary Key)
* Name
* Email
* Phone

**RESERVATION**  
Represents the reservations made by guests for their stay.  
Attributes:

* ReservationID (Primary Key)
* GuestID (Foreign Key)
* CheckinDate
* CheckOutDate
* RoomType

**GUEST**  
Represents guests who stay at the hotel.  
Attributes:

* GuestID (Primary Key)
* Name
* Email
* Phone
* IDCard

**ROOM**  
Represents the rooms available for guests in the hotel.  
Attributes:

* RoomNumber (Primary Key)
* RoomType
* Status

**RESTAURANT**  
Represents the restaurant facilities within the hotel.  
Attributes:

* RestaurantID (Primary Key)
* Name
* Location
* Cuisine

**HOTEL\_STAFF**  
Represents the staff working in the hotel.  
Attributes:

* StaffID (Primary Key)
* Name
* Role
* Email
* Phone

**BOOKING**  
Represents the bookings made by guests for rooms and services.  
Attributes:

* BookingID (Primary Key)
* GuestID (Foreign Key)
* RoomID (Foreign Key)
* BookingDate
* CheckinDate
* CheckOutDate
* Price

**SPA**  
Represents the spa services available in the hotel.  
Attributes:

* SpaID (Primary Key)
* Name
* Location
* Service

**CAR\_SERVICE**  
Represents the car services available for guests.  
Attributes:

* CarServiceID (Primary Key)
* ServiceType
* Price

### Relationships

1. **RESERVATION is made by a GUEST** (One-to-Many)
2. **RESERVATION is reserved for a ROOM** (One-to-One)
3. **GUEST dines in a RESTAURANT** (Many-to-Many)
4. **GUEST books a BOOKING** (One-to-Many)
5. **ROOM\_SERVICE is provided by a MANAGER** (Many-to-One)
6. **HOTEL\_STAFF supervises a MANAGER** (One-to-One)
7. **HOTEL\_STAFF works in a SPA** (Many-to-One)
8. **HOTEL\_STAFF works in a RESTAURANT** (Many-to-One)
9. **BOOKING uses a SPA** (Many-to-One)
10. **BOOKING uses a CAR\_SERVICE** (Many-to-One)

**CONCLUSION:-**

The ER DIAGRAM is Made for Hotel Management System.

**EXPERIMENT – 4A**

**AIM:-**

To Create Data Flow Diagram for Library Management System

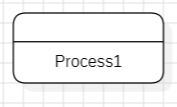
**REQUIREMENT:-**

StarUML.

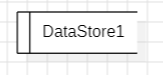
**THEORY:-**

A Data Flow Diagram (DFD) is a graphical tool used to represent the flow of data within a system. It depicts how data moves through various processes, data stores, and external entities. The primary purpose of a DFD is to provide a clear and structured visualization of the system's data processes and interactions, facilitating understanding, analysis, and communication of the system's functionality.

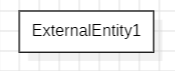
* **Processes**: Represent activities or functions that transform incoming data into outgoing data. Processes are depicted as circles or rectangles with rounded corners in a DFD. Each process should have a unique name and may have inputs and outputs.



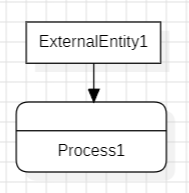
* **Data Stores**: Represent repositories where data is stored and retrieved. Data stores are depicted as open-ended rectangles or parallel lines in a DFD. They store data used or produced by processes.



* **External Entities**: Represent sources or destinations of data outside the system being modelled. External entities are depicted as rectangles in a DFD. They can be people, organizations, or other systems that interact with the system.



* **Data Flows**: Represent the movement of data between processes, data stores, and external entities. Data flows are depicted as arrows in a DFD, showing the direction of data transfer. Each data flow should be labelled to describe the data being transferred.



#### ****Levels of DFDs****

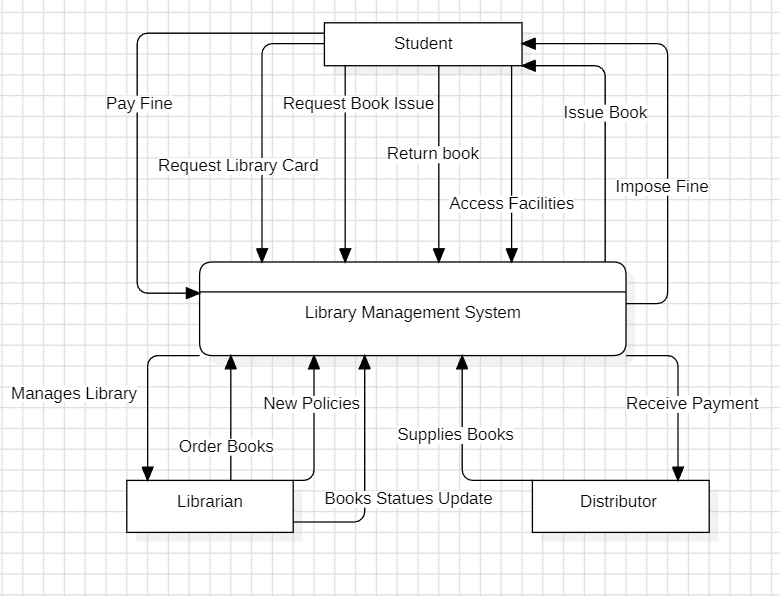
DFDs are often organized into different levels of detail, ranging from high-level overviews to detailed views:

* **Context Diagram (Level 0)**: Provides a high-level overview of the entire system, showing how it interacts with external entities. It includes only one process representing the whole system.
* **Level 1 DFD**: Breaks down the main process of the context diagram into major sub-processes. It provides a more detailed view of the system's interactions and data flows.
* **Level 2 DFD and Beyond**: Further decomposes the processes from the Level 1 DFD into more detailed sub-processes. Each level provides increasingly granular detail about the system’s data flow and processing.

**DATA FLOW DIAGRAM:-**

Library Management System:

Level 0 (Context Diagram)



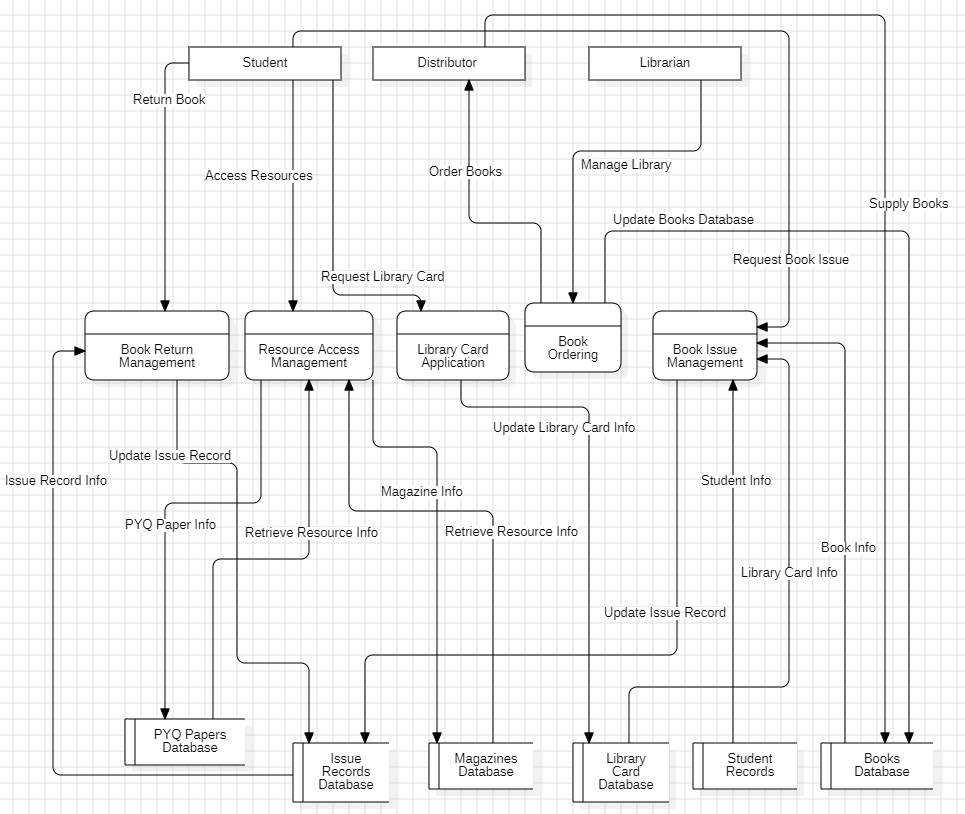
### **Components of Data Flow Diagram:**

### **External Entities:**

1. **Student**:
   * A user who interacts with the library to request books, pay fines, and access library facilities.
   * **Outgoing data flows**:
     + **Request Library Card**: Student applies for a library membership.
     + **Request Book Issue**: Student requests to borrow a book.
     + **Return Book**: Student returns a borrowed book.
     + **Pay Fine**: Student pays the imposed fines for late returns or lost books.
   * **Incoming data flows**:
     + **Issue Book**: Library system issues the requested book to the student.
     + **Impose Fine**: The system informs the student about any fines.
     + **Access Facilities**: Student is granted access to library resources (e.g., database, physical books).
2. **Librarian**:
   * A person responsible for managing library operations, including updating policies and handling book orders.
   * **Outgoing data flows**:
     + **Order Books**: Librarian orders new books through the system.
     + **New Policies**: Librarian inputs new or updated policies into the system.
     + **Manages Library**: The librarian oversees and manages library operations via the system.
   * **Incoming data flows**:
     + **Books Status Update**: System provides updates on the current status of books (e.g., available, checked out).
3. **Distributor**:
   * A third-party entity that supplies books to the library upon receiving orders from the librarian.
   * **Outgoing data flows**:
     + **Supplies Books**: Distributor delivers books to the library as per the orders.
     + **Receive Payment**: Distributor confirms the receipt of payments from the library.
   * **Incoming data flows**:
     + **Order for Books**: The library system sends book orders to the distributor.

### Process:

* **Library Management System (LMS)**:
  + A centralized system that manages book borrowing, fines, librarian operations, and interactions with distributors.
  + **Incoming data flows**:
    - Requests and actions from the Student (e.g., Request Library Card, Request Book Issue, Return Book).
    - Orders and new policies from the Librarian (e.g., Order Books, New Policies).
    - Supplies and receipts from the Distributor (e.g., Supplies Books, Receive Payment).
  + **Outgoing data flows**:
    - Provides services to the Student (e.g., Issue Book, Impose Fine, Access Facilities).
    - Communicates with the Librarian (e.g., Books Status Update).
    - Manages orders with the Distributor (e.g., Order for Books).

Level 1 (Detailed DFD)

### **Components of Data Flow Diagram:**

### External Entities:

1. **Student**:
   * A library user who requests access to resources, returns books, and applies for library cards.
   * **Outgoing data flows**:
     + **Request Library Card**: Student applies for a library card.
     + **Access Resources**: Student accesses library resources (e.g., books, magazines).
     + **Return Book**: Student returns borrowed books.
   * **Incoming data flows**:
     + **Student Info**: Library system processes student details for book issuance.
2. **Librarian**:
   * A person managing library operations, book orders, and updating records.
   * **Outgoing data flows**:
     + **Manage Library**: Librarian oversees the library system.
     + **Order Books**: Librarian places orders for new books.
     + **Update Books Data Store**: Librarian updates the book records with new acquisitions.
   * **Incoming data flows**:
     + **Book Info**: Library system provides information on books for management purposes.
3. **Distributor**:
   * A supplier that provides books to the library based on the orders.
   * **Outgoing data flows**:
     + **Supply Books**: Distributor delivers the ordered books to the library.
   * **Incoming data flows**:
     + **Order Books**: Library system sends requests for new book orders.

### Processes:

1. **Library Card Application**:
   * Handles student requests for library cards and updates the system.
   * **Incoming data flows**:
     + **Request Library Card**: Student requests for a new library card.
   * **Outgoing data flows**:
     + **Update Library Card Info**: Updates the Library Card Data Store with new or modified card information.
2. **Book Ordering**:
   * Manages the process of ordering books from distributors.
   * **Incoming data flows**:
     + **Order Books**: Librarian places orders for new books.
   * **Outgoing data flows**:
     + **Supply Books**: Distributor fulfills the order by delivering books.
     + **Update Books Data Store**: Book information is updated in the data store.
3. **Book Issue Management**:
   * Manages book issuance requests from students and records transaction details.
   * **Incoming data flows**:
     + **Request Book Issue**: Student requests a book for borrowing.
     + **Student Info**: System processes student information for issuing the book.
   * **Outgoing data flows**:
     + **Update Issue Record**: Updates the issue record in the Issue Records Data Store. **Book Info**: Provides the relevant book information from the data store.
4. **Book Return Management**:
   * Manages the return of books and updates the system accordingly.
   * **Incoming data flows**:
     + **Return Book**: Student returns the borrowed book.
   * **Outgoing data flows**:
     + **Update Issue Record**: Updates the book return details in the Issue Records Data Store.
5. **Resource Access Management**:
   * Provides students with access to library resources such as magazines and previous year question papers.
   * **Incoming data flows**:
     + **Access Resources**: Student requests to access library resources.
   * **Outgoing data flows**:
     + **Retrieve Resource Info**: Retrieves resource details from the data stores (e.g., Magazines Data Store, PYQ Papers Data Store).

### Data Stores:

1. **Books Data Store**:
   * Stores all information about the books in the library, including their availability.
   * **Incoming data flows**:
     + **Update Books Data Store**: New book data added by the librarian.
   * **Outgoing data flows**:
     + **Book Info**: Provides book details for issuance and returns.
2. **Library Card Data Store**:
   * Contains records of issued library cards and cardholder information.
   * **Incoming data flows**:
     + **Update Library Card Info**: Records new or updated library card information.
   * **Outgoing data flows**:
     + **Library Card Info**: Provides cardholder information for book issuance.
3. **Issue Records Data Store**:
   * Tracks the issuance and return of books to and from students.
   * **Incoming data flows**:
     + **Update Issue Record**: Updates details of issued or returned books.
   * **Outgoing data flows**:
     + **Issue Record Info**: Provides historical issuance details for management.
4. **Magazines Data Store**:
   * Contains information about magazines available in the library.
   * **Outgoing data flows**:
     + **Magazine Info**: Provides magazine details to the Resource Access Management process.
5. **PYQ Papers Data Store**:
   * Stores previous year question papers that can be accessed by students.
   * **Outgoing data flows**:
     + **PYQ Paper Info**: Provides past question paper details for student access.
6. **Student Records Data Store**:
   * Contains details of all students registered in the library system.
   * **Incoming data flows**:
     + **Student Info**: Information used for book issuance and library card processing.

**CONCLUSION:-**

The Data Flow Diagram is Made for Library Management System.

**EXPERIMENT – 4B**

**AIM:-**

To Create Data Flow Diagram for Hotel Management System

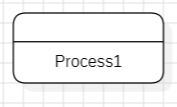
**REQUIREMENT:-**

StarUML.

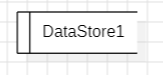
**THEORY:-**

A Data Flow Diagram (DFD) is a graphical tool used to represent the flow of data within a system. It depicts how data moves through various processes, data stores, and external entities. The primary purpose of a DFD is to provide a clear and structured visualization of the system's data processes and interactions, facilitating understanding, analysis, and communication of the system's functionality.

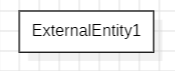
* **Processes**: Represent activities or functions that transform incoming data into outgoing data. Processes are depicted as circles or rectangles with rounded corners in a DFD. Each process should have a unique name and may have inputs and outputs.



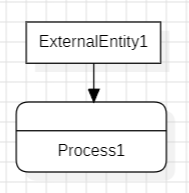
* **Data Stores**: Represent repositories where data is stored and retrieved. Data stores are depicted as open-ended rectangles or parallel lines in a DFD. They store data used or produced by processes.



* **External Entities**: Represent sources or destinations of data outside the system being modelled. External entities are depicted as rectangles in a DFD. They can be people, organizations, or other systems that interact with the system.



* **Data Flows**: Represent the movement of data between processes, data stores, and external entities. Data flows are depicted as arrows in a DFD, showing the direction of data transfer. Each data flow should be labelled to describe the data being transferred.



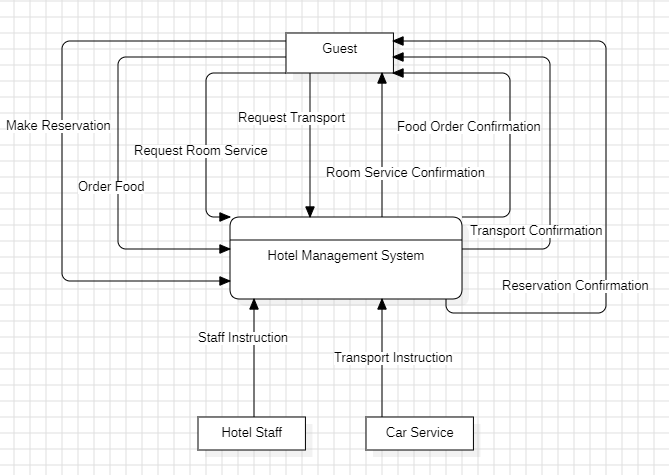
#### ****Levels of DFDs****

DFDs are often organized into different levels of detail, ranging from high-level overviews to detailed views:

* **Context Diagram (Level 0)**: Provides a high-level overview of the entire system, showing how it interacts with external entities. It includes only one process representing the whole system.
* **Level 1 DFD**: Breaks down the main process of the context diagram into major sub-processes. It provides a more detailed view of the system's interactions and data flows.
* **Level 2 DFD and Beyond**: Further decomposes the processes from the Level 1 DFD into more detailed sub-processes. Each level provides increasingly granular detail about the system’s data flow and processing.

**DATA FLOW DIAGRAM:-**

Hotel Management System:

Level 0 (Context Diagram)

### **Components of Data Flow Diagram:**

### Process:

1. **Hotel Management System**:
   * Central system managing guest reservations, room services, and transport requests, facilitating communication between guests, staff, and car services.
   * **Incoming data flows**:
     + **Make Reservation**: Guest requests to book a room.
     + **Request Transport**: Guest requests transportation services.
     + **Request Room Service**: Guest requests room service.
     + **Order Food**: Guest places an order for food.
     + **Staff Instructions**: Provides instructions to hotel staff regarding guest requests and services.
     + **Transport Instructions**: Sends instructions to the car service for transportation needs.
   * **Outgoing data flows**:
     + **Reservation Confirmation**: Confirms the room reservation to the guest.
     + **Food Order Confirmation**: Confirms the Food Order to the guest.
     + **Transport Confirmation**: Confirms transportation arrangements to the guest.
     + **Room Service Confirmation**: Confirms the room service c**onfirmation** to the guest.

### External Entities:

1. **Guest**:
   * A customer seeking accommodation and services from the hotel.
   * **Outgoing data flows**:
     + **Make Reservationt**: Guest applies for a room reservation.
     + **Request Transport**: Guest requests transportation.
     + **Request Room Service**: Guest requests services provided in the room.
     + **Order Food**: Guest places an order for food.
   * **Incoming data flows**:
     + **Reservation Confirmation**: Receives confirmation of the room reservation.
     + **Transport Confirmation**: Receives confirmation for transport arrangements.
     + **Food Order Confirmation**: Confirms the Food Order to the guest.
     + Room Service Confirmation: Confirms the room service c**onfirmation** to the guest.
2. **Hotel Staff**:
   * Personnel responsible for providing services to guests and managing hotel operations.
   * **Outgoing data flows**:
     + **Staff Instructions**: Receives instructions from the Hotel Management System regarding guest requests.
3. **Car Service**:
   * An external service provider for guest transportation needs.
   * **Outgoing data flows**:
     + **Transport Instructions**: Receives instructions from the Hotel Management System for transport arrangements.

Level 1 (Detailed DFD)

### **Components of Data Flow Diagram:**

**External Entities:**

1. **Guest:** An individual seeking accommodation and services from the hotel.
   * **Incoming Data Flows:**
     + **Food Order Confirmation:** Confirms the food order to the guest.
     + **Room Service Confirmation:** Confirms the room service request to the guest.
     + **Transport Confirmation:** Confirms transportation arrangements to the guest.
   * **Outgoing Data Flows:**
     + **Make Reservation:** Guest applies for a room reservation.
     + **Request Transportation:** Guest requests transportation services.
     + **Request Room Service Request:** Guest requests room service.
     + **Order Food:** Guest places an order for food.
2. **Car Service:** A service provider for guest transportation needs.
   * **Outgoing Data Flows:**
     + **Transport Instructions:** Provides transport service updates to the hotel management system.
3. **Hotel Staff:** Personnel responsible for managing hotel operations and providing services to guests.
   * **Incoming Data Flows:**
     + **Manage Staff and Resources:** Receives work instructions from the hotel management system based on guest needs.
   * **Outgoing Data Flows:**
     + **Instructions and Updates:** Provides updates or requests for support to the hotel management system.

**Processes:**

1. **Reservation Management:** Manages room bookings and confirmations for guests.
   * **Incoming Data Flows:**
     + **Make Reservation:** Receives reservation requests from Guests.
   * **Outgoing Data Flows:**
     + **Update Room Info:** **Update Room info in database**.
     + Update Reservation Record: **Update Reservation Record in database**.
2. **Food and Beverage Management:** Handles the management of guest food orders and dining services.
   * **Outgoing Data Flows:**
     + **Food Order Confirmation:** Confirms food orders from Guests.
     + Send Food Order: Send food order to food and beverage record.
3. **Room Service Management:** Manages guest requests for room service.
   * **Incoming Data Flows:**
     + **Room Service Request:** Receives room service requests from Guests.
   * **Outgoing Data Flows:**
     + **Send Room Service Request:** Sends room service confirmations to Guests.
4. **Transportation Management:** Oversees transportation arrangements for guests.
   * **Incoming Data Flows:**
     + **Request Transportation:** Receives transport requests from Guests.
     + Transportation Instructions: Instructions for transport services.
   * **Outgoing Data Flows:**
     + **Transport Confirmation:** Sends transport confirmations to Guests.
     + **Send Transport Request:** Sends transport instructions to Car Service.
5. **Resource and Staff Management:** Manages staff scheduling, resource allocation, and tasks.
   * **Incoming Data Flows:**
     + **Instructions and Updates:** Receives updates and requests from Hotel Staff.
   * **Outgoing Data Flows:**
     + **Manage Staff and Resources:** Sends instructions to Hotel Staff based on guest needs.

**Data Stores:**

1. **Room Database:** Stores information about available rooms and their current statuses.
   * **Incoming Data Flows:**
     + **Update Room Info:** Receives updates about available rooms from Reservation Management.
2. **Reservation Records:** Maintains a log of all guest reservations.
   * **Incoming Data Flows:**
     + **Update Reservation Record:** Receives new reservation details from Reservation Management.
3. **Food and Beverage Orders:** Records details of food and beverage orders placed by guests.
   * **Incoming Data Flows:**
     + **Send Food Order:** Receives new food order data from Food and Beverage Management.
4. **Room Service Requests:** Logs all room service requests made by guests.
   * **Incoming Data Flows:**
     + **Send Room Service Request:** Receives new room service requests from Room Service Management.
5. **Car Service Records:** Stores transportation service details for guest arrangements.
   * **Incoming Data Flows:**
     + **Send Transport Request:** Receives updates about transport services from Transportation Management.

**CONCLUSION:-**

The Data Flow Diagram is Made for Hotel Management System.

**EXPERIMENT – 5A**

**AIM:-**

To Prepare Use Case Diagram for Library Management System.

**REQUIREMENT:-**

StarUML.

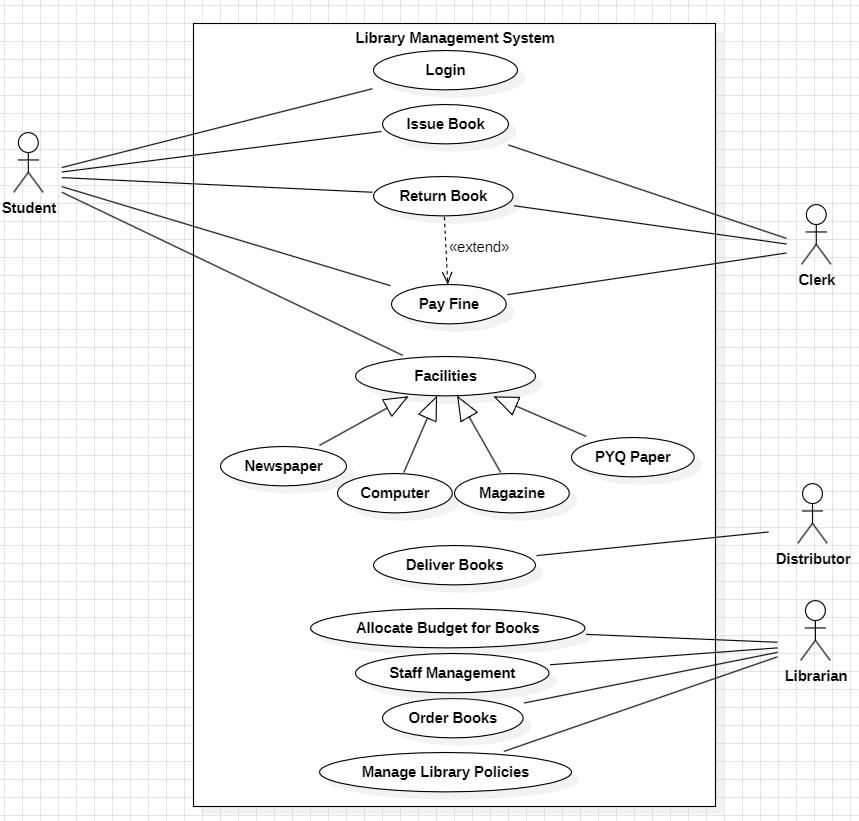
**THEORY:-**

A **Use Case Diagram** is a visual representation in software engineering that depicts the interactions between users (actors) and a system to achieve specific goals. It is part of the Unified Modeling Language (UML) and is primarily used to capture the functional requirements of a system.

#### Key Components of a Use Case Diagram:

1. **Actors:**
   * Represent users or entities that interact with the system.
   * Actors can be human users, external systems, or hardware devices.
   * They are represented by stick figures.
   * **Example:** A “Customer” booking a hotel room, or a “Receptionist” managing check-ins.
2. **Use Cases:**
   * Describe the actions or tasks that the system performs in response to the actor’s requests.
   * Use cases capture high-level functionality and represent specific services or operations within the system.
   * They are represented as ovals and named to reflect the task or service.
   * **Example:** “Make Reservation,” “Check-In,” “Provide Room Service.”
3. **System Boundary:**
   * Defines the scope of the system and what is included in the system's functionality.
   * It is represented as a rectangle enclosing the use cases.
4. **Relationships:**
   * **Association:** A line connecting actors to the use cases they interact with.
   * **Include:** A relationship where one use case always uses another use case. Represented by a dashed arrow with the label <<include>>.
   * **Extend:** A relationship where a use case can be optionally extended by another. Represented by a dashed arrow with the label <<extend>>.
   * **Generalization:** Shows inheritance between actors or use cases, where one actor/use case is a specialized version of another.

**Use Case Diagram: Library Management System:-**

****

**CONCLUSION:-**

The Use Case Diagram is Made for Library Management System.

**EXPERIMENT – 5B**

**AIM:-**

To Prepare Use Case Diagram for Hotel Management System.

**REQUIREMENT:-**

StarUML.

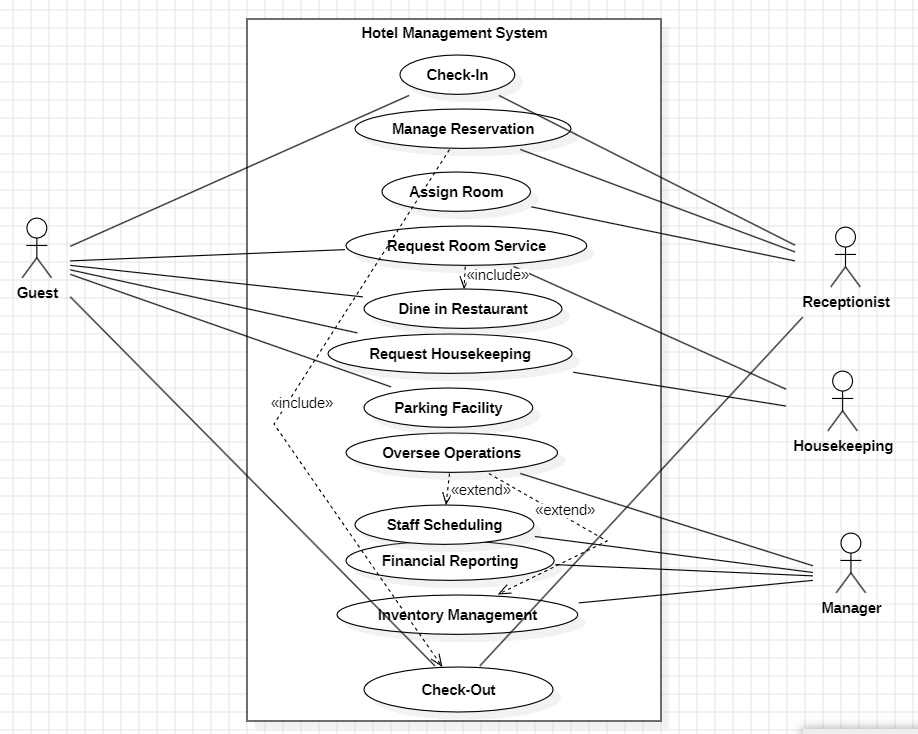
**THEORY:-**

A **Use Case Diagram** is a visual representation in software engineering that depicts the interactions between users (actors) and a system to achieve specific goals. It is part of the Unified Modeling Language (UML) and is primarily used to capture the functional requirements of a system.

#### Key Components of a Use Case Diagram:

1. **Actors:**
   * Represent users or entities that interact with the system.
   * Actors can be human users, external systems, or hardware devices.
   * They are represented by stick figures.
   * **Example:** A “Customer” booking a hotel room, or a “Receptionist” managing check-ins.
2. **Use Cases:**
   * Describe the actions or tasks that the system performs in response to the actor’s requests.
   * Use cases capture high-level functionality and represent specific services or operations within the system.
   * They are represented as ovals and named to reflect the task or service.
   * **Example:** “Make Reservation,” “Check-In,” “Provide Room Service.”
3. **System Boundary:**
   * Defines the scope of the system and what is included in the system's functionality.
   * It is represented as a rectangle enclosing the use cases.
4. **Relationships:**
   * **Association:** A line connecting actors to the use cases they interact with.
   * **Include:** A relationship where one use case always uses another use case. Represented by a dashed arrow with the label <<include>>.
   * **Extend:** A relationship where a use case can be optionally extended by another. Represented by a dashed arrow with the label <<extend>>.
   * **Generalization:** Shows inheritance between actors or use cases, where one actor/use case is a specialized version of another.

**Use Case Diagram: Hotel Management System:-**

****

**CONCLUSION:-**

The Use Case Diagram is Made for Hotel Management System.

**EXPERIMENT – 6A**

**AIM:-**

To Prepare Class Diagram & Object Diagram for Library Management System.

**REQUIREMENT:-**

StarUML.

**THEORY:-**

**Class diagrams** are generally used for conceptual modeling of the static view of a software application. They help translate models into programming code in a detailed manner. During the development or construction of software systems, class diagrams are widely utilized. They are also instrumental in data modeling, showcasing classes, relationships among them, interfaces, associations, and more. A class in a class diagram serves as a blueprint for an object, describing and explaining the different types of objects in the system and the various relationships that exist between them.

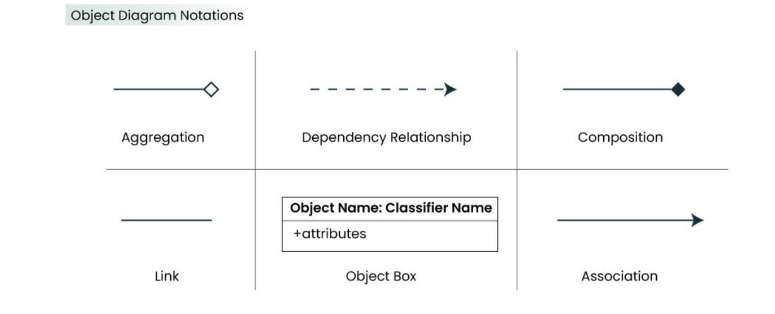
When designing a class diagram, two important points to consider are **aggregation** and **multiplicity**. Let us explore these concepts in detail:

1. **Aggregation**: This represents a relationship where one entity can exist independently of another. It means creating or composing different abstractions together to define a class. Aggregation is depicted as a part-of relationship in the class diagram.
2. **Multiplicity**: This refers to the number of elements of a class that are associated with another class. These relationships can be one-to-one, many-to-many, many-to-one, or one-to-many. To denote the number of elements:
   * **1** for one element
   * **0** for zero elements
   * **\*** for many elements

**Object diagrams** are a visual representation in UML (Unified Modeling Language) that illustrate instances of classes and their relationships within a system at a specific point in time. They display objects, their attributes, and the links between them, providing a snapshot of the system’s structure during execution. An object diagram can be viewed as a screenshot of the instances in a system and the relationships that exist between them.

### Object Diagram Notations

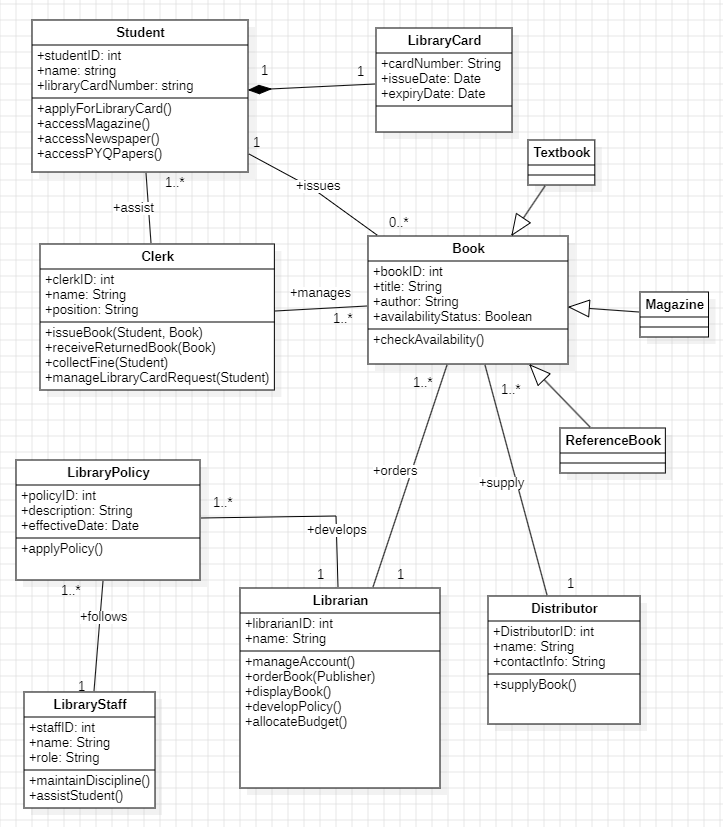
* **Objects**: Represented as rectangles, showing the name of the object followed by its class name. For example, objectName: ClassName.



When we instantiate a classifier in a system, the object we create represents an entity that exists within the system. Changes in the object over time can be represented by creating multiple instance specifications. In an object diagram, a rectangle represents an object.

* **Attributes and Values**: Inside the object box, attributes of the object are listed along with their specific values.
* **Link**: A link represents a relationship between two objects. The number of participants on the link for each object is indicated at the end of the link. A solid line is used to represent a link between two objects, specifying the relationship between instance specifications.
* **Dependency Relationships**: This shows when one element depends on another. Key points include:
  + Any change in the definition or structure of one element may cause changes to the other.
  + This is a unidirectional relationship between two objects.
  + Types of dependency relationships in UML include:
    - **Abstraction**
    - **Binding**
    - **Realization**
    - **Substitution**
    - **Usage**
* **Association**: Association is a reference relationship between two objects (or classes). An association line connects two object boxes, representing a relationship between instances of two classes. It can be uni-directional or bi-directional, represented by an arrow.
* **Aggregation**: This represents a "has a" relationship. A hollow diamond is placed on the containing object with a line connecting it to the contained object. Key characteristics include:
  + Aggregation is a specific form of association.
  + It represents a parent-child relationship, though it isn’t inheritance.
  + The lifecycle of the contained objects does not strongly depend on the lifecycle of the container objects.
* **Composition**: Composition is a type of association where the child cannot exist independently of the parent. A filled diamond is used on the containing object with a line connecting it to the contained object. Important notes include:
  + Composition is a special type of association.
  + It represents a parent-child relationship, but it is not inheritance.
  + This relationship is used when the independent existence of the child is not possible.

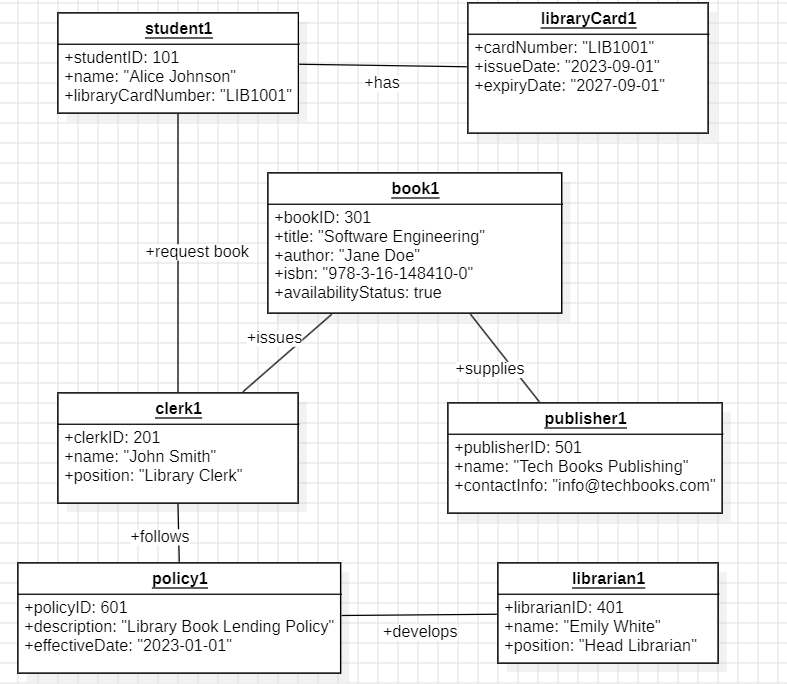
**Class Diagram: Library Management System**

****

### ****Classes and Relationships****

1. **Class: Student**
   * **Attributes**:
     + studentID: int
     + name: String
     + libraryCardNumber: String
   * **Methods**:
     + requestLibraryCard()
     + accessMagazine()
     + accessNewspaper()
     + accessPYQPapers()
   * **Relationships**:
     + **Association**: 1..\* with **LibraryCard** (A student can have one or more library cards.)
     + **Association**: 0..\* with **Book** (A student may issue multiple books with the help of the clerk.)
2. **Class: Clerk**
   * **Attributes**:
     + clerkID: int
     + name: String
     + position: String
   * **Methods**:
     + issueBook(Student, Book)
     + receiveReturnedBook(Book)
     + collectFine(Student)
     + manageLibraryCardRequest(Student)
   * **Relationships**:
     + **Association**: 1..\* with **Book** (A clerk manages the issuance of multiple books.)
     + **Association**: 1..\* with **Student** (The clerk assists multiple students with book issues and returns.)
3. **Class: LibraryCard**
   * **Attributes**:
     + cardNumber: String
     + issueDate: Date
     + expiryDate: Date
   * **Methods**:
     + renewCard()
   * **Relationships**:
     + **Composition**: 1..1 with **Student** (Each library card is part of one student, and if the student is removed, the library card is also removed.)
4. **Class: Book**
   * **Attributes**:
     + bookID: int
     + title: String
     + author: String
     + isbn: String
     + availabilityStatus: Boolean
   * **Methods**:
     + checkAvailability()
   * **Relationships**:
     + **Association**: 1..\* with **Student** (Books are issued to students.)
     + **Association**: 1..\* with **Clerk** (Managed by clerks for issuance and returns.)
     + **Generalization**: Generalize into **ReferenceBook** and **TextBook** (if applicable).
5. **Class: Librarian**
   * **Attributes**:
     + librarianID: int
     + name: String
     + position: String
   * **Methods**:
     + orderBook(Publisher)
     + manageAccount()
     + displayBook()
     + developPolicy()
   * **Relationships**:
     + **Association**: 1..\* with **Book** (Librarians manage the library's book inventory.)
     + **Association**: 1..\* with **LibraryPolicy** (A librarian develops multiple policies.)
6. **Class: LibraryPolicy**
   * **Attributes**:
     + policyID: int
     + description: String
     + effectiveDate: Date
   * **Methods**:
     + applyPolicy()
   * **Relationships**:
     + **Association**: 1..\* with **Librarian** (Policies are developed and implemented by librarians.)
7. **Class: Publisher**
   * **Attributes**:
     + publisherID: int
     + name: String
     + contactInfo: String
   * **Methods**:
     + supplyBook()
   * **Relationships**:
     + **Association**: 1..\* with **Book** (Supplies books to the library.)
8. **Class: LibraryStaff**
   * **Attributes**:
     + staffID: int
     + name: String
     + role: String
   * **Methods**:
     + maintainDiscipline()
     + assistStudent()
   * **Relationships**:
     + **Association**: 1..\* with **LibraryPolicy** (Library staff follow library policies.)

**Object Diagram: Library Management System**

****

#### ****Objects and Instances****

1. **Student**
   * **Instance**: student1
     + Attributes:
       - studentID: 101
       - name: "Alice Johnson"
       - libraryCardNumber: "LIB1001"
2. **LibraryCard**
   * **Instance**: libraryCard1
     + Attributes:
       - cardNumber: "LIB1001"
       - issueDate: "2023-09-01"
       - expiryDate: "2025-09-01"
3. **Clerk**
   * **Instance**: clerk1
     + Attributes:
       - clerkID: 201
       - name: "John Smith"
       - position: "Library Clerk"
4. **Book**
   * **Instance**: book1
     + Attributes:
       - bookID: 301
       - title: "Data Structures and Algorithms"
       - author: "Jane Doe"
       - isbn: "978-3-16-148410-0"
       - availabilityStatus: true
5. **Librarian**
   * **Instance**: librarian1
     + Attributes:
       - librarianID: 401
       - name: "Emily White"
       - position: "Head Librarian"
6. **Publisher**
   * **Instance**: publisher1
     + Attributes:
       - publisherID: 501
       - name: "Tech Books Publishing"
       - contactInfo: "info@techbooks.com"
7. **LibraryPolicy**
   * **Instance**: policy1
     + Attributes:
       - policyID: 601
       - description: "Library Book Lending Policy"
       - effectiveDate: "2023-01-01"

#### ****Object Relationships****

* **student1** **has** a **libraryCard1** (Composition)
* **student1** **requests** **book1** **through** **clerk1** (Association)
* **clerk1** **issues** **book1** **to** **student1** (Association)
* **librarian1** **develops** **policy1** (Association)
* **publisher1** **supplies** **book1** (Association)
* **librarian1** **manages** **book1** (Association)

**CONCLUSION:-**

The Class Diagram & Object Diagram is Made for Library Management System.

**EXPERIMENT – 6B**

**AIM:-**

To Prepare Class Diagram & Object Diagram for Hotel Management System.

**REQUIREMENT:-**

StarUML.

**THEORY:-**

**Class diagrams** are generally used for conceptual modeling of the static view of a software application. They help translate models into programming code in a detailed manner. During the development or construction of software systems, class diagrams are widely utilized. They are also instrumental in data modeling, showcasing classes, relationships among them, interfaces, associations, and more. A class in a class diagram serves as a blueprint for an object, describing and explaining the different types of objects in the system and the various relationships that exist between them.

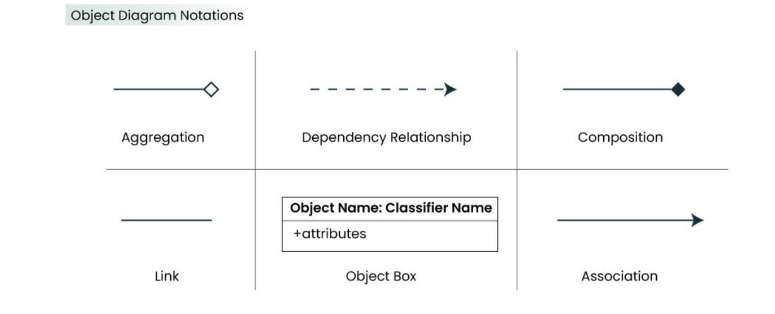
When designing a class diagram, two important points to consider are **aggregation** and **multiplicity**. Let us explore these concepts in detail:

1. **Aggregation**: This represents a relationship where one entity can exist independently of another. It means creating or composing different abstractions together to define a class. Aggregation is depicted as a part-of relationship in the class diagram.
2. **Multiplicity**: This refers to the number of elements of a class that are associated with another class. These relationships can be one-to-one, many-to-many, many-to-one, or one-to-many. To denote the number of elements:
   * **1** for one element
   * **0** for zero elements
   * **\*** for many elements

**Object diagrams** are a visual representation in UML (Unified Modeling Language) that illustrate instances of classes and their relationships within a system at a specific point in time. They display objects, their attributes, and the links between them, providing a snapshot of the system’s structure during execution. An object diagram can be viewed as a screenshot of the instances in a system and the relationships that exist between them.

### Object Diagram Notations

* **Objects**: Represented as rectangles, showing the name of the object followed by its class name. For example, objectName: ClassName.



When we instantiate a classifier in a system, the object we create represents an entity that exists within the system. Changes in the object over time can be represented by creating multiple instance specifications. In an object diagram, a rectangle represents an object.

* **Attributes and Values**: Inside the object box, attributes of the object are listed along with their specific values.
* **Link**: A link represents a relationship between two objects. The number of participants on the link for each object is indicated at the end of the link. A solid line is used to represent a link between two objects, specifying the relationship between instance specifications.
* **Dependency Relationships**: This shows when one element depends on another. Key points include:
  + Any change in the definition or structure of one element may cause changes to the other.
  + This is a unidirectional relationship between two objects.
  + Types of dependency relationships in UML include:
    - **Abstraction**
    - **Binding**
    - **Realization**
    - **Substitution**
    - **Usage**
* **Association**: Association is a reference relationship between two objects (or classes). An association line connects two object boxes, representing a relationship between instances of two classes. It can be uni-directional or bi-directional, represented by an arrow.
* **Aggregation**: This represents a "has a" relationship. A hollow diamond is placed on the containing object with a line connecting it to the contained object. Key characteristics include:
  + Aggregation is a specific form of association.
  + It represents a parent-child relationship, though it isn’t inheritance.
  + The lifecycle of the contained objects does not strongly depend on the lifecycle of the container objects.
* **Composition**: Composition is a type of association where the child cannot exist independently of the parent. A filled diamond is used on the containing object with a line connecting it to the contained object. Important notes include:
  + Composition is a special type of association.
  + It represents a parent-child relationship, but it is not inheritance.
  + This relationship is used when the independent existence of the child is not possible.

**Class Diagram: Hotel Management System**

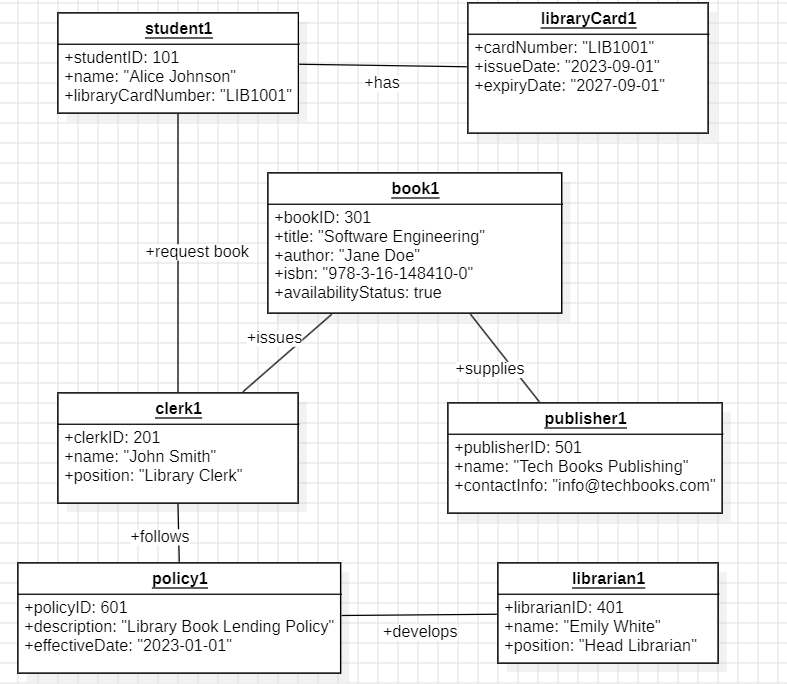
****

#### ****Classes, Attributes, Operations, and Relationships****

1. **Guest**
   * Attributes:
     + guestID: int
     + name: String
     + contactInfo: String
     + reservationStatus: String
   * Operations:
     + makeReservation()
     + requestRoomService()
     + checkOut()
     + receiveRefreshments()
   * Relationships:
     + **Association**:
       - Guest has a Reservation (one-to-many relationship, a guest can have multiple reservations).
       - Guest requests RoomService (one-to-many relationship, a guest can place multiple orders).
2. **Reservation**
   * Attributes:
     + reservationID: int
     + checkInDate: Date
     + checkOutDate: Date
     + roomType: String
     + paymentStatus: String
   * Operations:
     + confirmReservation()
     + cancelReservation()
     + updatePaymentStatus()
   * Relationships:
     + **Association**:
       - Associated with Guest (a guest makes a reservation).
       - Associated with Room (a reservation is for a specific room).
3. **Room**
   * Attributes:
     + roomNumber: int
     + roomType: String
     + availabilityStatus: boolean
     + price: double
   * Operations:
     + checkAvailability()
     + assignRoom()
     + updateAvailabilityStatus()
4. **Receptionist**
   * Attributes:
     + staffID: int
     + name: String
     + position: String
   * Operations:
     + manageReservation()
     + checkInGuest()
     + processCheckOut()
   * Relationships:
     + **Association**: Handles Reservation (the receptionist manages reservations for guests).
5. **RoomService**
   * Attributes:
     + serviceID: int
     + orderDetails: String
     + deliveryStatus: String
   * Operations:
     + takeOrder()
     + deliverOrder()
     + updateDeliveryStatus()
   * Relationships:
     + **Association**: Associates with Guest (room service is requested by guests).
6. **Housekeeping**
   * Attributes:
     + staffID: int
     + name: String
     + schedule: String
   * Operations:
     + cleanRoom()
     + handleSpecialRequest()
     + updateRoomStatus()
   * Relationships:
     + **Composition**: Maintains Room (housekeeping is responsible for maintaining the state of rooms).
7. **Manager**
   * Attributes:
     + staffID: int
     + name: String
     + position: String

* Operations:
  + overseeOperations()
  + manageStaff()
  + handleBudgeting()
* Relationships:
  + **Generalization**: Oversees Receptionist, Housekeeping.

**Object Diagram: Hotel Management System**

****

#### ****Instances****

1. **Guest**
   * Instance: guest1
     + Attributes:
       - guestID: 101
       - name: "John Doe"
       - contactInfo: "john.doe@example.com"
       - reservationStatus: "Confirmed"
2. **Reservation**
   * Instance: reservation1
     + Attributes:
       - reservationID: 501
       - checkInDate: "2024-10-30"
       - checkOutDate: "2024-11-05"
       - roomType: "Deluxe"
       - paymentStatus: "Paid"
3. **Room**
   * Instance: room101
     + Attributes:
       - roomNumber: 101
       - roomType: "Deluxe"
       - availabilityStatus: false (indicating it is occupied)
       - price: 200.00
4. **Receptionist**
   * Instance: receptionist1
     + Attributes:
       - staffID: 201
       - name: "Emily Smith"
       - position: "Front Desk Manager"
5. **Chef**
   * Instance: chef1
     + Attributes:
       - staffID: 301
       - name: "Michael Gordon"
       - specialty: "Italian Cuisine"
6. **RoomService**
   * Instance: roomService1
     + Attributes:
       - serviceID: 401
       - orderDetails: "Pizza and Coke"
       - deliveryStatus: "Delivered"
7. **Housekeeping**
   * Instance: housekeeping1
     + Attributes:
       - staffID: 501
       - name: "Linda Johnson"
       - schedule: "Daily"
8. **CarService**
   * Instance: carService1
     + Attributes:
       - serviceID: 601
       - serviceDetails: "Airport Pickup"
9. **Gardener**
   * Instance: gardener1
     + Attributes:
       - staffID: 701
       - name: "Sam Green"
       - gardenArea: "Main Lawn"
10. **Manager**
    * Instance: manager1
      + Attributes:
        - staffID: 801
        - name: "Anna Taylor"
        - position: "Operations Manager"

**CONCLUSION:-**

The Class Diagram & Object Diagram is Made for Hotel Management System.

**EXPERIMENT – 7A**

**AIM:-**

To Prepare State-chart Diagram & Activity Diagram for Library Management System

**REQUIREMENT:-**

StarUML.

**THEORY:-**

A **State-chart Diagram** (or State Diagram) is a behavioral UML (Unified Modeling Language) diagram that models the dynamic behavior of an object over time. It focuses on the different states the object undergoes and the transitions between those states. State-chart diagrams are particularly useful for representing complex entities that go through multiple stages or statuses in response to events, making them ideal for systems with a lifecycle or workflow, such as order processing, user authentication, or equipment monitoring.

### Purpose of State-chart Diagrams

The primary purposes of a state-chart diagram include:

* **Lifecycle Representation**: Illustrates how an object transitions from one state to another based on events.
* **Behavior Modeling**: Models the behavior of a system or subsystem where objects exhibit multiple distinct states.
* **Event Trigger Visualization**: Provides a clear visual representation of how different events trigger state changes in a specific object or process.

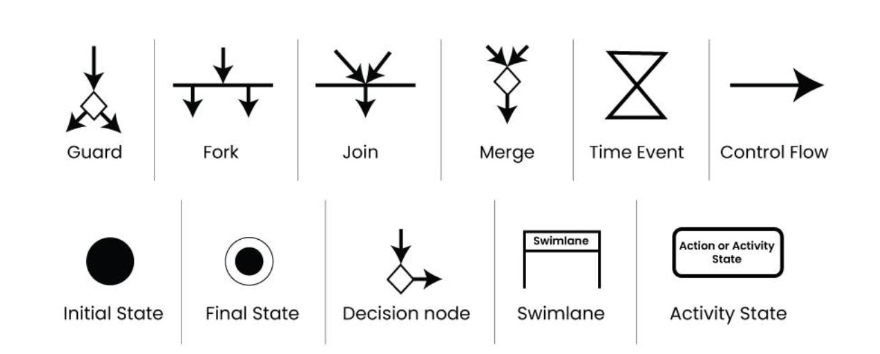
### Key Elements of State-chart Diagrams

* **States**:
  + **Definition**: A state represents a condition or situation during the life of an object where it satisfies some condition, performs an activity, or waits for an event.
  + **Notation**: Represented by rounded rectangles labeled with the state name, possibly including actions associated with entry, exit, or internal activities.
  + **Examples**: In an order processing system, states might include Pending, Shipped, Delivered, Canceled, etc.
* **Transitions**:
  + **Definition**: Transitions are directed lines that show the movement from one state to another in response to an event.
  + **Notation**: Represented by arrows between states, typically labeled with the event name and optional guard conditions.
  + **Triggering Events**: Conditions or actions that initiate a transition, such as “Payment Approved” or “Cancel Order.”
* **Events**:
  + **Definition**: An event is a trigger that causes an object to transition from one state to another.
  + **Examples**: User actions like “Submit Form,” system actions like “Timeout,” or external events like “Receive Shipment.”
* **Actions**:
  + **Definition**: Actions are operations executed during a transition or as an entry/exit action in a state.
  + **Types**:
    - **Entry Action**: An action performed on entering a state.
    - **Exit Action**: An action performed on leaving a state.
    - **Internal Action**: An ongoing activity within the state.
* **Conditions (Guard Conditions)**:
  + **Definition**: Boolean expressions that determine whether a transition can occur.
  + **Usage**: Guard conditions must be true for the transition to take place; for instance, “if inventory > 0” might allow an order to transition from Pending to Confirmed.
* **Initial and Final States**:
  + **Initial State**: Indicates where the object lifecycle begins, shown as a filled black circle.
  + **Final State**: Represents the completion of an object’s lifecycle, depicted by a bullseye symbol (a filled circle within another circle).

### **Activity Diagram**

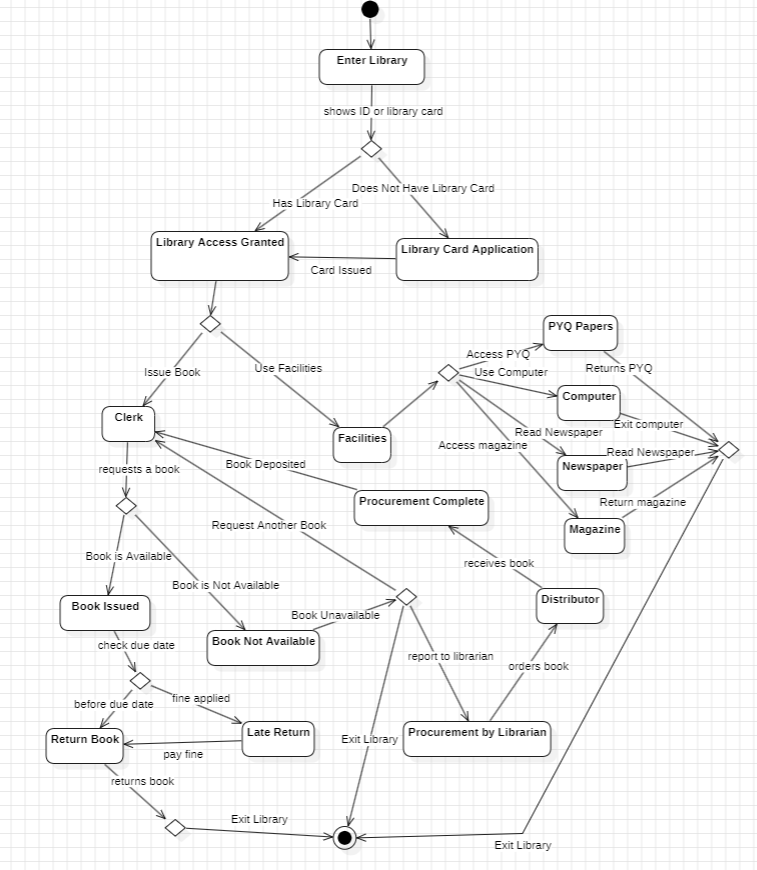
Activity diagrams are a crucial part of the Unified Modeling Language (UML) that help visualize workflows, processes, or activities within a system. They illustrate how different actions are connected and how a system transitions from one state to another. By providing a clear picture of both simple and complex workflows, activity diagrams facilitate understanding among developers and stakeholders regarding how various elements interact within a system.

### Activity Diagram Notations

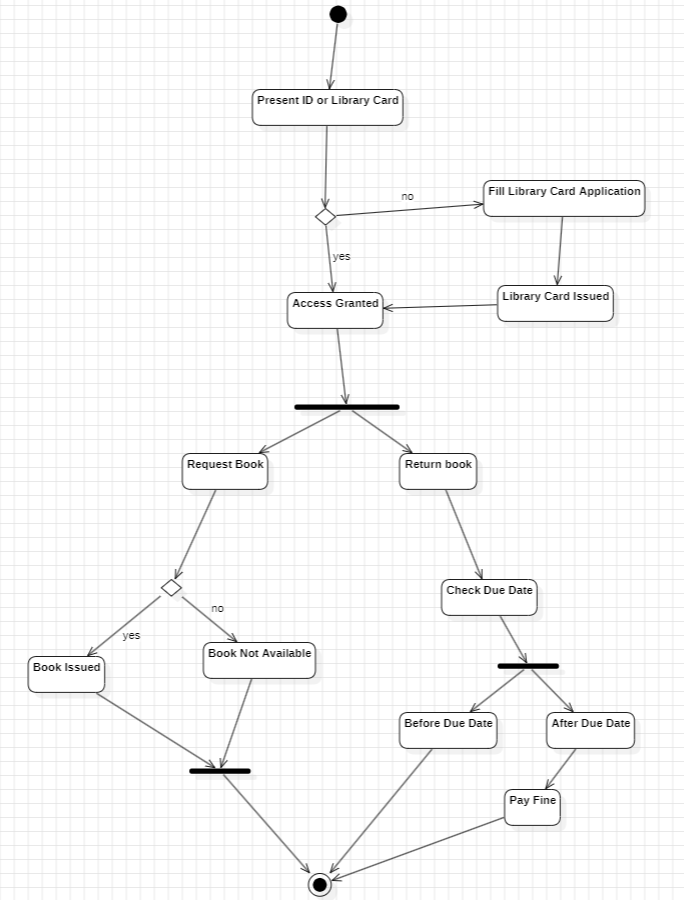


1. **Initial State**:
   * The starting point before any activity occurs is depicted using the initial state, represented by a filled black circle. A process can have only one initial state unless depicting nested activities. This state marks the entry point and the initial activity state.
2. **Action or Activity State**:
   * An activity represents the execution of an action on objects or by objects. Activities are depicted using rectangles with rounded corners, representing any action or event that takes place.
3. **Action Flow or Control Flows**:
   * These are the paths or edges used to show transitions from one activity state to another.
4. **Decision Node and Branching**:
   * A decision node is used to indicate where a decision is required before continuing the flow of control. It typically has two or more outgoing arrows, which can be labeled with conditions or guard expressions.
5. **Guard**:
   * A guard is a statement written next to a decision node on an arrow (often in square brackets) that must be true for control to follow a particular direction. Guards help define the constraints and conditions determining the flow of a process.
6. **Fork**:
   * Fork nodes support concurrent activities, allowing both activities to execute simultaneously without making a decision first. Forks are represented by a rounded solid rectangular bar, with an incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.
7. **Join**:
   * Join nodes converge concurrent activities into one. They have two or more incoming edges and one outgoing edge.
8. **Merge or Merge Event**:
   * This notation is used when activities that are not executed concurrently need to merge. It allows for merging two or more activities into one, irrespective of the path chosen to reach it.
9. **Swimlanes**:
   * Swimlanes group related activities into columns or rows, adding modularity to the activity diagram. While not mandatory, using swimlanes enhances clarity by delineating responsibilities, similar to creating functions in programming.
10. **Time Event**:
    * Represented by an hourglass, a time event indicates a stoppage in the flow for a duration, often occurring in scenarios where an event takes time to complete.
11. **Final State or End State**:
    * The final state, represented by a filled circle within another circle, indicates when a particular process or activity concludes. A system or process can have multiple final states.

**State-chart Diagram: Library Management System**

****

**Activity Diagram: Library Management System**

****

**CONCLUSION:-**

The State-chart Diagram and Activity Diagram is Made for Library Management

System.

**EXPERIMENT – 7B**

**AIM:-**

To Prepare State-chart Diagram & Activity Diagram for Hotel Management System

**REQUIREMENT:-**

StarUML.

**THEORY:-**

A **State-chart Diagram** (or State Diagram) is a behavioral UML (Unified Modeling Language) diagram that models the dynamic behavior of an object over time. It focuses on the different states the object undergoes and the transitions between those states. State-chart diagrams are particularly useful for representing complex entities that go through multiple stages or statuses in response to events, making them ideal for systems with a lifecycle or workflow, such as order processing, user authentication, or equipment monitoring.

### Purpose of State-chart Diagrams

The primary purposes of a state-chart diagram include:

* **Lifecycle Representation**: Illustrates how an object transitions from one state to another based on events.
* **Behavior Modeling**: Models the behavior of a system or subsystem where objects exhibit multiple distinct states.
* **Event Trigger Visualization**: Provides a clear visual representation of how different events trigger state changes in a specific object or process.

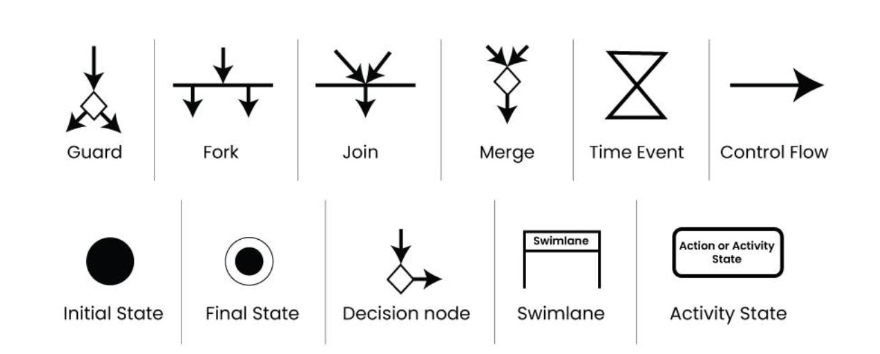
### Key Elements of State-chart Diagrams

* **States**:
  + **Definition**: A state represents a condition or situation during the life of an object where it satisfies some condition, performs an activity, or waits for an event.
  + **Notation**: Represented by rounded rectangles labeled with the state name, possibly including actions associated with entry, exit, or internal activities.
  + **Examples**: In an order processing system, states might include Pending, Shipped, Delivered, Canceled, etc.
* **Transitions**:
  + **Definition**: Transitions are directed lines that show the movement from one state to another in response to an event.
  + **Notation**: Represented by arrows between states, typically labeled with the event name and optional guard conditions.
  + **Triggering Events**: Conditions or actions that initiate a transition, such as “Payment Approved” or “Cancel Order.”
* **Events**:
  + **Definition**: An event is a trigger that causes an object to transition from one state to another.
  + **Examples**: User actions like “Submit Form,” system actions like “Timeout,” or external events like “Receive Shipment.”
* **Actions**:
  + **Definition**: Actions are operations executed during a transition or as an entry/exit action in a state.
  + **Types**:
    - **Entry Action**: An action performed on entering a state.
    - **Exit Action**: An action performed on leaving a state.
    - **Internal Action**: An ongoing activity within the state.
* **Conditions (Guard Conditions)**:
  + **Definition**: Boolean expressions that determine whether a transition can occur.
  + **Usage**: Guard conditions must be true for the transition to take place; for instance, “if inventory > 0” might allow an order to transition from Pending to Confirmed.
* **Initial and Final States**:
  + **Initial State**: Indicates where the object lifecycle begins, shown as a filled black circle.
  + **Final State**: Represents the completion of an object’s lifecycle, depicted by a bullseye symbol (a filled circle within another circle).

### **Activity Diagram**

Activity diagrams are a crucial part of the Unified Modeling Language (UML) that help visualize workflows, processes, or activities within a system. They illustrate how different actions are connected and how a system transitions from one state to another. By providing a clear picture of both simple and complex workflows, activity diagrams facilitate understanding among developers and stakeholders regarding how various elements interact within a system.

### Activity Diagram Notations

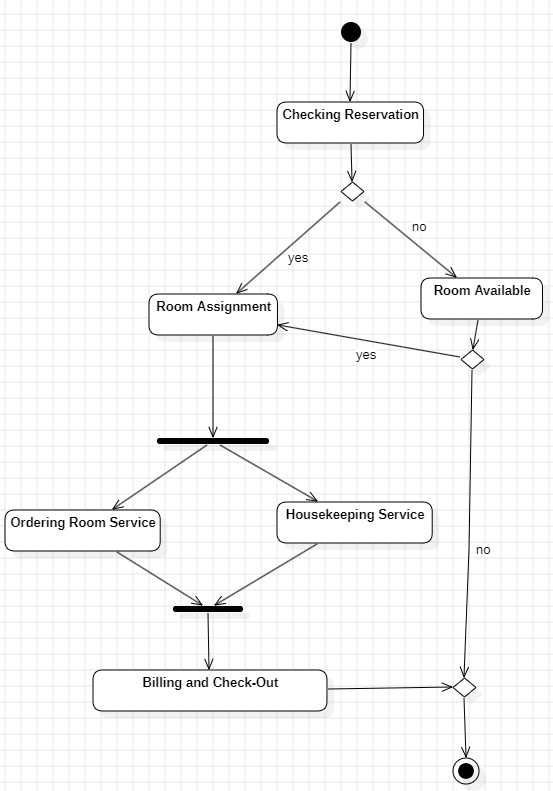


1. **Initial State**:
   * The starting point before any activity occurs is depicted using the initial state, represented by a filled black circle. A process can have only one initial state unless depicting nested activities. This state marks the entry point and the initial activity state.
2. **Action or Activity State**:
   * An activity represents the execution of an action on objects or by objects. Activities are depicted using rectangles with rounded corners, representing any action or event that takes place.
3. **Action Flow or Control Flows**:
   * These are the paths or edges used to show transitions from one activity state to another.
4. **Decision Node and Branching**:
   * A decision node is used to indicate where a decision is required before continuing the flow of control. It typically has two or more outgoing arrows, which can be labeled with conditions or guard expressions.
5. **Guard**:
   * A guard is a statement written next to a decision node on an arrow (often in square brackets) that must be true for control to follow a particular direction. Guards help define the constraints and conditions determining the flow of a process.
6. **Fork**:
   * Fork nodes support concurrent activities, allowing both activities to execute simultaneously without making a decision first. Forks are represented by a rounded solid rectangular bar, with an incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.
7. **Join**:
   * Join nodes converge concurrent activities into one. They have two or more incoming edges and one outgoing edge.
8. **Merge or Merge Event**:
   * This notation is used when activities that are not executed concurrently need to merge. It allows for merging two or more activities into one, irrespective of the path chosen to reach it.
9. **Swimlanes**:
   * Swimlanes group related activities into columns or rows, adding modularity to the activity diagram. While not mandatory, using swimlanes enhances clarity by delineating responsibilities, similar to creating functions in programming.
10. **Time Event**:
    * Represented by an hourglass, a time event indicates a stoppage in the flow for a duration, often occurring in scenarios where an event takes time to complete.
11. **Final State or End State**:
    * The final state, represented by a filled circle within another circle, indicates when a particular process or activity concludes. A system or process can have multiple final states.

**State-chart Diagram: Hotel Management System**

****

**Activity Diagram: Hotel Management System**

****

**CONCLUSION:-**

The State-chart Diagram and Activity Diagram is Made for Hotel Management

System.

**EXPERIMENT – 8A**

**AIM:-**

To Prepare Sequence Diagram & Collaboration Diagram for Library Management

System.

**REQUIREMENT:-**

StarUML.

**THEORY:-**

A Sequence Diagram is a key component of Unified Modeling Language (UML) used to visualize the interaction between objects in a sequential order. It focuses on how objects communicate with each other over time, making it an essential tool for modeling dynamic behavior in a system. Sequence diagrams illustrate object interactions, message flows, and the sequence of operations, making them valuable for understanding use cases, designing system architecture, and documenting complex processes.

### Sequence Diagram Notations

1. **Actors**  
   An actor in a UML diagram represents a type of role that interacts with the system and its objects. It is important to note that an actor is always outside the scope of the system we aim to model using the UML diagram. We use actors to depict various roles, including human users and other external subjects. We represent an actor in a UML diagram using a stick person notation. We can have multiple actors in a sequence diagram.
2. **Lifelines**  
   A lifeline is a named element that depicts an individual participant in a sequence diagram. Each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. We display a lifeline in a rectangle called a head with its name and type.
3. **Messages**  
   Communication between objects is depicted using messages, which appear in a sequential order on the lifeline.
   * We represent messages using arrows.
   * Lifelines and messages form the core of a sequence diagram.  
     Messages can be broadly classified into the following categories:
   * **Synchronous Messages**  
     A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed processing the message. The caller continues only when it knows that the receiver has processed the previous message, i.e., it receives a reply message.
     + A large number of calls in object-oriented programming are synchronous.
     + Notation: Solid arrowhead.
   * **Asynchronous Messages**  
     An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not.
     + Notation: Lined arrowhead.
4. **Create Message**  
   We use a Create message to instantiate a new object in the sequence diagram. In certain situations, a particular message call requires the creation of an object. It is represented with a dotted arrow and the word "create" labeled on it to specify that it is the Create Message symbol.
5. **Delete Message**  
   We use a Delete Message to delete an object. When an object is deallocated from memory or destroyed within the system, we use the Delete Message symbol. It represents the destruction of the occurrence of the object in the system.
   * Notation: Arrow terminating with an "X."
6. **Self-Message**  
   Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U-shaped arrow.
7. **Reply Message**  
   Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.
8. **Found Message**  
   A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an endpoint.
9. **Lost Message**  
   A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an endpoint from a lifeline.
10. **Guards**  
    To model conditions, we use guards in UML. They are used when we need to restrict the flow of messages based on certain conditions being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.

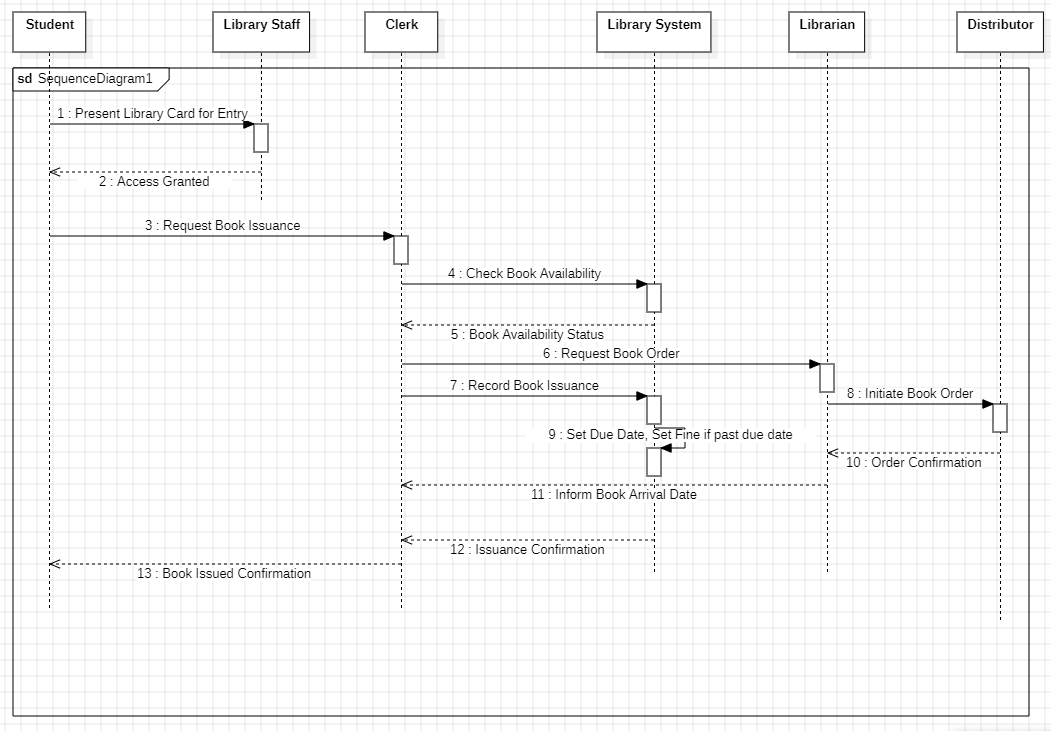
A Collaboration Diagram is a type of Interaction Diagram that visualizes the interactions and relationships between objects in a system. It shows how objects collaborate to achieve a specific task or behavior. Collaboration diagrams are used to model the dynamic behavior of a system and illustrate the flow of messages between objects during a particular scenario or use case.

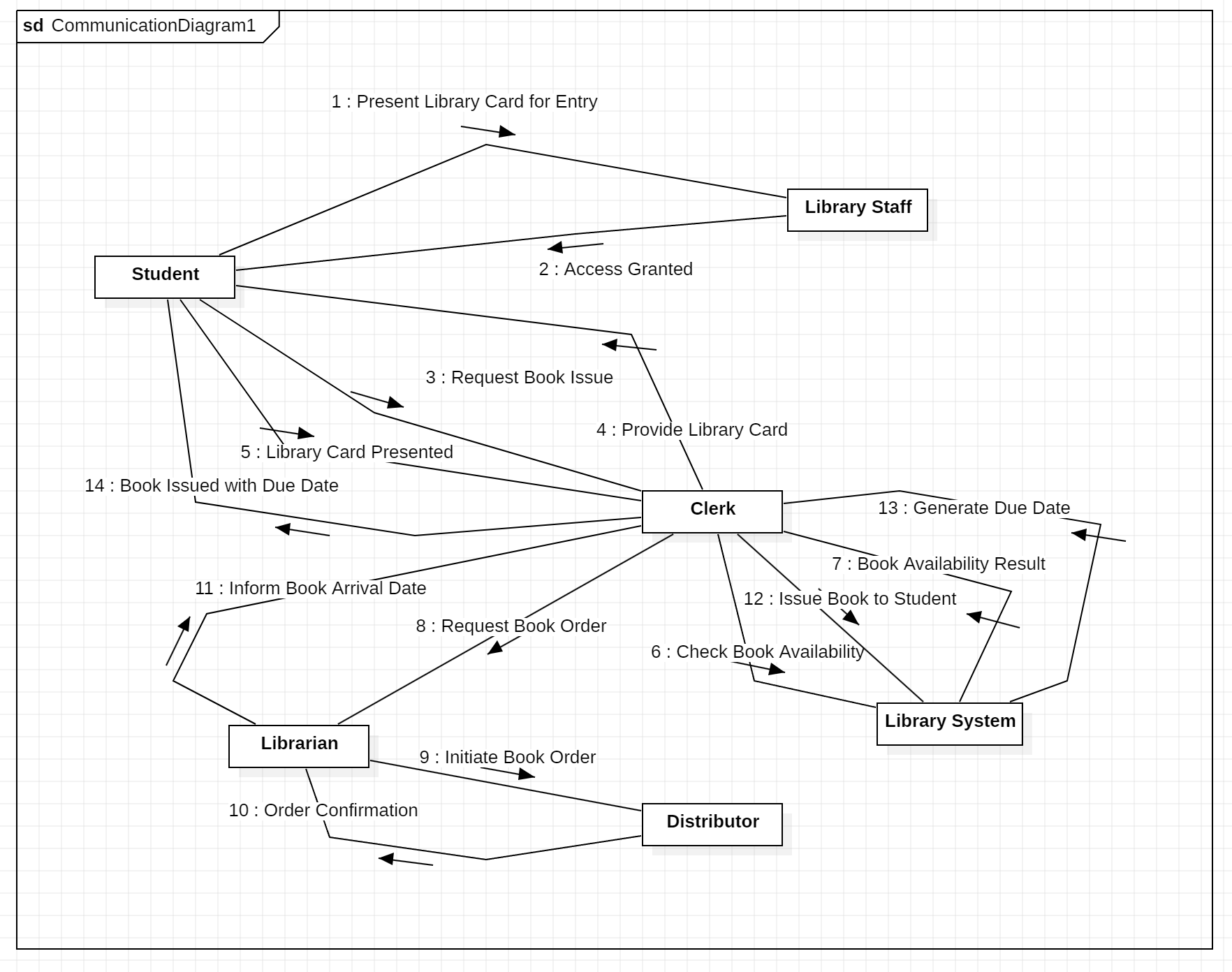
### Components and Their Notations in Collaboration Diagrams

There are several components in a Collaboration Diagram. Below are the components and their notations:

1. **Objects/Participants**  
   Objects are represented by rectangles with the object's name at the top. Each object participating in the interaction is shown as a separate rectangle in the diagram. Objects are connected by lines to indicate messages being passed between them.
2. **Multiple Objects**  
   Multiple objects are represented by rectangles, each with the object's name inside, and interactions between them are shown using arrows to indicate message flows.
3. **Actors**  
   Actors are usually shown at the top or side of the diagram. They indicate their involvement in the interactions with the system's objects or components. Actors are connected to objects through messages, showing the communication with the system.
4. **Messages**  
   Messages represent communication between objects. They are shown as arrows between objects, indicating the flow of communication. Each message may include a label indicating the type of message (e.g., method call, signal). Messages can be asynchronous (indicated by a dashed arrow) or synchronous (indicated by a solid arrow).
5. **Self-Message**  
   A self-message is a message that an object sends to itself. It represents an action or behavior that the object performs internally without involving any other objects. Self-messages are useful for modeling scenarios where an object triggers its own methods or processes.
6. **Links**  
   Links represent associations or relationships between objects. Links are shown as lines connecting objects, with optional labels to indicate the nature of the relationship. Links can be unidirectional or bidirectional, depending on the nature of the association.
7. **Return Messages**  
   Return messages represent the return value of a message. They are shown as dashed arrows with a label indicating the return value. Return messages are used to indicate that a message has been processed and a response is being sent back to the calling object.

**Sequence Diagram: Library Management System**

****

**Collaboration Diagram: Library Management System**

**CONCLUSION:-**

The Sequence Diagram and Collaboration Diagram is Made for Library Management System.

**EXPERIMENT – 8B**

**AIM:-**

To Prepare Sequence Diagram & Collaboration Diagram for Hotel Management

System.

**REQUIREMENT:-**

StarUML.

**THEORY:-**

A Sequence Diagram is a key component of Unified Modeling Language (UML) used to visualize the interaction between objects in a sequential order. It focuses on how objects communicate with each other over time, making it an essential tool for modeling dynamic behavior in a system. Sequence diagrams illustrate object interactions, message flows, and the sequence of operations, making them valuable for understanding use cases, designing system architecture, and documenting complex processes.

### Sequence Diagram Notations

1. **Actors**  
   An actor in a UML diagram represents a type of role that interacts with the system and its objects. It is important to note that an actor is always outside the scope of the system we aim to model using the UML diagram. We use actors to depict various roles, including human users and other external subjects. We represent an actor in a UML diagram using a stick person notation. We can have multiple actors in a sequence diagram.
2. **Lifelines**  
   A lifeline is a named element that depicts an individual participant in a sequence diagram. Each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. We display a lifeline in a rectangle called a head with its name and type.
3. **Messages**  
   Communication between objects is depicted using messages, which appear in a sequential order on the lifeline.
   * We represent messages using arrows.
   * Lifelines and messages form the core of a sequence diagram.  
     Messages can be broadly classified into the following categories:
   * **Synchronous Messages**  
     A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed processing the message. The caller continues only when it knows that the receiver has processed the previous message, i.e., it receives a reply message.
     + A large number of calls in object-oriented programming are synchronous.
     + Notation: Solid arrowhead.
   * **Asynchronous Messages**  
     An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not.
     + Notation: Lined arrowhead.
4. **Create Message**  
   We use a Create message to instantiate a new object in the sequence diagram. In certain situations, a particular message call requires the creation of an object. It is represented with a dotted arrow and the word "create" labeled on it to specify that it is the Create Message symbol.
5. **Delete Message**  
   We use a Delete Message to delete an object. When an object is deallocated from memory or destroyed within the system, we use the Delete Message symbol. It represents the destruction of the occurrence of the object in the system.
   * Notation: Arrow terminating with an "X."
6. **Self-Message**  
   Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U-shaped arrow.
7. **Reply Message**  
   Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.
8. **Found Message**  
   A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an endpoint.
9. **Lost Message**  
   A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an endpoint from a lifeline.
10. **Guards**  
    To model conditions, we use guards in UML. They are used when we need to restrict the flow of messages based on certain conditions being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.

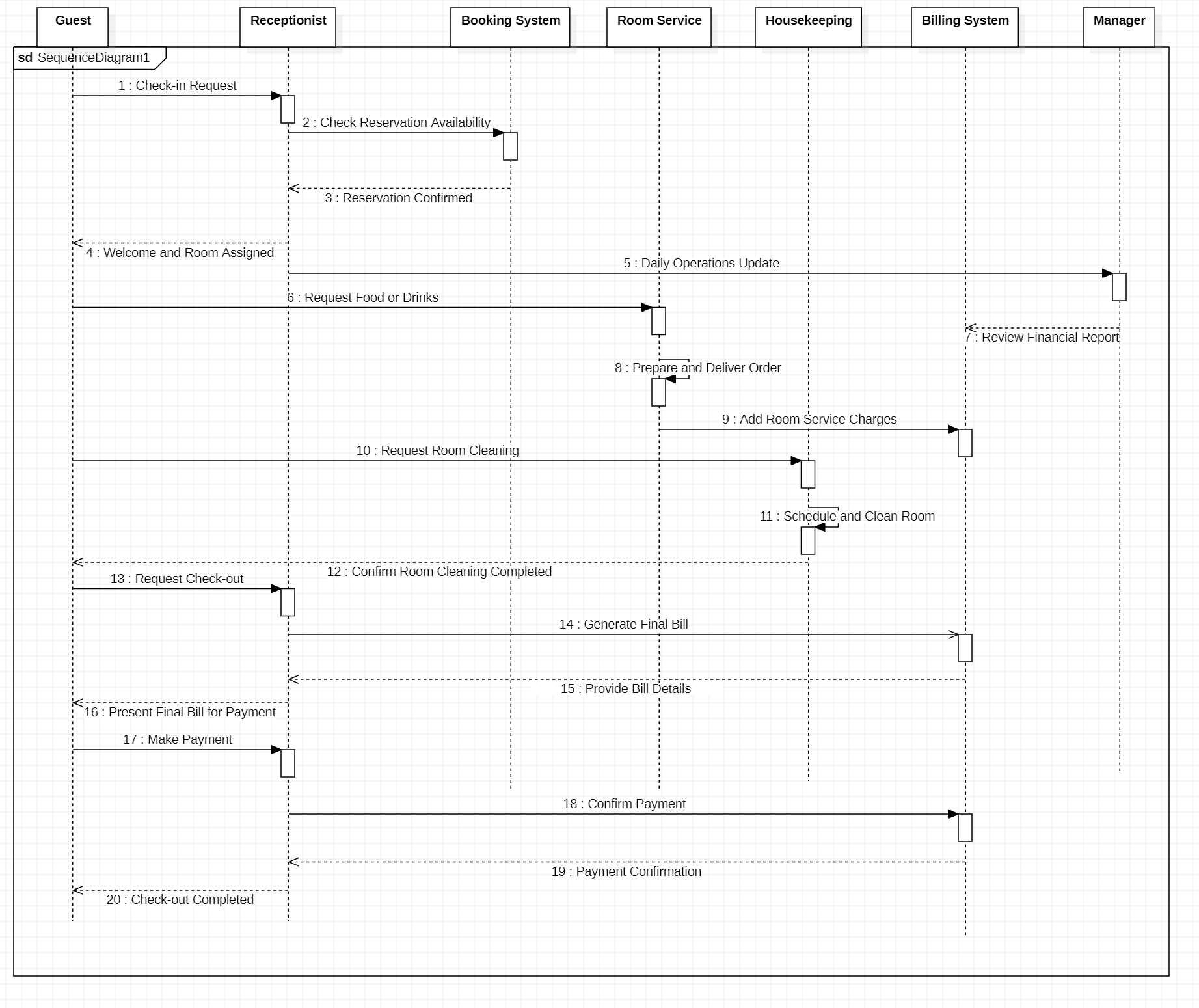
A Collaboration Diagram is a type of Interaction Diagram that visualizes the interactions and relationships between objects in a system. It shows how objects collaborate to achieve a specific task or behavior. Collaboration diagrams are used to model the dynamic behavior of a system and illustrate the flow of messages between objects during a particular scenario or use case.

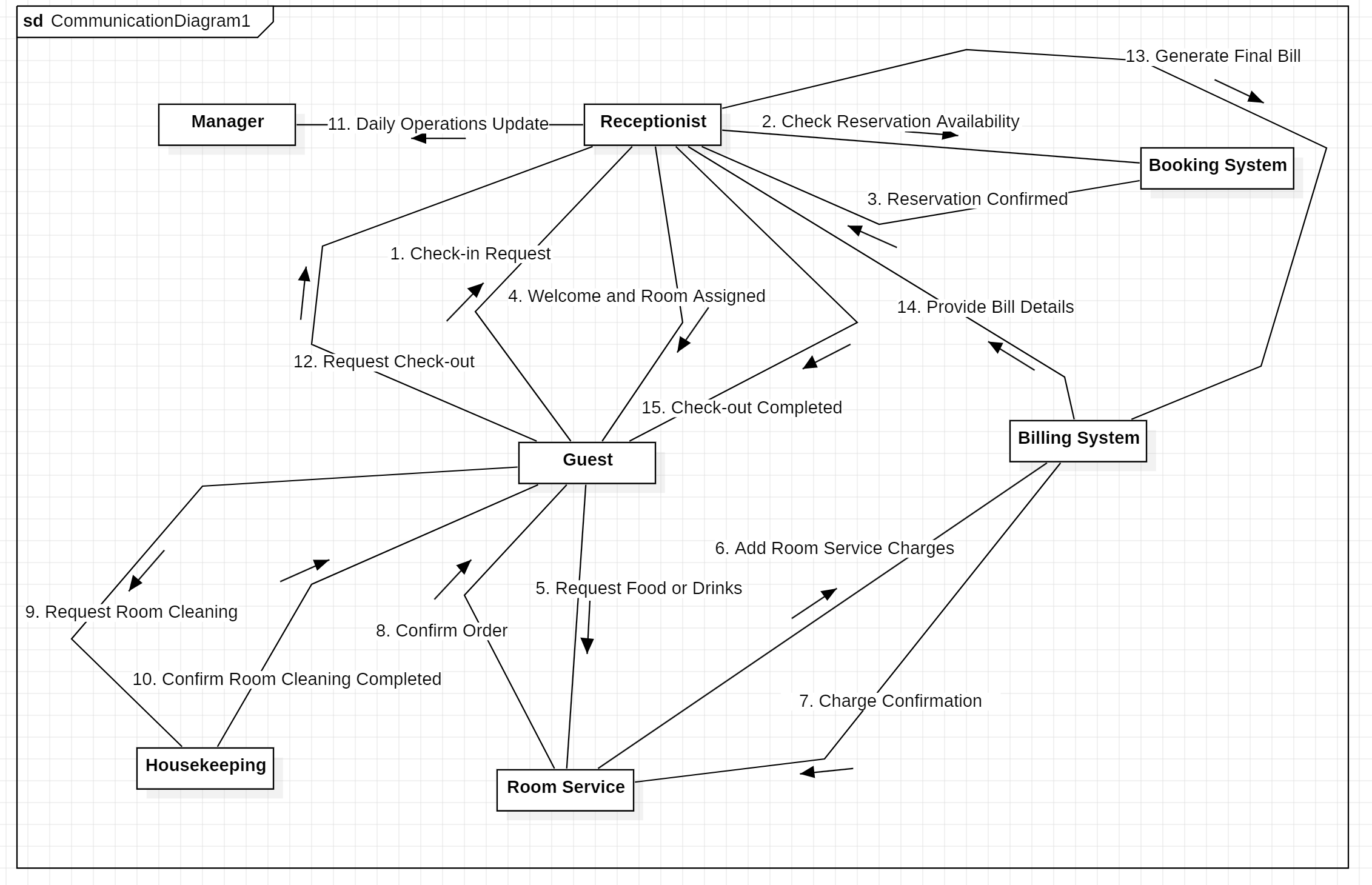
### Components and Their Notations in Collaboration Diagrams

There are several components in a Collaboration Diagram. Below are the components and their notations:

1. **Objects/Participants**  
   Objects are represented by rectangles with the object's name at the top. Each object participating in the interaction is shown as a separate rectangle in the diagram. Objects are connected by lines to indicate messages being passed between them.
2. **Multiple Objects**  
   Multiple objects are represented by rectangles, each with the object's name inside, and interactions between them are shown using arrows to indicate message flows.
3. **Actors**  
   Actors are usually shown at the top or side of the diagram. They indicate their involvement in the interactions with the system's objects or components. Actors are connected to objects through messages, showing the communication with the system.
4. **Messages**  
   Messages represent communication between objects. They are shown as arrows between objects, indicating the flow of communication. Each message may include a label indicating the type of message (e.g., method call, signal). Messages can be asynchronous (indicated by a dashed arrow) or synchronous (indicated by a solid arrow).
5. **Self-Message**  
   A self-message is a message that an object sends to itself. It represents an action or behavior that the object performs internally without involving any other objects. Self-messages are useful for modeling scenarios where an object triggers its own methods or processes.
6. **Links**  
   Links represent associations or relationships between objects. Links are shown as lines connecting objects, with optional labels to indicate the nature of the relationship. Links can be unidirectional or bidirectional, depending on the nature of the association.
7. **Return Messages**  
   Return messages represent the return value of a message. They are shown as dashed arrows with a label indicating the return value. Return messages are used to indicate that a message has been processed and a response is being sent back to the calling object.

**Sequence Diagram: Hotel Management System**

****

**Collaboration Diagram: Hotel Management System**

**CONCLUSION:-**

The Sequence Diagram and Collaboration Diagram is Made for Hotel Management System.

**EXPERIMENT – 9A**

**AIM:-**

To Prepare Component Diagram for Library Management System.

**REQUIREMENT:-**

StarUML.

**THEORY:-**

Component diagrams are used to visualize the organization of system components and the dependency relationships between them. They provide a high-level view of the components within a system. The components can be:

* **Software components** such as databases or user interfaces.
* **Hardware components** such as circuits, microchips, or devices.
* **Business units** such as suppliers, payroll, or shipping.

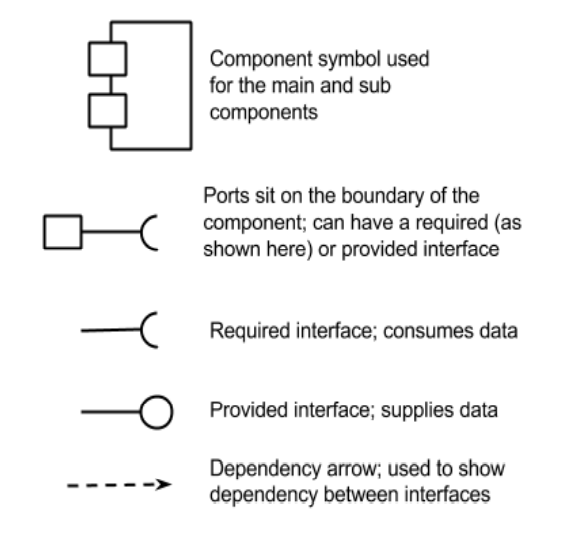
### Uses of Component Diagrams

* Used in Component-Based Development to describe systems with Service-Oriented Architecture.
* Show the structure of the code itself.
* Focus on the relationships between components while hiding specification details.
* Help communicate and explain the functions of the system being built to stakeholders.

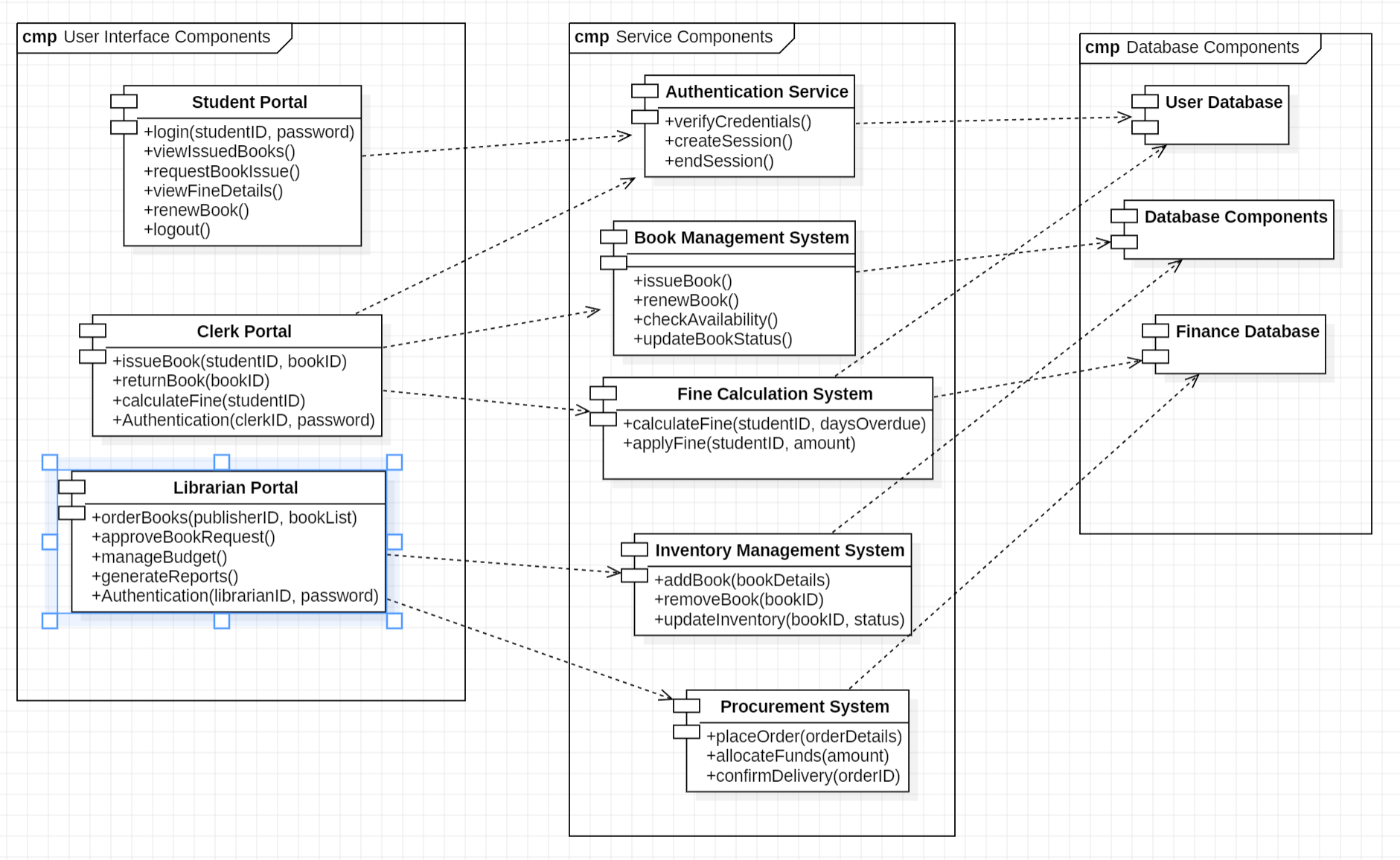
### Component Diagram Symbols

Here are the common notations used to draw a component diagram:

* **Component**  
  There are three ways the component symbol can be used:
  1. **Rectangle with the component stereotype** (the text <<component>>). This stereotype is usually placed above the component name to avoid confusion with a class icon.
  2. **Rectangle with the component icon** in the top right corner and the name of the component.
  3. **Rectangle with the component icon** and the component stereotype.
* **Provided Interface and Required Interface**  
  Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component’s required interface (represented with a semi-circle and a solid line) with the provided interface (represented with a circle and a solid line) of another component. This indicates that one component is providing a service that another component requires.
* **Port**  
  A port (represented by a small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.
* **Dependencies**  
  You can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface). Alternatively, you can use a dependency arrow to indicate the relationship between two components.

****

**Component Diagram: Library Management System**

****

**CONCLUSION:-**

The Component Diagram is Made for Library Management System.

**EXPERIMENT – 9B**

**AIM:-**

To Prepare Component Diagram for Hotel Management System.

**REQUIREMENT:-**

StarUML.

**THEORY:-**

Component diagrams are used to visualize the organization of system components and the dependency relationships between them. They provide a high-level view of the components within a system. The components can be:

* **Software components** such as databases or user interfaces.
* **Hardware components** such as circuits, microchips, or devices.
* **Business units** such as suppliers, payroll, or shipping.

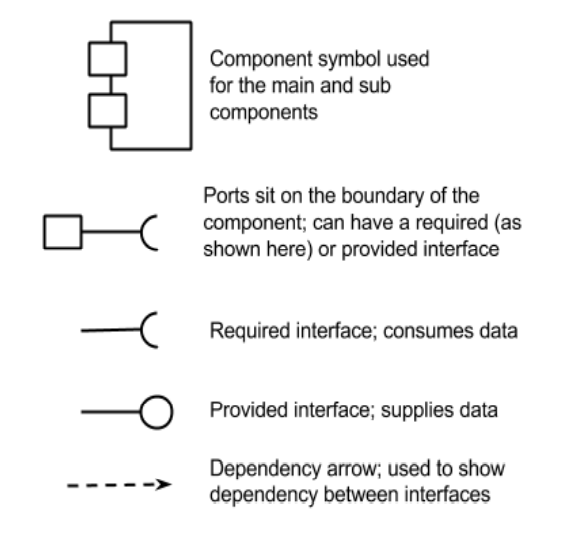
### Uses of Component Diagrams

* Used in Component-Based Development to describe systems with Service-Oriented Architecture.
* Show the structure of the code itself.
* Focus on the relationships between components while hiding specification details.
* Help communicate and explain the functions of the system being built to stakeholders.

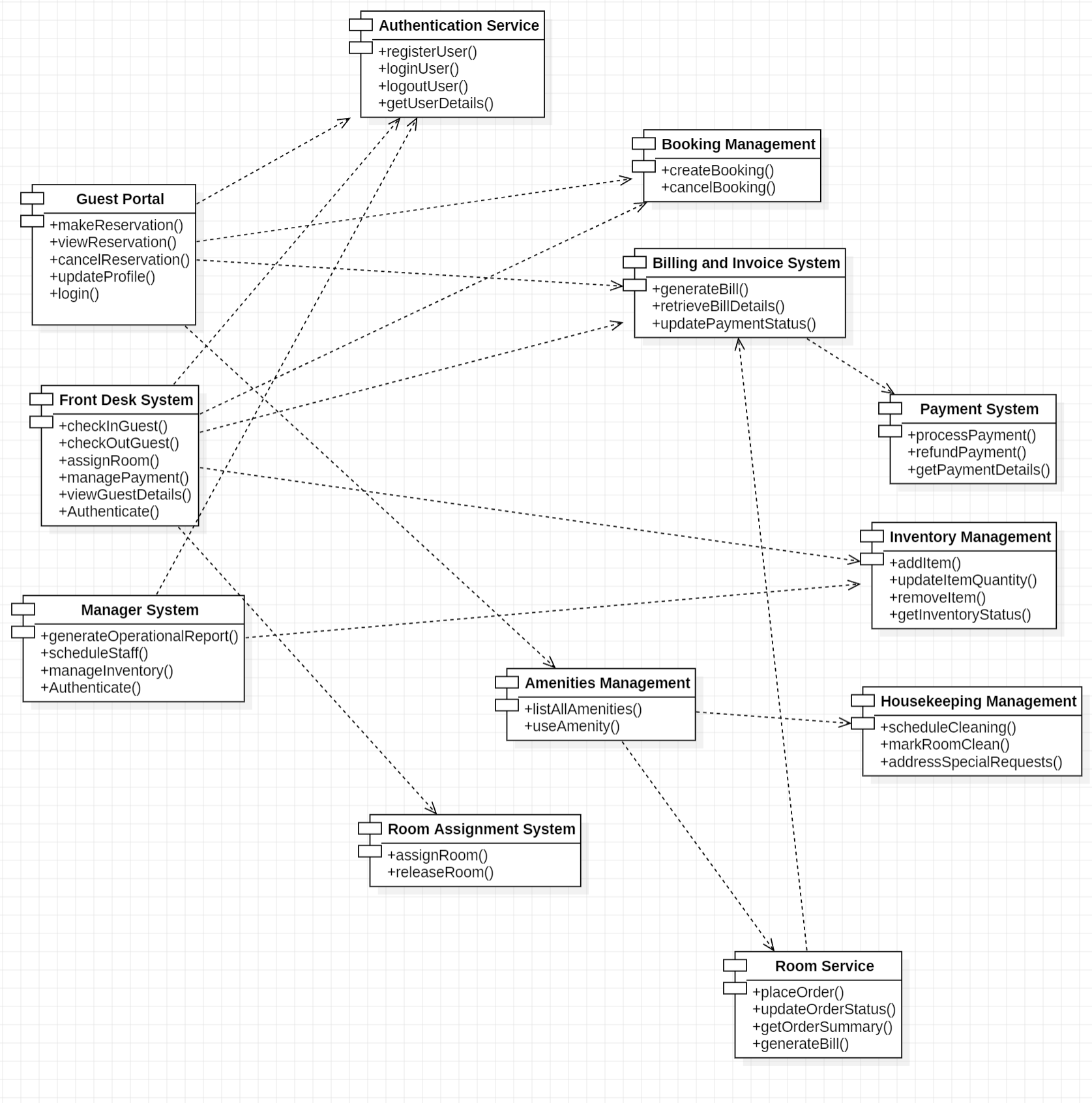
### Component Diagram Symbols

Here are the common notations used to draw a component diagram:

* **Component**  
  There are three ways the component symbol can be used:
  1. **Rectangle with the component stereotype** (the text <<component>>). This stereotype is usually placed above the component name to avoid confusion with a class icon.
  2. **Rectangle with the component icon** in the top right corner and the name of the component.
  3. **Rectangle with the component icon** and the component stereotype.
* **Provided Interface and Required Interface**  
  Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component’s required interface (represented with a semi-circle and a solid line) with the provided interface (represented with a circle and a solid line) of another component. This indicates that one component is providing a service that another component requires.
* **Port**  
  A port (represented by a small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.
* **Dependencies**  
  You can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface). Alternatively, you can use a dependency arrow to indicate the relationship between two components.

****

**Component Diagram: Hotel Management System**

****

**CONCLUSION:-**

The Component Diagram is Made for Hotel Management System.

**EXPERIMENT – 10A**

**AIM:-**

To Prepare Deployment Diagram for Library Management System.

**REQUIREMENT:-**

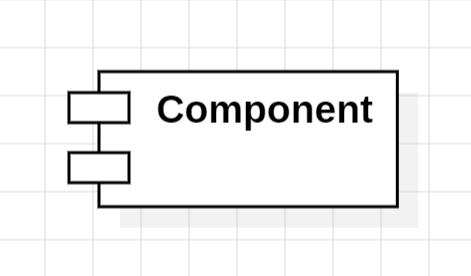
StarUML.

**THEORY:-**

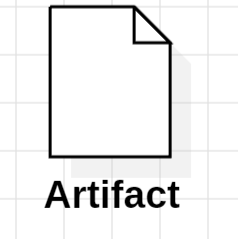
The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships. It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths. It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

Below are the components and their notations used in deployment diagrams:

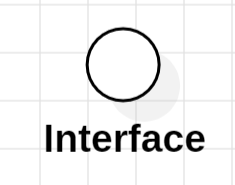
1. **Component**  
   A component represents a modular and reusable part of a system, typically implemented as a software module, class, or package. It encapsulates its behavior and data and can be deployed independently.



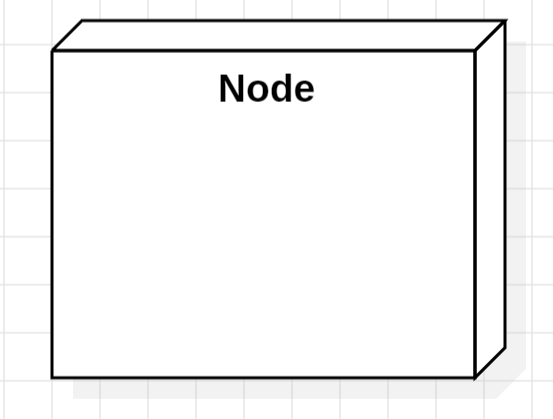
1. **Artifact**  
   An artifact represents a physical piece of information or data that is used or produced in the software development process. This can include source code files, executables, documents, libraries, configuration files, or any other item.



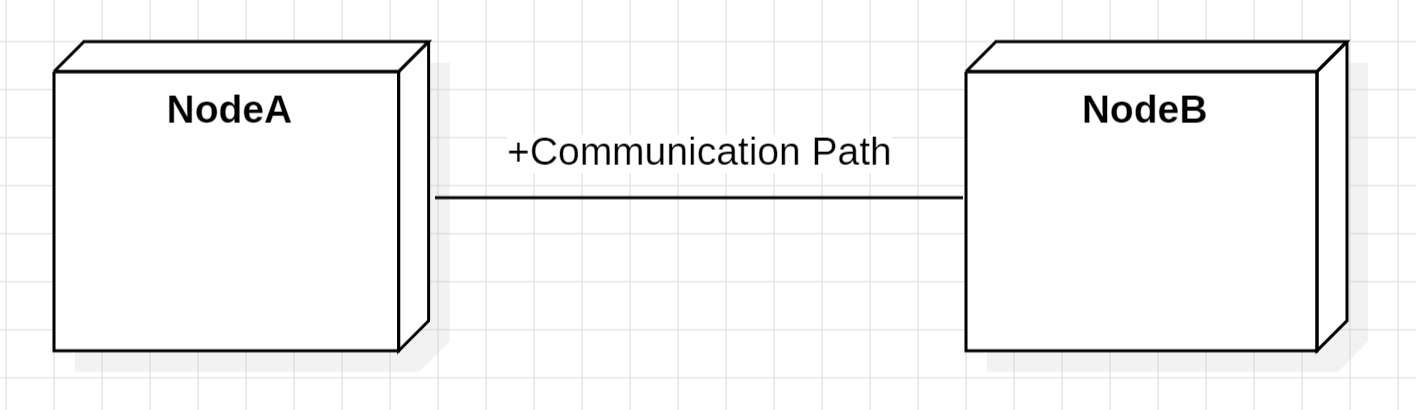
1. **Interface**  
   An interface defines a contract specifying the methods or operations that a component must implement. It represents a point of interaction between different components or subsystems.



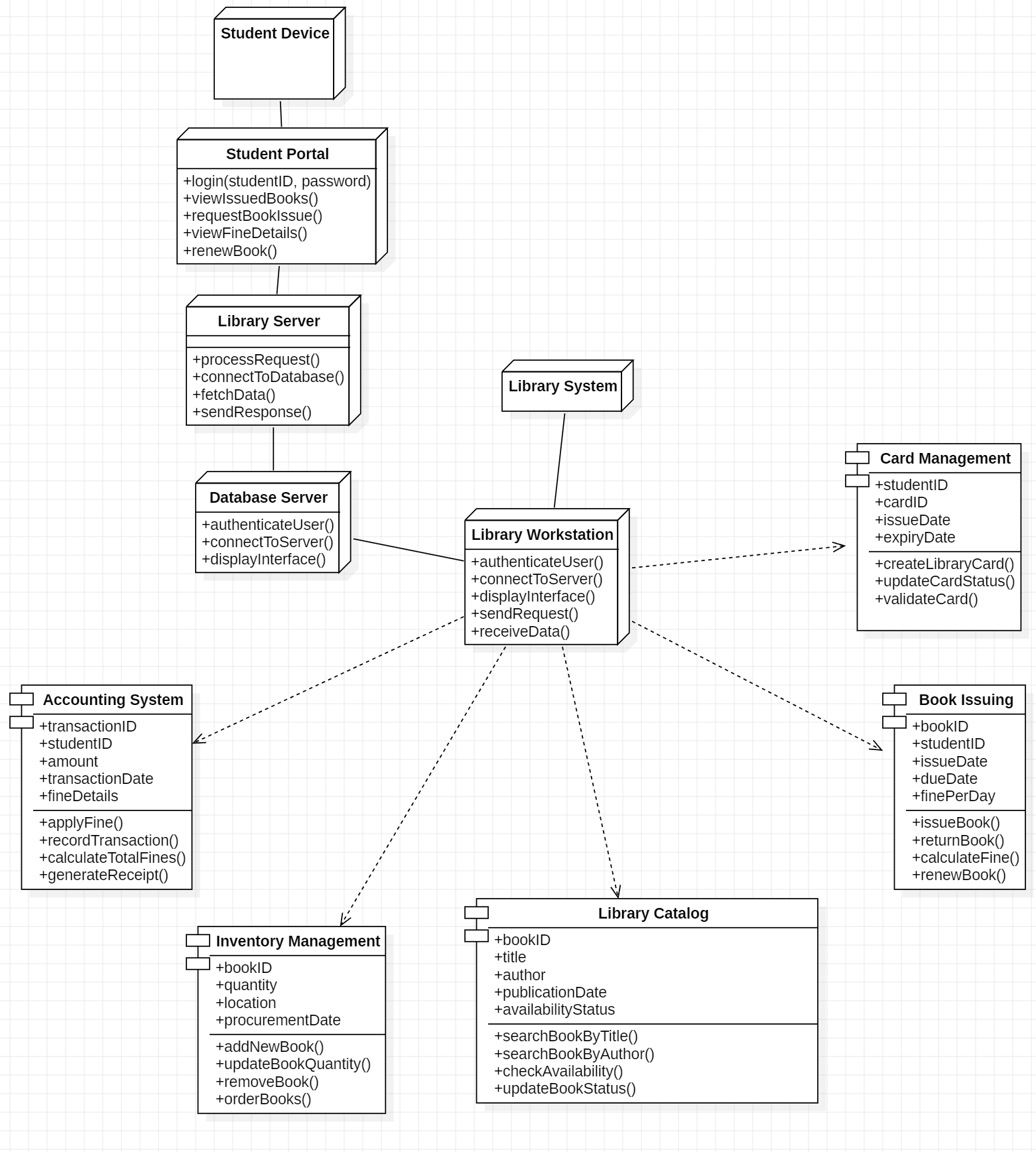
1. **Node**  
   A node represents a physical or computational resource, such as a hardware device, server, workstation, or computing resource, on which software components can be deployed or executed.

****

1. **Communication Path**  
   A communication path is represented by a straight line that indicates communication between two device nodes. Dashed lines in deployment diagrams represent relationships or dependencies between elements, indicating that one element is related to or dependent on another.



**Deployment Diagram: Library Management System**

****

**CONCLUSION:-**

The Deployment Diagram is Made for Library Management System.

**EXPERIMENT – 10B**

**AIM:-**

To Prepare Deployment Diagram for Hotel Management System.

**REQUIREMENT:-**

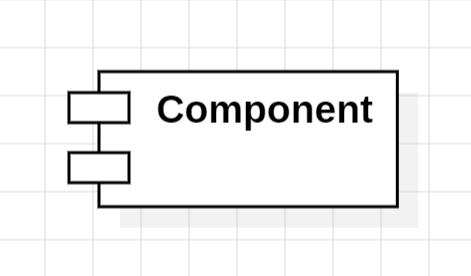
StarUML.

**THEORY:-**

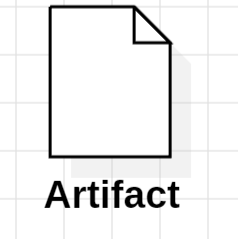
The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships. It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths. It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

Below are the components and their notations used in deployment diagrams:

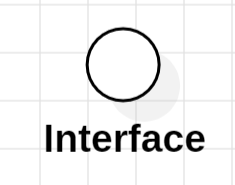
1. **Component**  
   A component represents a modular and reusable part of a system, typically implemented as a software module, class, or package. It encapsulates its behavior and data and can be deployed independently.



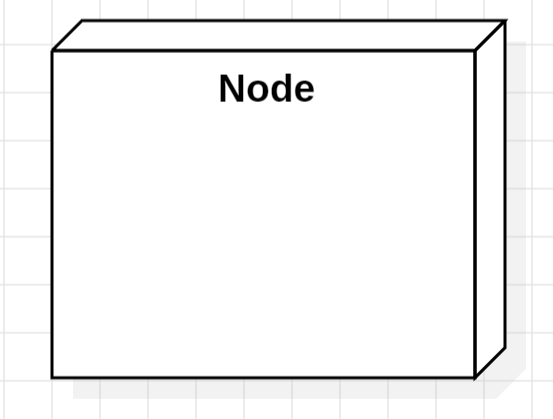
1. **Artifact**  
   An artifact represents a physical piece of information or data that is used or produced in the software development process. This can include source code files, executables, documents, libraries, configuration files, or any other item.



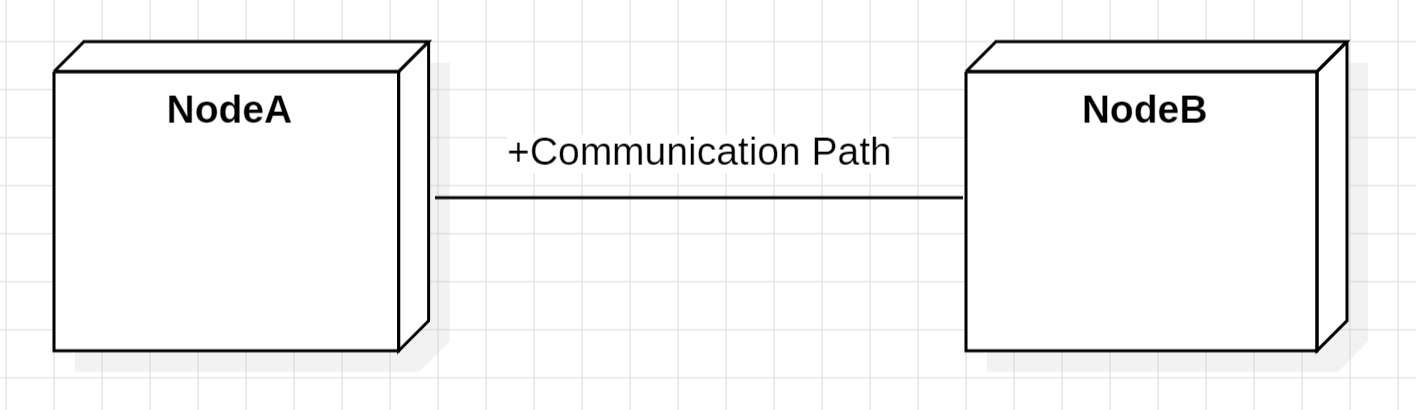
1. **Interface**  
   An interface defines a contract specifying the methods or operations that a component must implement. It represents a point of interaction between different components or subsystems.



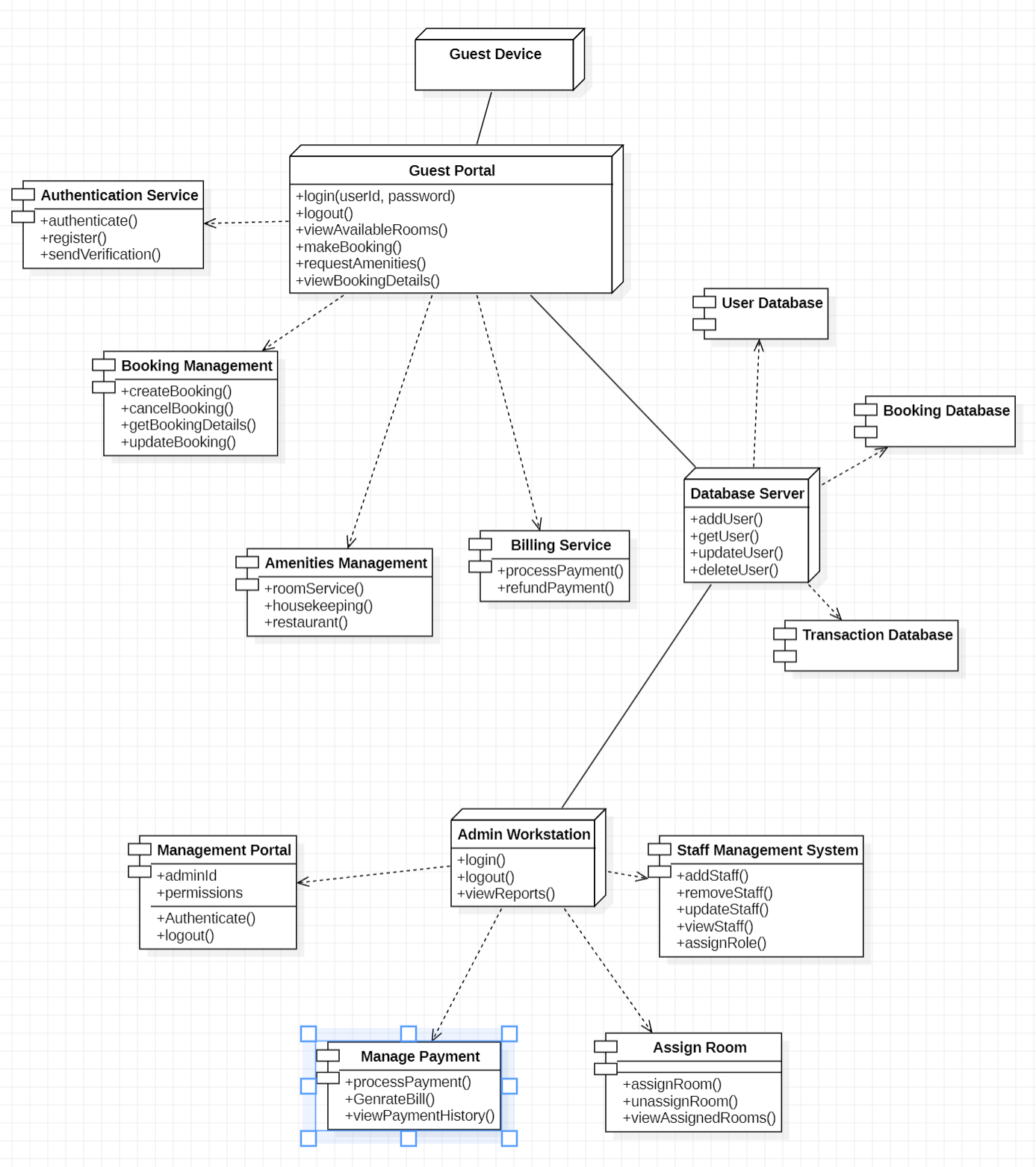
1. **Node**  
   A node represents a physical or computational resource, such as a hardware device, server, workstation, or computing resource, on which software components can be deployed or executed.

****

1. **Communication Path**  
   A communication path is represented by a straight line that indicates communication between two device nodes. Dashed lines in deployment diagrams represent relationships or dependencies between elements, indicating that one element is related to or dependent on another.



**Deployment Diagram: Hotel Management System**

****

**CONCLUSION:-**

The Deployment Diagram is Made for Hotel Management System.