RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

HARDWORK DISCIPLINE
REC

RAJALAKSHMI
ENGINEERING COLLEGE

CB23332
SOFTWARE ENGINEERING LAB

Laboratory Record Note Book

Name : . . . .

Year / Branch / Section : . . . .

Register No. : . . . .

Semester : . . . .

Academic Year : . . . .

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

RAJALAKSHMI NAGAR, THANDALAM – 602-105

BONAFIDE CERTIFICATE

NAME:______________________REGISTER NO.: _______________

ACADEMIC YEAR: 2024-25 **SEMESTER:** III **BRANCH:_________**B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on ____________

Internal Examiner External Examiner

INDEX

1. PREPARING PROBLEM STATEMENT

Aim :

The aim of the Mall Directory System project is to develop an intuitive and efficient digital solution that helps customers easily locate unique items within a shopping mall and assists store managers by providing timely restocking alerts, enhancing both the shopping experience and inventory management.

Algorithm :

1. **Define the Problem:**
   - Clearly state the challenges that the mall and its customers face in locating items and managing inventory levels.
   - Avoid specifying any particular solutions, focusing instead on the high-level needs of both customers and mall management.
2. **Gather Contextual Information:**
   - Provide background information on the mall environment, item diversity, and how the current lack of a system impacts both shoppers and store management.
   - Identify primary stakeholders, including customers and store managers, and explain the problem's impact on their experiences.
3. **State the Objective:**
   - Outline the main objectives, such as simplifying item location for customers and optimizing restocking alerts for managers.
   - Specify measurable outcomes if possible, such as increased customer satisfaction or reduced stockout occurrences.
4. **Explain the Relevance:**
   - Emphasize why a mall directory system with real-time item location and inventory tracking is essential.
   - Outline how this system can enhance customer convenience and improve overall mall operations.
5. **Detail the Requirements:**
   - Include initial requirements that stakeholders might expect, such as a user-friendly interface, accurate item location, and timely restocking notifications.
   - Define the project's scope boundaries to align with realistic goals and functionalities.

INPUTS:

The initial input for the requirements engineering phase comes from a high-level problem statement based on customer and manager feedback. Key points include:

- An overview of the current challenges customers and managers face in item location and stock management.
- Expectations for a system that addresses these pain points and delivers a streamlined shopping experience.

SAMPLE OUTPUT

Stakeholder Problem Statement:

Shoppers at [XYZ Mall] often struggle to find specific items across different stores, causing frustration and potentially driving customers away. Additionally, store managers face difficulties in keeping track of stock levels and often encounter stockouts without timely alerts, affecting sales and customer satisfaction.

Problem:

Customers at XYZ Mall cannot efficiently locate specific items across the numerous stores, affecting their shopping experience. Additionally, store managers lack a system to monitor and receive alerts for low stock, leading to missed sales opportunities.

Background:

XYZ Mall houses multiple stores offering a wide range of items. Currently, customers need to search manually or seek assistance to find specific products, which is time-consuming and impacts satisfaction. The absence of a centralized directory and stock monitoring system also means that managers lack real-time insights into inventory, leading to frequent stock shortages.

Relevance:

A mall directory system that provides item locations in real time will improve the shopping experience, making it easier for customers to find what they need. For managers, an automated restocking alert system would facilitate better inventory management, ensuring products are readily available and reducing missed sales due to stockouts.

Objectives:

- Design a customer-friendly directory system that enables shoppers to locate unique items easily.
- Develop an automated restocking alert feature for store managers, improving stock availability.
- Enhance overall mall operations and customer satisfaction through efficient item navigation and inventory control.

Result :

The problem statement was written successfully by following the steps described above.

Ex.NO. 2 : Write the software requirement specification document

Aim :

To design and develop a user-friendly Mall Directory System that allows customers to efficiently locate unique items within a shopping mall and enables store managers to receive timely restocking alerts, improving customer satisfaction and operational productivity.

Algorithm :

The SRS will address the following aspects:

1. **Functionality**: What the system should do.
2. **External Interfaces**: How the system interacts with users, hardware, and other systems.
3. **Performance**: The system's speed, availability, and response time.
4. **Attributes**: Factors like portability, maintainability, and security.
5. **Design Constraints**: Standards, implementation languages, and environmental requirements.

1. INTRODUCTION

1.1 **Purpose**

The purpose of this document is to outline the requirements for developing a mall directory system that helps customers locate items across different stores and alerts managers about restocking needs.

1.2 **Document Conventions**

- DB: Database
- ER: Entity Relationship
- POS: Point of Sale System

1.3 **Intended Audience and Reading Suggestions**

This project is intended for mall management, store owners, and developers. It provides a blueprint for system development to address the needs of both customers and store managers.

1.4 **Project Scope**

The mall directory system aims to improve the shopping experience by allowing customers to locate specific items easily and enabling managers to receive timely restocking alerts. The system will be deployed across all stores in the mall and accessible to customers via kiosks or a mobile application.

1.5 **References**

- Fundamentals of Software Engineering by Rajib Mall
- https://mall-directory.com/sample_srs

2. OVERALL DESCRIPTION

2.1 **Product Perspective**

The mall directory system is a distributed application that connects with store databases and POS systems to track item availability, location, and stock levels.

- **Item Location**: Allows customers to search for specific items and displays the store locations where they are available.
- **Inventory Tracking**: Monitors stock levels and notifies managers when items require restocking.

2.2 **Product Features**

- Real-time item location tracking.
- Automated restocking notifications.
- User-friendly interface for both customers and managers.

2.3 **User Classes and Characteristics**

- **Customers**: Can search for items by name, category, or store and view their locations within the mall.
- **Managers**: Receive alerts for low-stock items and have access to inventory data for effective stock management.

2.4 **Operating Environment**

- **Database**: SQL-based database for managing inventory and item location data.
- **Platform**: Web-based kiosks and mobile application.
- **Operating System**: Compatible with Windows and Android/iOS for mobile.

2.5 **Design and Implementation Constraints**

- Use of standard SQL queries for database access.
- Consistent UI design across kiosk and mobile interfaces.
- Compliance with data security policies for customer data.

2.6 **Assumptions and Dependencies**

- The system assumes that each store has a POS system integrated with the mall directory.
- The application will rely on stable internet connectivity for real-time updates.

3. SYSTEM FEATURES

3.1 **Description and Priority**

- **Item Locator** (High): Enables customers to locate unique items.
- **Restocking Alerts** (High): Notifies managers about low-stock items.

3.2 **Stimulus/Response Sequences**

- **Customer Search**: Users enter an item name or category; the system displays the item location and stock availability.
- **Restocking Notification**: When an item reaches the restocking threshold, an alert is sent to the manager.

3.3 **Functional Requirements**

- **Search Function**: Allows users to search for items by keywords or categories.
- **Stock Monitoring**: Tracks item availability and alerts managers.
- **Manager Dashboard**: Displays inventory levels and alerts for items needing restocking.

4. EXTERNAL INTERFACE REQUIREMENTS

4.1 **User Interfaces**

- **Front-end Software**: Web-based application with a responsive design.
- **Back-end Software**: SQL database for inventory tracking.

4.2 **Hardware Interfaces**

- Kiosks with touchscreens located throughout the mall.

- Integration with store POS systems.

4.3 **Software Interfaces**

- **Operating System**: Compatible with Windows, Android, and iOS.
- **Database**: SQL database to store inventory data.
- **Programming Language**: JavaScript, React for front-end, and Node.js for back-end.

4.4 **Communication Interfaces**

- Supports HTTP/HTTPS protocols for secure data exchange.

5. NONFUNCTIONAL REQUIREMENTS

5.1 **Performance Requirements**

- **Response Time**: Less than 2 seconds for search results.
- **Availability**: 99% uptime to ensure system reliability.

5.2 **Maintainability**

- System should support regular updates for new product listings and security patches.

5.3 **Usability**

- Simple, intuitive interface for easy navigation by customers and managers.

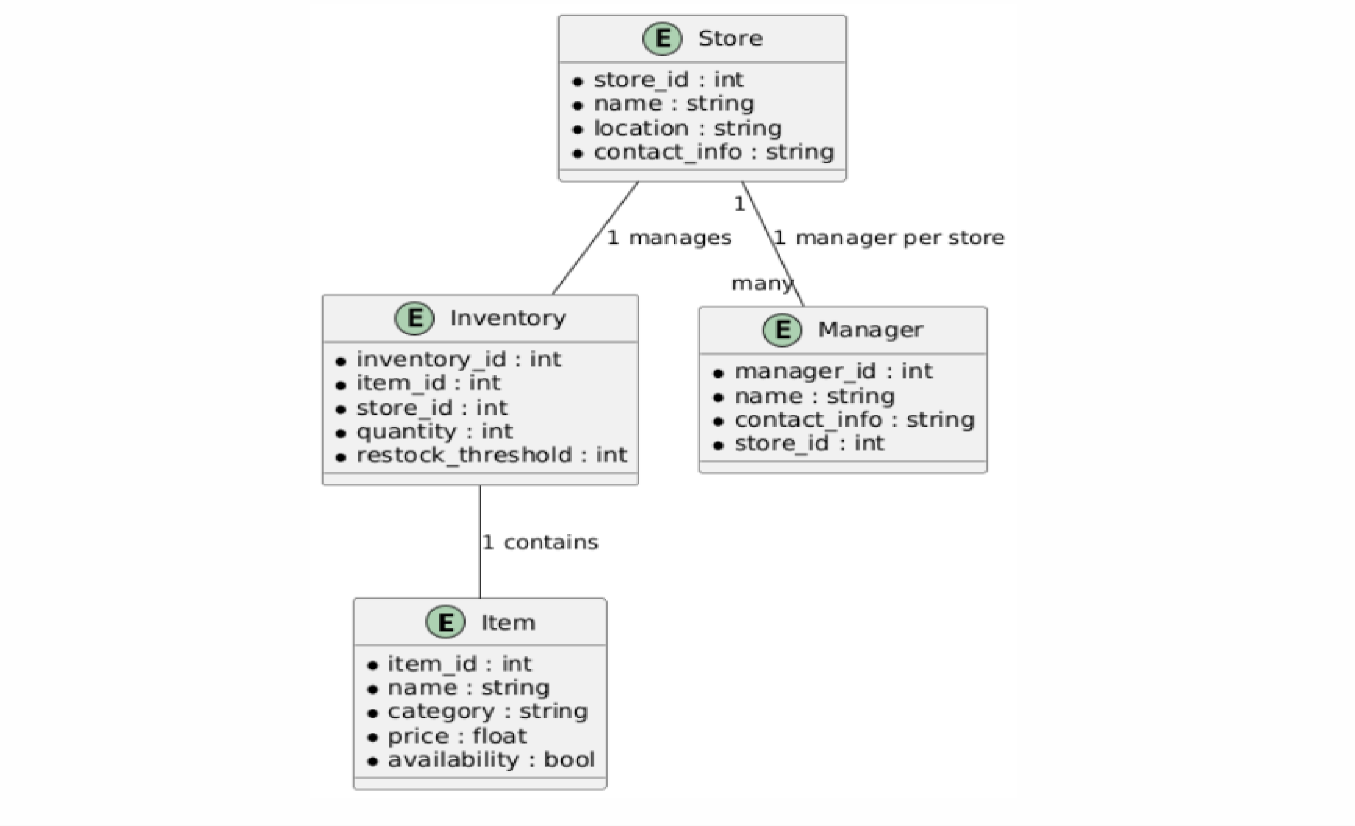
5.4 **Reliability and Recovery**

- System should be able to recover from any disruptions with minimal downtime.

5.5 **Security**

- User data protection with SSL encryption.

DIAGRAM



Result :

The SRS was made successfully by following the steps described above.

3. ENTITY RELATIONSHIP MODEL

Aim :

To Draw the Entity Relationship Diagram for mall directory system.

Algorithm :

1. **Identify Entities**: Determine the primary entities involved in the system, such as Item, Category, Location, Customer, and Manager.
2. **Define Attributes**: Specify relevant attributes for each entity to capture essential data.
3. **Establish Relationships**: Determine relationships between entities (e.g., Items are located in Locations, Managers receive alerts for restocking).
4. **Set Cardinality**: Define the cardinality for each relationship (one-to-one, one-to-many, many-to-many).
5. **Draw ER Diagram**: Use a tool like PlantUML to create the ER diagram, illustrating entities, attributes, and relationships.
6. **Review and Refine**: Validate the ERD with project requirements, adjusting as needed to ensure alignment with functional specifications.

Input :

1. Customer Information

- **customerId**: Unique identifier for each customer (Integer)
- **name**: Customer's name (String)
- **contact**: Customer's contact information, such as phone number or email (String)

2. Manager Information

- **managerId**: Unique identifier for each manager (Integer)
- **name**: Manager's name (String)
- **email**: Manager's email address (String)
- **phone**: Manager's contact number (String)

3. Item Details

- **itemId**: Unique identifier for each item (Integer)
- **itemName**: Name or description of the item (String)
- **stockLevel**: Current stock level of the item (Integer)
- **restockThreshold**: Minimum stock level that triggers a restocking alert (Integer)
- **price**: Price of the item (Float)

4. Category Details

- **categoryId**: Unique identifier for each category (Integer)
- **categoryName**: Name of the category (e.g., Electronics, Apparel) (String)

5. Location Details

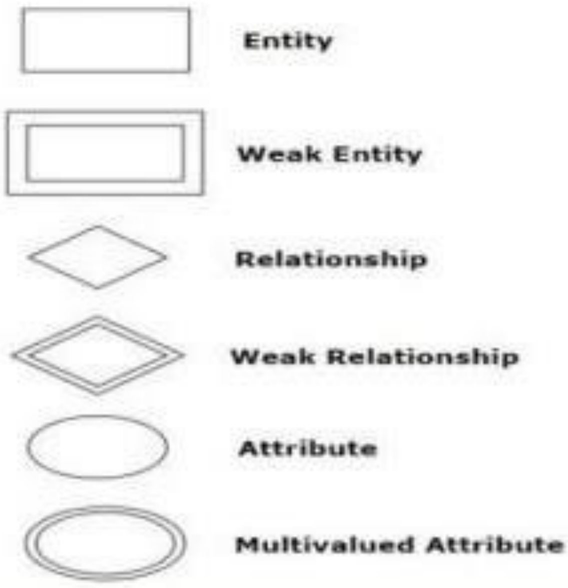
- **locationId**: Unique identifier for each location within the mall (Integer)
- **locationName**: Name or description of the location (e.g., Store A, Kiosk B) (String)
- **floor**: Floor number where the location is situated (Integer)

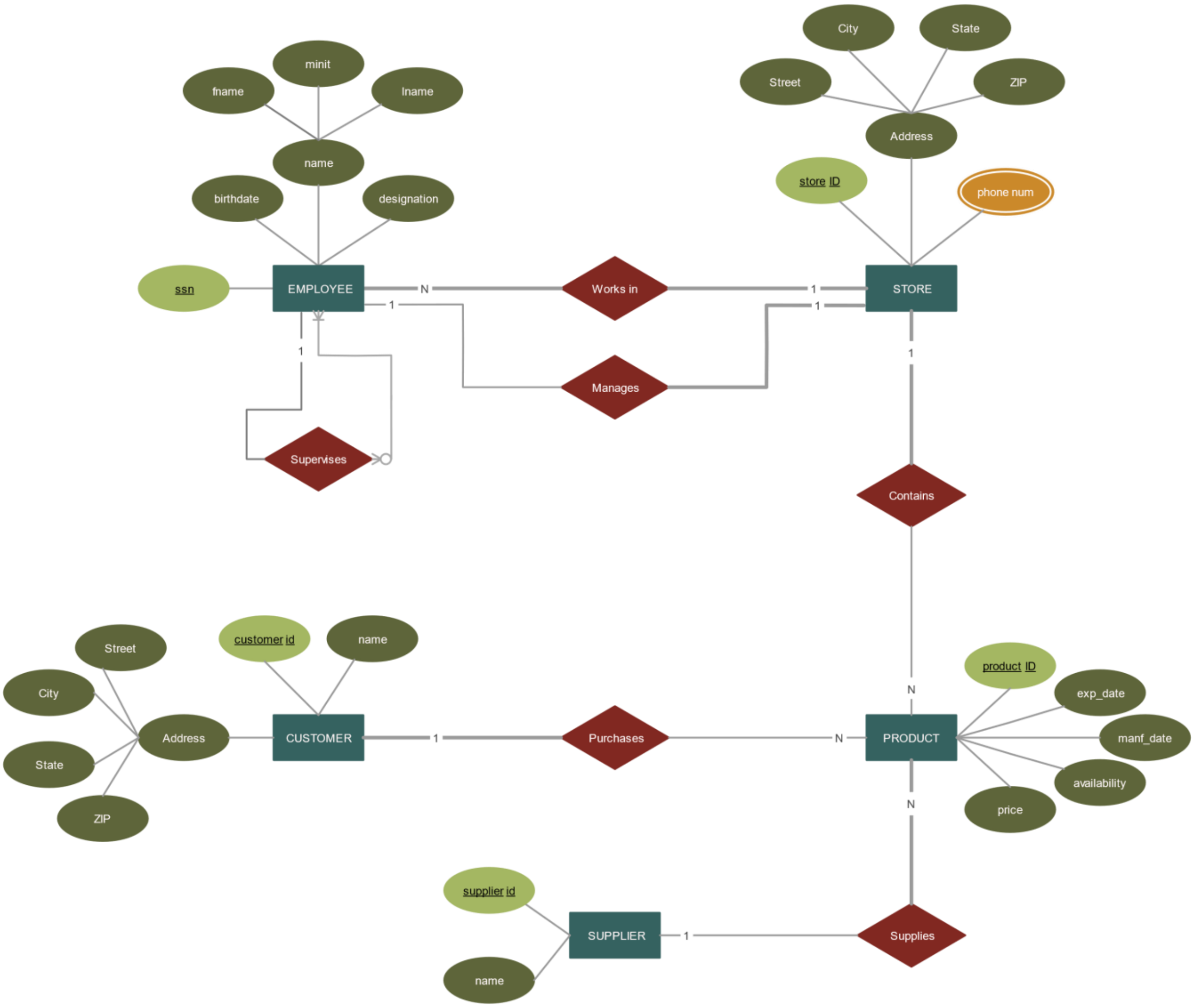
6. Restocking Alert

- **alertId**: Unique identifier for each restocking alert (Integer)
- **alertDate**: Date the restocking alert was generated (Date)
- **status**: Status of the alert (e.g., Pending, Resolved) (String)

ER DIAGRAM:

SYMBOLS:





Result: The entity relationship diagram was made successfully by following the steps described above.

4. DATA FLOW DIAGRAM

Aim :

To Draw the Data Flow Diagram for MALL DIRECTORY SYSTEM and List the Modules in the Application.

Algorithm :

1. Identify External Entities: Determine the main users interacting with the system (e.g., Customers, Managers).
2. Define Processes: Outline the primary functions that the system needs to perform (e.g., Locate Item, Check Stock Levels, Send Restocking Alert).
3. Identify Data Stores: Define where data will be stored within the system (e.g., Item Database, Location Database).
4. Map Data Flows:
   - Connect external entities to processes by mapping data inputs and outputs.
   - Connect processes to data stores where necessary data is stored or retrieved.
   - Show data flow among different processes within the system.
5. Create DFD Levels:
   - Level 0 (Context Diagram): Illustrates the system as a single process and the external entities that interact with it.
   - Level 1: Breaks down the main process into sub-processes and shows data flow among them.
   - Level 2 (if needed): Further decomposes specific sub-processes for detailed analysis.

Input and Outputs :

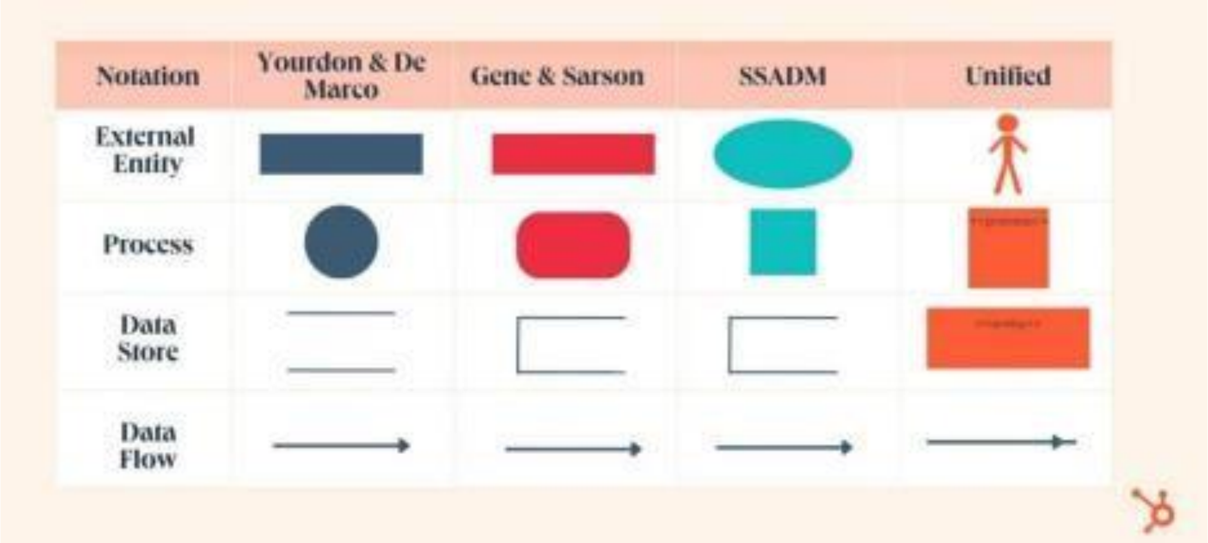
Inputs

1. Item Details: Item ID, Item Name, Stock Level, Restock Threshold
2. Location Details: Location ID, Location Name, Floor Number
3. Manager Details: Manager ID, Name, Contact Information
4. Customer Requests: Item ID (or name), Location Request

Outputs

1. Item Location: Location information provided to the customer.
2. Restocking Alert: Notification sent to the manager for restocking low stock items.
3. Reports: Summary of stock levels, restocking needs, and item location details.

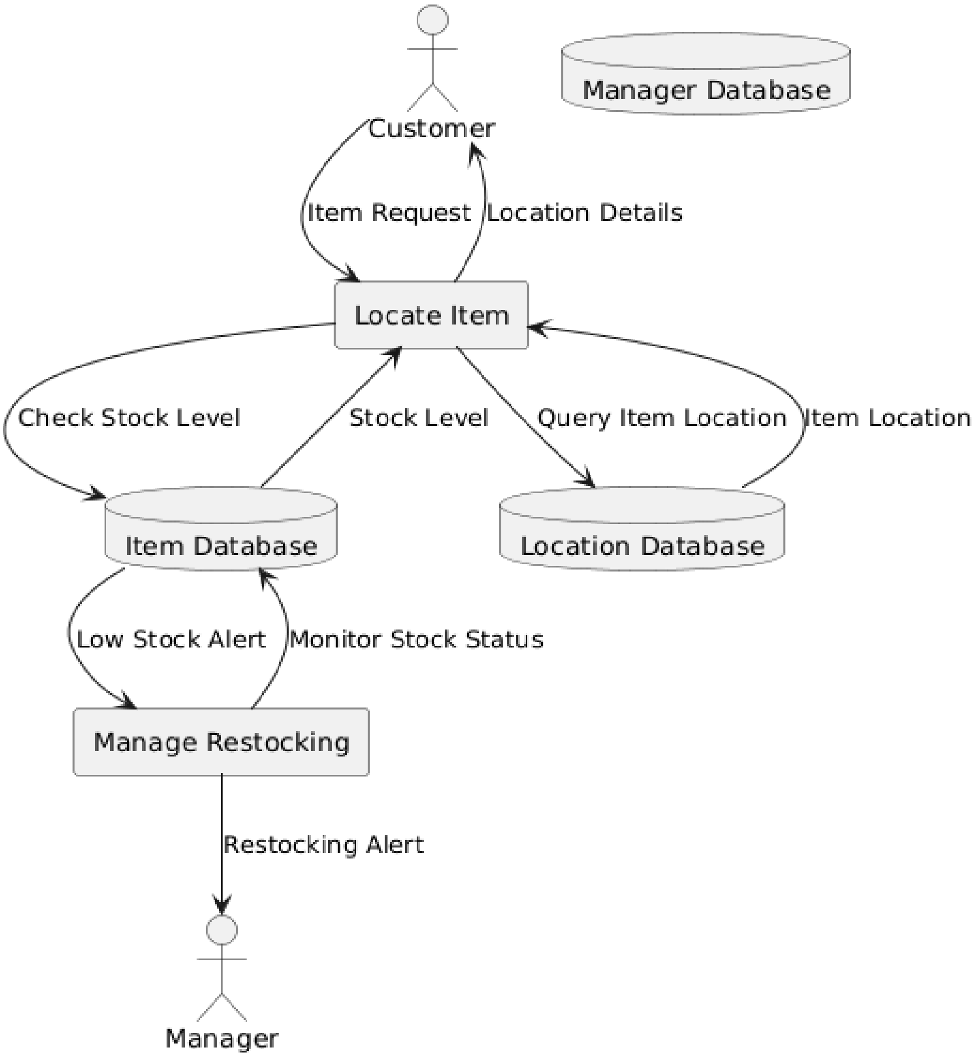
Sample diagrams:



Notation	Yourdon & De Marco	Gene & Sarson	SSADM	Unified
External Entity	▬	▬	⬬	🯅
Process	●	▢	■	■
Data Store	=	⊏	⊏	▬
Data Flow	→	→	→	→

Diagram:

Mall Directory System - Data Flow Diagram (DFD)



Result:

The Data Flow diagram was made successfully by following the steps described above.

5. USE CASE DIAGRAM

Aim :

To Draw the Use Case Diagram for mall directory system.

Algorithm :

The algorithm to create the use case diagram for the **Mall Directory System** is as follows:

1. **Identify Actors**: Determine the users interacting with the system, such as **Customer** and **Manager**.
2. **Identify Use Cases**: Identify the main functionalities of the system, such as:
   - Locating items within the mall.
   - Viewing details of specific items.
   - Checking stock levels for item availability.
   - Sending restocking alerts to managers.
   - Updating inventory after restocking.
3. **Establish Relationships**:
   - Define how each actor interacts with specific use cases.
   - Use "include" or "extend" relationships where necessary to show dependencies between use cases.
4. **Construct Use Case Diagram**:
   - Create a visual representation of actors and use cases.
   - Draw lines connecting actors to their relevant use cases.
5. **Review and Validate**: Ensure that all required interactions and functionalities are captured accurately.

Input:

Actors:

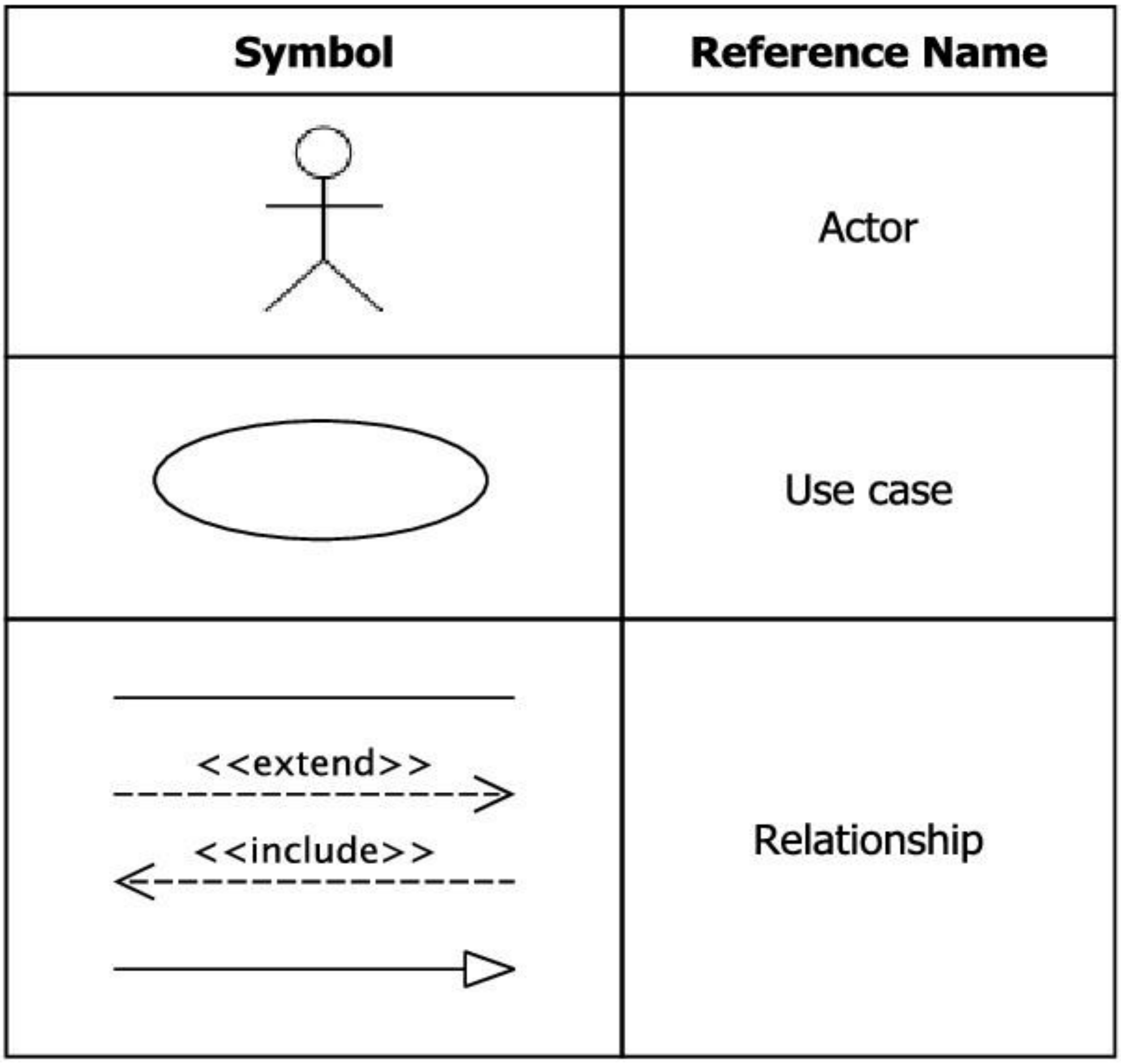
1. **Customer**: Searches for items, views item details, and checks item availability.
2. **Manager**: Receives alerts for items needing restocking and updates inventory.

Use Cases:

1. **Search Item Location**: Allows customers to locate items in the mall.
2. **View Item Details**: Allows customers to see details like price and availability.
3. **Check Stock Levels**: Part of the item search process to verify availability.
4. **Receive Restocking Alert**: Notifies the manager to restock low-stock items.

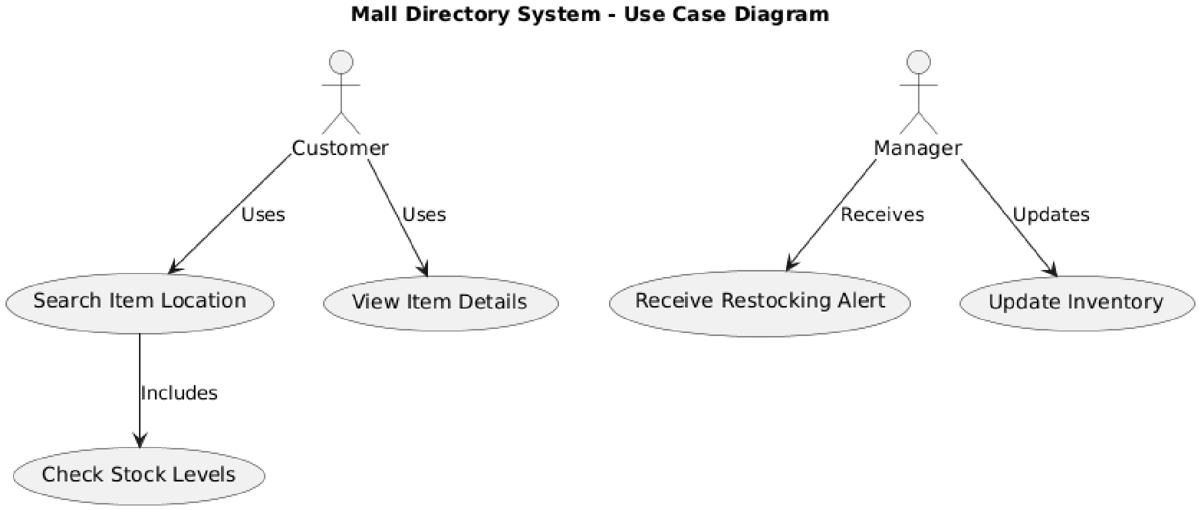
5. **Update Inventory**: Allows the manager to update stock after restocking.

Sample Symbols:



Symbol	Reference Name
	Actor
	Use case
<<extend>> <<include>>	Relationship

Diagram:



Result :

The use case diagram has been created successfully by following the steps given.

6. ACTIVITY DIAGRAM

Aim:

To Draw the activity Diagram for MALL DIRECTORY SYSTEM.

Algorithm :

The algorithm to create the activity diagram for the **Mall Directory System** is as follows:

1. **Define Activities**:
   - Identify the key activities involved for both **Customer** and **Manager**.
   - For the **Customer**: Searching for an item, viewing item details, and checking stock levels.
   - For the **Manager**: Receiving restocking alerts and updating inventory.
2. **Define Decision Points**:
   - Add decision points where there are choices, such as checking if an item is in stock and whether restocking is required.
3. **Map Activity Flow**:
   - Create the flow from the start to the end of each activity, showing the order in which tasks are performed.
4. **Add Synchronization Bars (if needed)**:
   - Show concurrent activities if necessary.
5. **Draw Activity Diagram**:
   - Use an activity diagram tool or PlantUML code to visually represent the sequence of actions and decision points.
6. **Review and Validate**:
   - Ensure the workflow accurately represents the intended operations.

Inputs :

1. **Customer Activities**:
   - **Search Item**: Search for items in the mall directory.
   - **View Item Details**: View details of specific items.
   - **Check Stock Levels**: Check the availability of items.
2. **Manager Activities**:
   - **Receive Restocking Alert**: Receive an alert when an item needs to be restocked.

- **Update Inventory**: Update stock levels after restocking.

Sample Symbols:

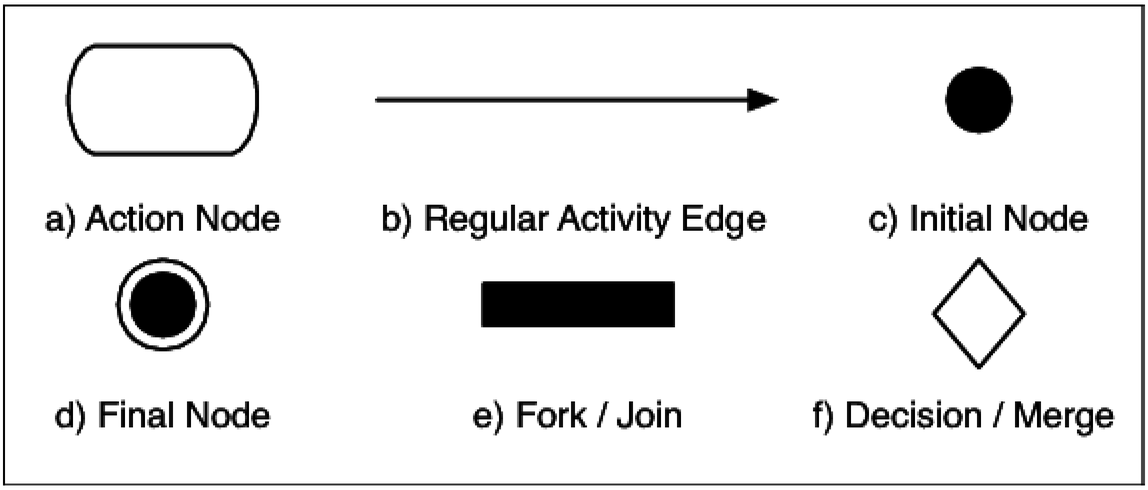
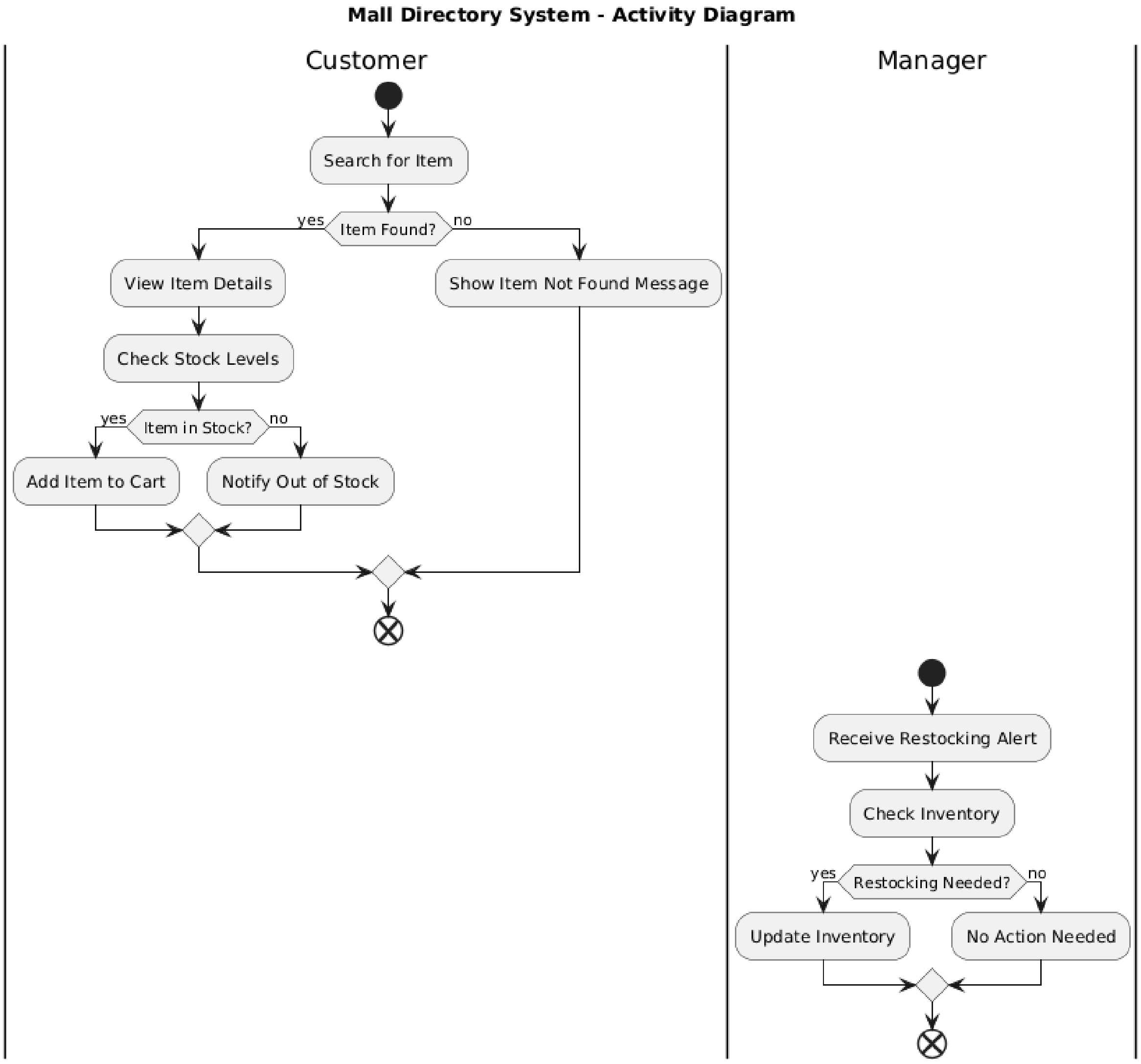


Diagram:



Result :

The Activity diagram has been created successfully by following the steps given.

7. STATE CHART DIAGRAM

Aim :

To Draw the State Chart Diagram for LIBRARY MANAGEMENT SYSTEM.

Algorithm :

To create the state chart diagram for the **Mall Directory System**, follow these steps:

1. **Identify Key States**:
   - Determine the states an item or inventory entry can be in. For example, states might include **In Stock**, **Out of Stock**, **Reserved**, and **Restocking**.
2. **Identify Events and Triggers**:
   - Identify events that cause transitions between states. For example, events could include **Customer Searches Item**, **Item Added to Cart**, and **Inventory Updated**.
3. **Map State Transitions**:
   - Define the transitions from one state to another, showing the conditions or actions that trigger each transition.
4. **Draw State Chart Diagram**:
   - Use a diagramming tool or PlantUML code to represent states as boxes and transitions as arrows between these boxes.
5. **Review and Validate**:
   - Verify that all states and transitions accurately reflect the behavior of the system.

Inputs :

1. **States**:
   - **In Stock**: Item is available in sufficient quantity.
   - **Out of Stock**: Item is not available in the inventory.
   - **Reserved**: Item has been reserved by a customer.
   - **Restocking**: Item is being restocked by the manager.
2. **Transitions/Events**:
   - **Customer Searches Item**: Triggers a check on the item’s availability.
   - **Item Added to Cart**: Changes item state to **Reserved** if in stock.
   - **Out of Stock Alert**: Changes item state to **Out of Stock** if stock level is zero.
   - **Restock Item**: Changes item state to **Restocking** and then back to **In Stock** once complete.

Sample Symbol:

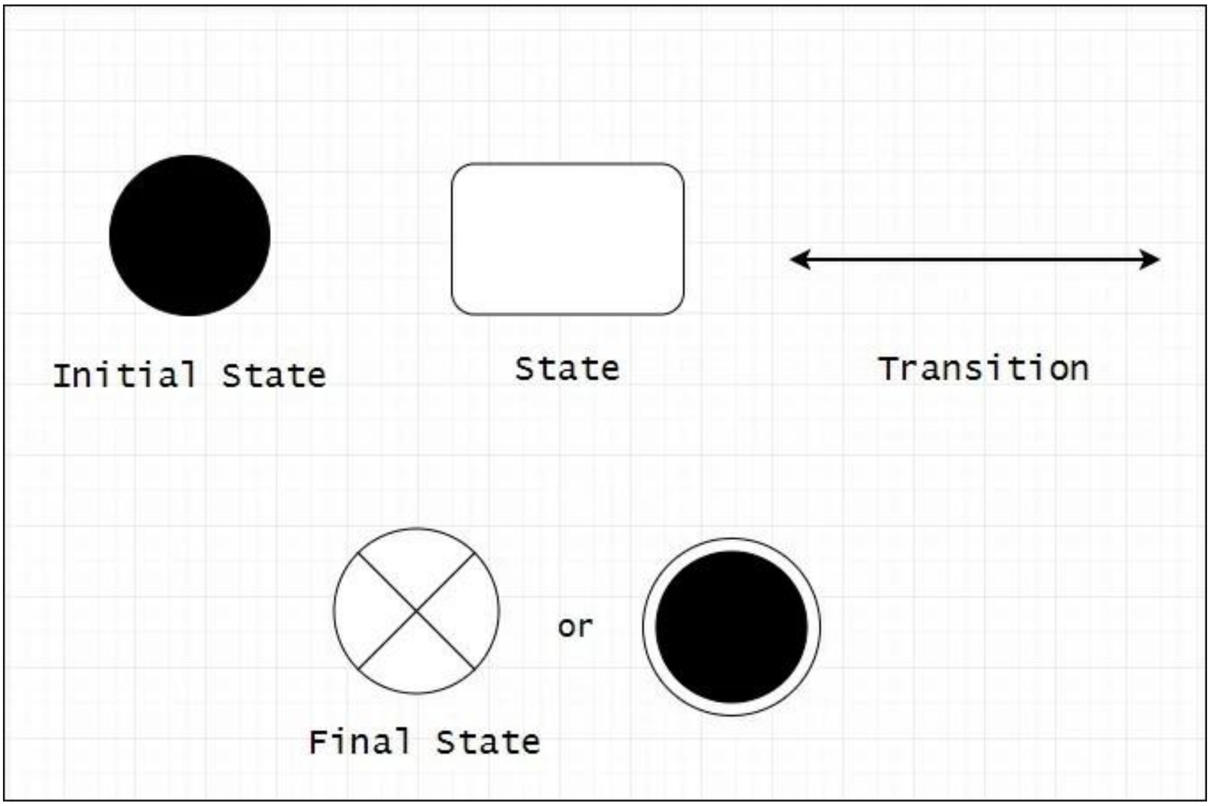
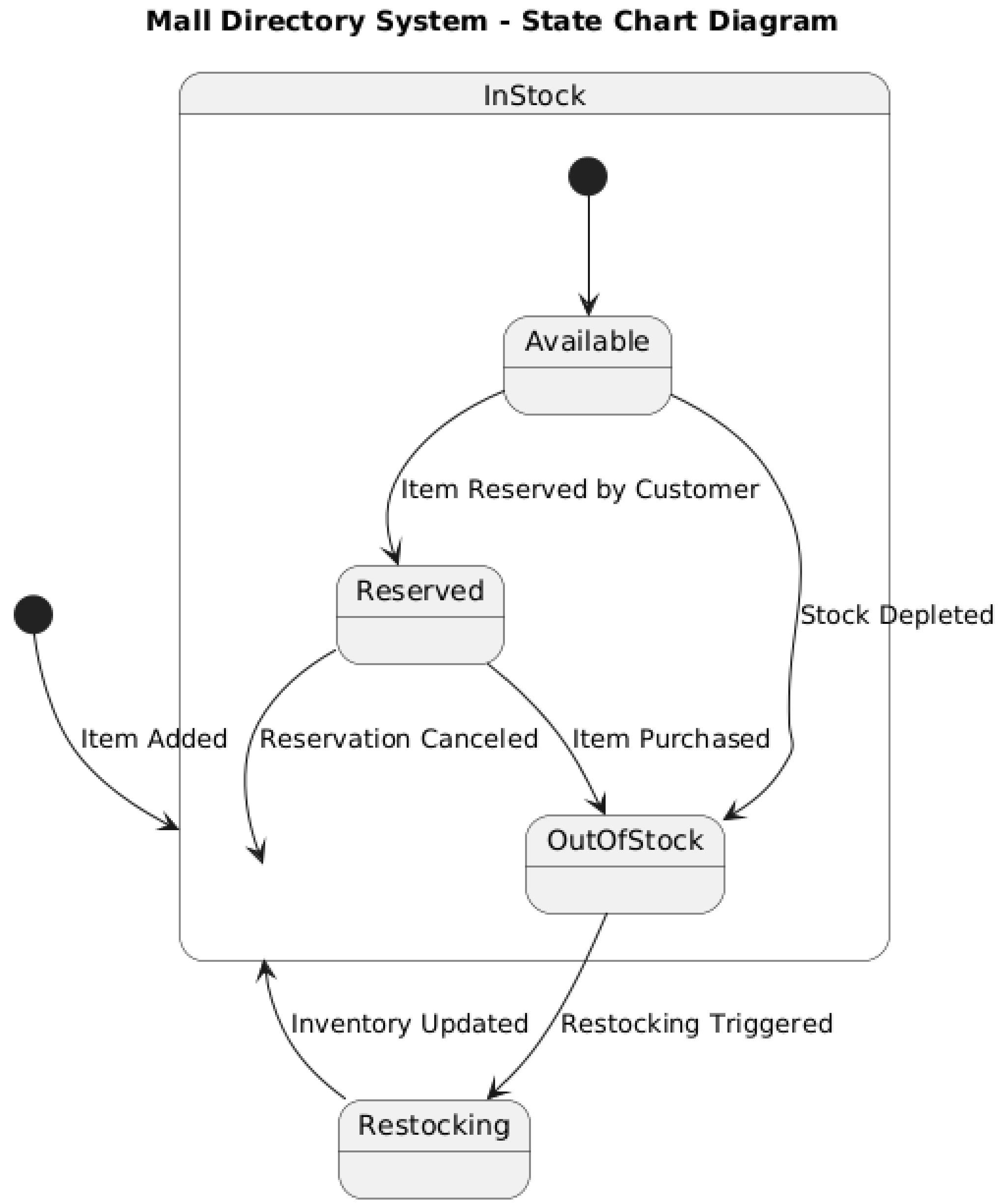


Diagram:



Result :

The State Chart diagram has been created successfully by following the steps given.

8. SEQUENCE DIAGRAM

Aim :

To Draw the Sequence Diagram for MALL DIRECTORY SYSTEM.

Algorithm :

To create the sequence diagram for the **Mall Directory System**, follow these steps:

1. **Identify Actors and Objects**:
   - Determine the main actors (e.g., **Customer** and **Manager**) and system components or objects (e.g., **Mall Directory System**, **Inventory**, and **Notification System**).
2. **Define the Sequence of Interactions**:
   - Outline the messages exchanged between actors and system components. For example, **Customer** might send a request to locate an item, and **Mall Directory System** might check with the **Inventory** to confirm availability.
3. **Determine Control Flow**:
   - Define the order in which messages are sent, such as **Customer** initiating a search, **Mall Directory System** responding with item location, and **Inventory** notifying **Manager** if restocking is needed.
4. **Draw Sequence Diagram**:
   - Use a diagramming tool or PlantUML code to represent actors and objects as lifelines, and messages as arrows with labels showing the interaction.
5. **Validate the Diagram**:
   - Verify that all interactions accurately reflect the behavior of the system.

Inputs :

1. **Actors**:
   - **Customer**: User searching for an item and adding it to the cart.
   - **Manager**: Responsible for handling restocking alerts.
2. **Objects**:
   - **Mall Directory System**: Central system that helps locate items.
   - **Inventory**: Keeps track of item stock levels.
   - **Notification System**: Notifies the manager about items needing restocking.
3. **Interactions**:
   - **Search Item**: Customer requests the location of an item.
   - **Check Availability**: Mall Directory System checks the item's stock level in the Inventory.

- **Add to Cart**: Customer reserves the item by adding it to the cart.
- **Restocking Alert**: Notification System sends an alert to the Manager if stock is low.

Sample Diagram:

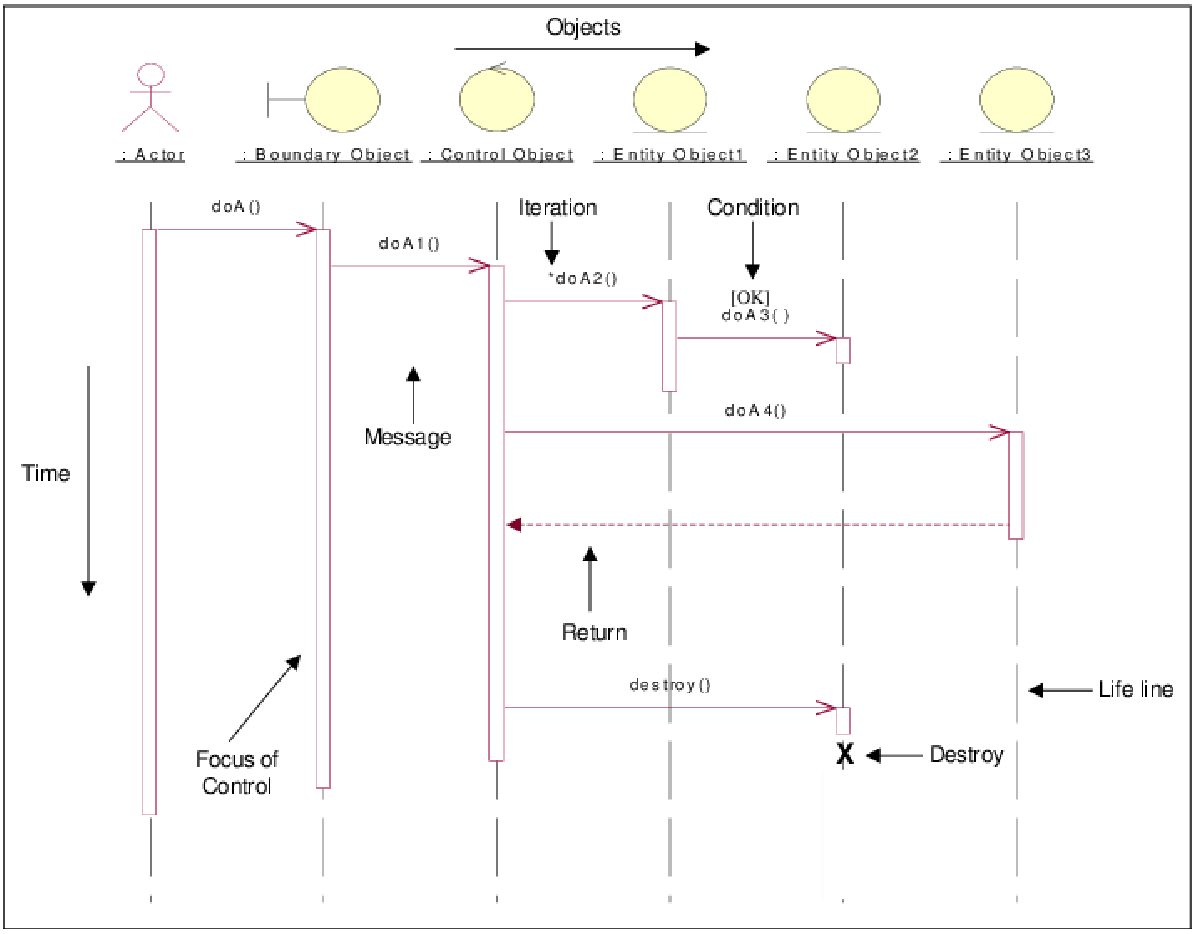
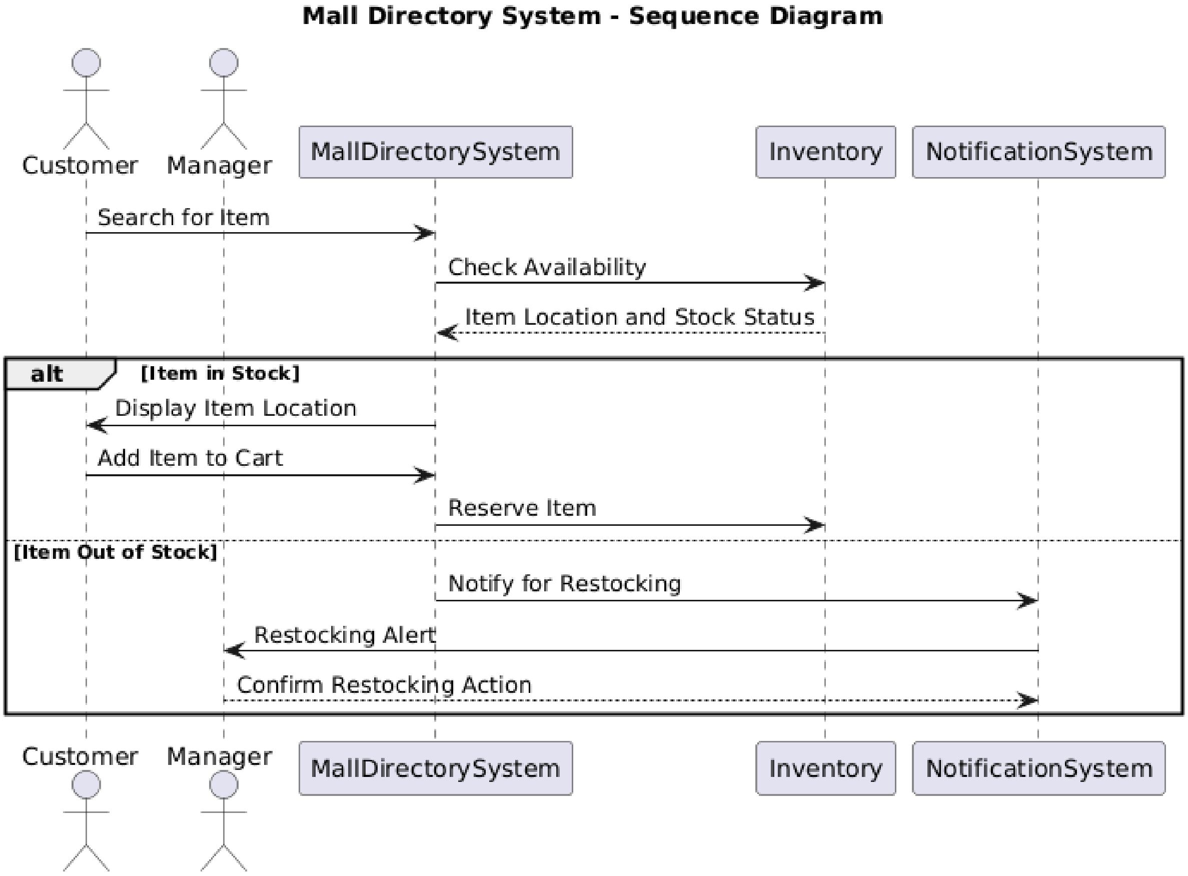


Diagram:



Result :

The Sequence diagram has been created successfully by following the steps given.

9. COLLABORATION DIAGRAM

Aim :

To Draw the Collaboration Diagram **for MALL DIRECTORY SYSTEM.**

Algorithm :

To create the **Collaboration Diagram** for the **Mall Directory System**, follow these steps:

1. **Identify Actors and Objects**:
   - Determine the main actors (e.g., **Customer** and **Manager**) and key objects (e.g., **Mall Directory System**, **Inventory**, **Notification System**) involved in the system.
2. **Define Relationships and Links**:
   - Establish the relationships between objects, such as **Customer** interacting with **Mall Directory System** to search for items and **Mall Directory System** interacting with **Inventory** to check stock levels.
3. **Define Interactions and Numbering**:
   - Label each interaction with numbers to indicate the sequence of messages. For example, **Customer** requests the item location, and **Mall Directory System** checks the stock in **Inventory**.
4. **Draw the Collaboration Diagram**:
   - Use a diagramming tool or PlantUML code to represent actors and objects as nodes, and interactions as lines connecting them with numbered messages.
5. **Validate the Diagram**:
   - Confirm that all relationships and messages accurately represent the behavior of the system.

Inputs :

1. **Actors**:
   - **Customer**: The user searching for an item.
   - **Manager**: The individual who receives restocking alerts.
2. **Objects**:
   - **Mall Directory System**: Central system assisting the customer.
   - **Inventory**: System managing item stock levels.
   - **Notification System**: System that alerts the manager for restocking.
3. **Interactions**:
   - **Locate Item**: Customer asks the **Mall Directory System** to locate an item.

- **Check Availability**: **Mall Directory System** checks item availability in **Inventory**.
- **Add to Cart**: Customer reserves the item.
- **Restocking Alert**: Notification sent to the **Manager** if the item is low in stock.

Sample Symbol:

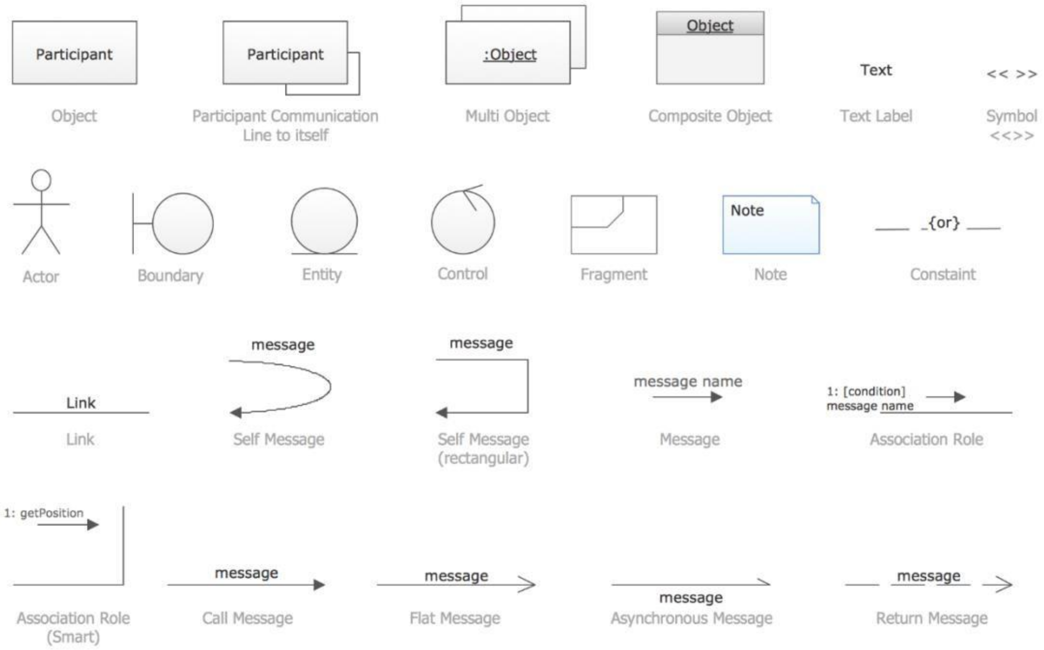
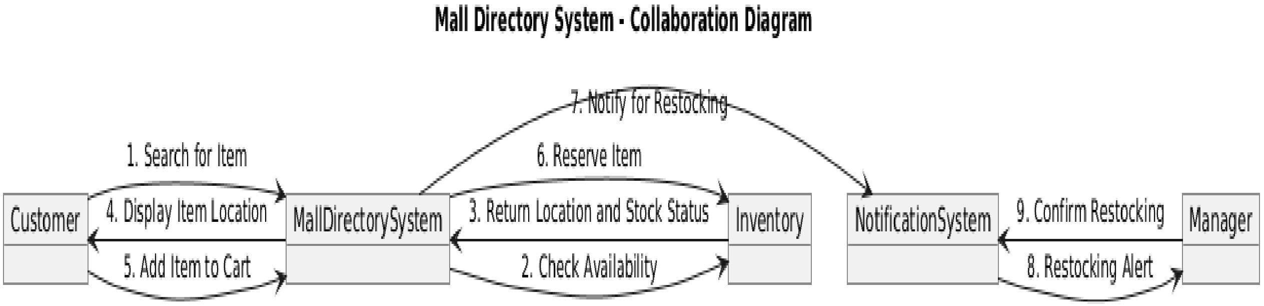


Diagram:



Result :

The Collaboration diagram has been created successfully by following the steps given.

10. CLASS DIAGRAM

Aim:

To Draw the Class Diagram for MALL DIRECTORY SYSTEM.

Algorithm :

1. **Identify Classes**: Determine the main classes involved in the system, such as MallDirectorySystem, Customer, Inventory, NotificationSystem, and Manager.
2. **Define Attributes and Methods**: For each class, specify the relevant attributes (data) and methods (operations) needed to perform the class functions.
3. **Establish Relationships**: Identify how classes are related (e.g., association, aggregation, composition).
4. **Set Visibility**: Assign visibility (public, private, protected) to attributes and methods to encapsulate data appropriately.
5. **Draw Class Diagram**: Use a diagramming tool or code (like PlantUML) to visually represent the classes, attributes, methods, and relationships.
6. **Review and Refine**: Validate the diagram with project requirements and refine it as needed.

Inputs :

- **Customer Information**: customerId, name, email
- **Inventory Data**: itemId, name, location, stock
- **Notification Data**: notificationId, message, date
- **Manager Information**: managerId, name
- **Mall Directory System Information**: This class will integrate various services, such as searching for items and handling restocking alerts.

Sample Symbol :

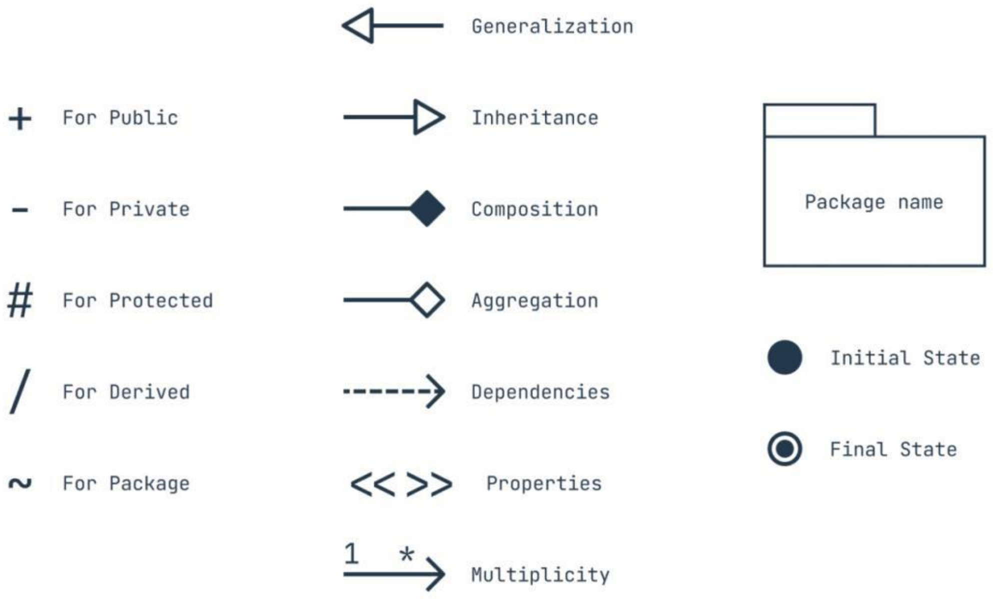
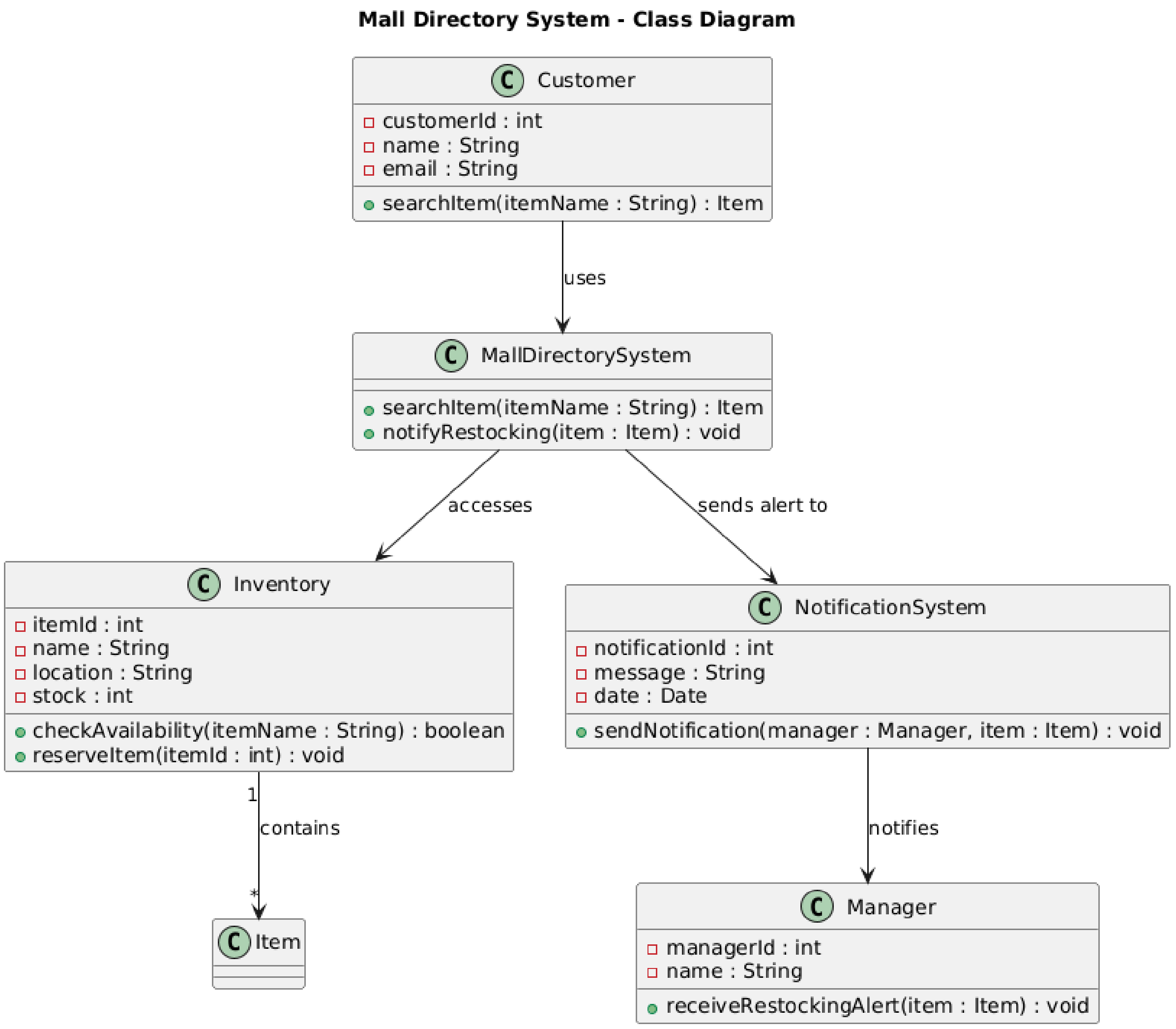


Diagram:


Mall Directory System - Class Diagram
C Customer
customerId : int
name : String
email : String
searchItem(itemName : String) : Item
uses
C MallDirectorySystem
searchItem(itemName : String) : Item
notifyRestocking(item : Item) : void
accesses
sends alert to
C Inventory
itemId : int
name : String
location : String
stock : int
checkAvailability(itemName : String) : boolean
reserveItem(itemId : int) : void
C NotificationSystem
notificationId : int
message : String
date : Date
sendNotification(manager : Manager, item : Item) : void
1
contains
*
notifies
C Item
C Manager
managerId : int
name : String
receiveRestockingAlert(item : Item) : void

Result:

The Class diagram has been created successfully by following the steps given.

CODE FOR MINI PROJECT

CODE:

IMPLEMENTATION CODE USING PYTHON

```
# Mall Directory and Goods Restocking System

class Item:
    def __init__(self, item_id, name, stock_level, restock_threshold, price, location, category):
        self.item_id = item_id
        self.name = name
        self.stock_level = stock_level
        self.restock_threshold = restock_threshold
        self.price = price
        self.location = location
        self.category = category

    def check_restock(self):
        return self.stock_level < self.restock_threshold


class Location:
    def __init__(self, location_id, name, floor):
        self.location_id = location_id
        self.name = name
        self.floor = floor


class Category:
    def __init__(self, category_id, name):
        self.category_id = category_id
        self.name = name


class RestockingAlert:
    def __init__(self, item, alert_date, status="Pending"):
        self.item = item
        self.alert_date = alert_date
        self.status = status

    def mark_as_resolved(self):
        self.status = "Resolved"


class Manager:
    def __init__(self, manager_id, name, email, phone):
        self.manager_id = manager_id
        self.name = name
        self.email = email
        self.phone = phone
    def receive_alert(self, alert):
        print(f"ALERT: Item '{alert.item.name}' needs restocking! Stock: {alert.item.stock_level}")
        print(f"Location: {alert.item.location.name} (Floor {alert.item.location.floor})")
        print(f"Category: {alert.item.category.name}")


class MallDirectorySystem:
    def __init__(self):
        self.items = []
        self.locations = []
        self.categories = []
        self.alerts = []

    def add_location(self, location_id, name, floor):
        location = Location(location_id, name, floor)
        self.locations.append(location)
        return location

    def add_category(self, category_id, name):
        category = Category(category_id, name)
        self.categories.append(category)
        return category

    def add_item(self, item_id, name, stock_level, restock_threshold, price, location, category):
        item = Item(item_id, name, stock_level, restock_threshold, price, location, category)
        self.items.append(item)
        return item

    def check_stock(self, manager):
        for item in self.items:
            if item.check_restock():
                alert = RestockingAlert(item, alert_date="2024-11-21")
                self.alerts.append(alert)
                manager.receive_alert(alert)

    def locate_item(self, item_name):
        for item in self.items:
            if item.name.lower() == item_name.lower():
                return f"Item '{item.name}' is located at {item.location.name} (Floor {item.location.floor})."
        return f"Item '{item_name}' not found in the mall."

    def display_items(self):
        print("\n--- Mall Items ---")
        for item in self.items:
            print(
                f"ID: {item.item_id}, Name: {item.name}, Stock: {item.stock_level}, "
                f"Price: {item.price}, Location: {item.location.name}, Category: {item.category.name}"
            )
        print("------------------")


# Example Usage
```

```
if __name__ == "__main__":
    # Create the mall directory system
    mall_system = MallDirectorySystem()

    # Add locations
    loc1 = mall_system.add_location(1, "Electronics Section", 1)
    loc2 = mall_system.add_location(2, "Clothing Section", 2)

    # Add categories
    cat1 = mall_system.add_category(1, "Electronics")
    cat2 = mall_system.add_category(2, "Clothing")

    # Add items
    mall_system.add_item(1, "Laptop", 5, 10, 75000, loc1, cat1)
    mall_system.add_item(2, "Jeans", 3, 5, 1500, loc2, cat2)

    # Create a manager
    manager = Manager(1, "John Doe", "john@example.com", "1234567890")

    # Display items
    mall_system.display_items()

    # Locate an item
    print(mall_system.locate_item("Laptop"))

    # Check stock and alert manager
    mall_system.check_stock(manager)
```

Output:

```
--- Mall Items ---
ID: 1, Name: Laptop, Stock: 5, Price: 75000, Location: Electronics Section, Category: Electronics
ID: 2, Name: Jeans, Stock: 3, Price: 1500, Location: Clothing Section, Category: Clothing
------------------

Item 'Laptop' is located at Electronics Section (Floor 1).

ALERT: Item 'Laptop' needs restocking! Stock: 5
Location: Electronics Section (Floor 1)
Category: Electronics

ALERT: Item 'Jeans' needs restocking! Stock: 3
Location: Clothing Section (Floor 2)
Category:Clothing
```