Delivery Management System

Maryam Alraeesi - 202304233

ZAYED UNIVERSITY

ICS220 > 22111 Program. Fund.

Prof Sujith Mathew

Feb 28, 2025

1) Identify Use-Cases

a) The main use cases are:

Create Delivery Order

Manage Recipient Details

Process Payment

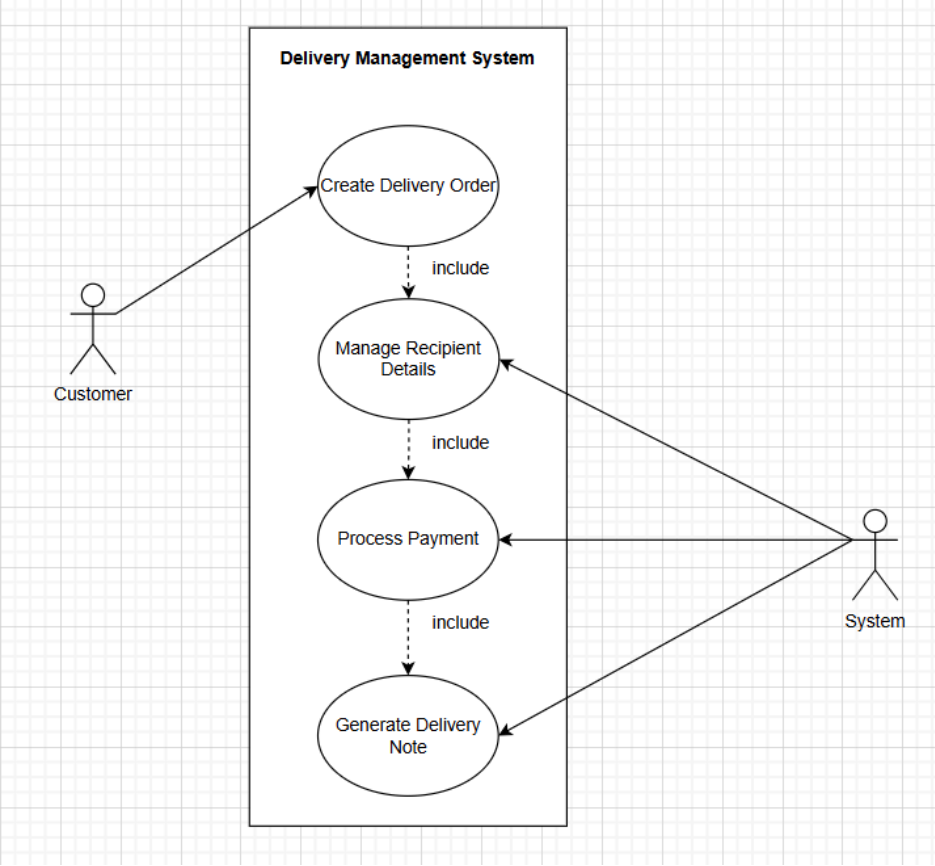
Generate Delivery Note

Track Delivery

Update Delivery Status

b) UML use-case diagram 3 scenarios

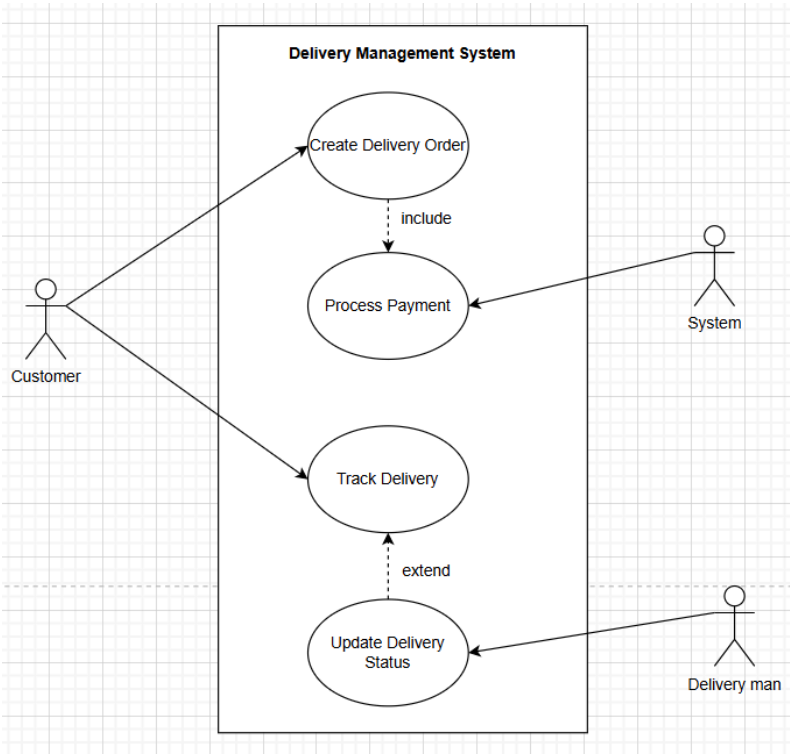
Scenario 1



Scenario 1: Basic Order Flow
This scenario represents the fundamental process of creating and completing a delivery order:

1) The Customer initiates the process by creating a delivery order.
2) As part of the order creation, the system includes managing recipient details.
3) Once recipient details are confirmed, the system proceeds to process the payment.
4) After successful payment, the system generates a delivery note.

Scenario 2



Scenario 2: Extended Delivery Tracking

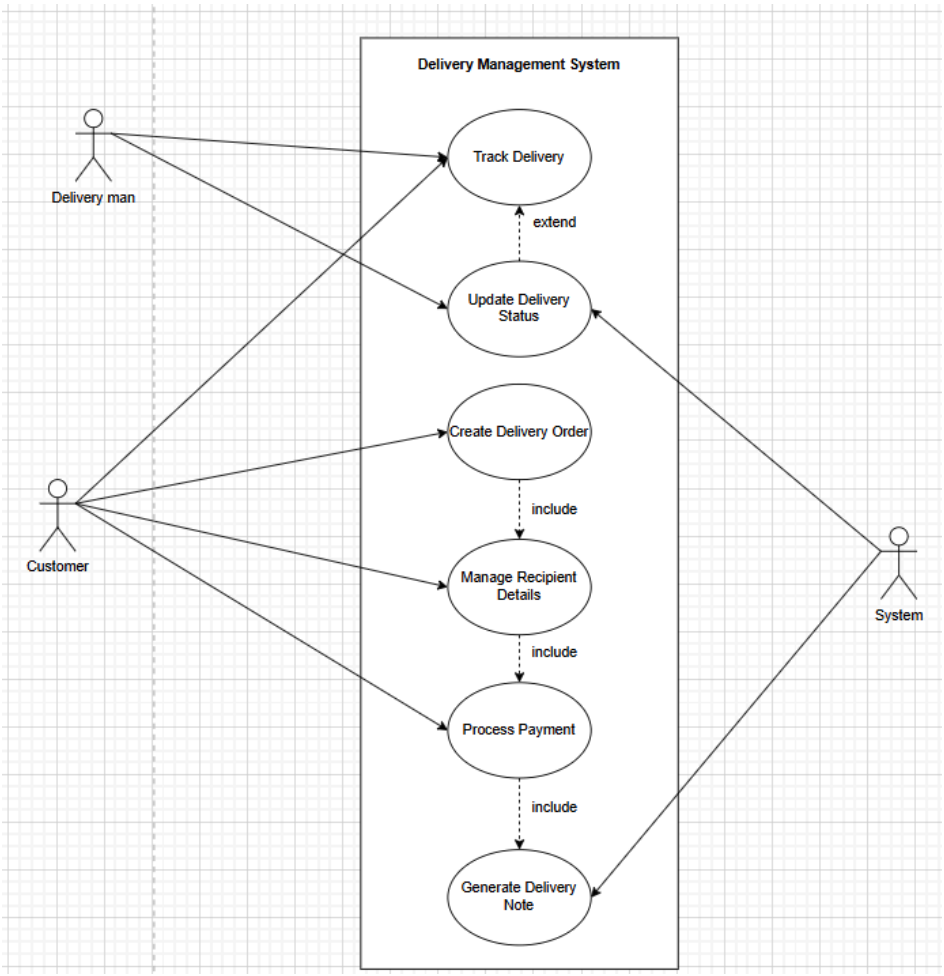
This scenario focuses on order creation and tracking:
1) The Customer creates a delivery order, which includes processing the payment.

2) The Customer can track the delivery status of their order.

3) The Delivery Man updates the delivery status.

4) The tracking functionality is extended by the status updates provided by the Delivery Man.

Scenario 3



Scenario 3: Comprehensive System Overview

This scenario provides a complete view of the system, including all actors and use cases:

1) The Customer can create delivery orders, manage recipient details, process payments, and track deliveries.

2) The Delivery Man is involved in tracking deliveries and updating delivery statuses.

3) The Admin generates delivery notes and can also update delivery statuses.

4) The system shows the relationships between different use cases, such as order creation including recipient management and payment processing, which then leads to generating a delivery note.

5) The tracking functionality is extended by status updates, accessible to both customers and delivery personnel.

c) Use-Case Description Tables

1) Create Delivery Order

Use-Case	**Create Delivery Order**
Actors	Customer
Description	Customer starts a new delivery order
Preconditions	Customer is logged into the system
Main Flow	Customer selects "Create New Order" System displays available items for delivery Customer selects items and specifies quantities System calculates subtotal, taxes, and total charges Customer reviews order summary Customer confirms order System generates unique order number and reference number System prompts for recipient details
Alternate Flow	If an item is out of stock, system notifies customer and suggests alternatives If customer wants to modify order, they can return to item selection
Postcondition	New delivery order is created and saved in the system

2) Manage Recipient Details

Use-Case	**Manage Recipient Details**
Actors	Customer
Description	Customer enters or updates recipient

	information
Preconditions	Delivery order initiated
Main Flow	System prompts for recipient details Customer enters recipient's name, contact information, and delivery address System validates entered information Customer confirms recipient details System saves recipient information
Alternate Flow	If information is invalid, system highlights errors and prompts for correction Customer can choose to use previously saved recipient details
Postcondition	Recipient details are associated with the delivery order

3) Process Payment

Use-Case	**Process Payment**
Actors	Customer
Description	Customer completes payment for the order
Preconditions	Delivery order created and recipient details entered
Main Flow	System displays order summary and total charges Customer selects payment method (e.g., credit card, digital wallet) Customer enters payment details System securely processes payment through payment gateway System receives payment confirmation

	System updates order status to "Paid" System generates payment receipt
Alternate Flow	If payment is declined, system notifies customer and offers retry or alternative payment method Customer can cancel payment process and return to order modification
Postcondition	Payment is processed and order is confirmed

4) Generate Delivery Note

Use-Case	**Generate Delivery Note**
Actors	System
Description	System creates a detailed delivery note for the order
Preconditions	Order is created and payment is processed
Main Flow	System retrieves order details, recipient information, and payment data System generates a unique delivery note number System compiles delivery information (recipient details, order number, items, prices, etc.) System calculates package dimensions and weight System formats the delivery note with company branding System makes the delivery note available for printing or digital access
Alternate Flow	If any required information is missing, system flags the issue for manual review System can regenerate delivery note if

	order details are updated
Postcondition	Delivery note is generated and associated with the order

5) Track Delivery

Use-Case	**Track Delivery**
Actors	Customer, Delivery man
Description	Allows tracking of the delivery status and location
Preconditions	Delivery order is confirmed and assigned to a delivery man
Main Flow	Actor enters order number or scans QR code System retrieves current status and location of the delivery System displays estimated delivery time and route information Actor can opt to receive real-time updates
Alternate Flow	If delivery status hasn't been updated, system shows last known information Customer can contact support if tracking information is unclear
Postcondition	Actor is informed of the current delivery status and location

6) Update Delivery Status

Use-Case	**Update Delivery Status**
Actors	Delivery man, System
Description	Updates the status of the delivery at various stages
Preconditions	Delivery is in progress

Main Flow	Delivery man selects the order to update System presents status options (e.g., "In Transit", "Out for Delivery", "Delivered") Delivery man selects appropriate status Delivery man adds any necessary notes or photos System updates the delivery status in real-time System notifies customer of status change
Alternate Flow	If delivery fails, delivery man selects "Failed Delivery" and provides reason System can automatically update status based on GPS data or scheduled events
Postcondition	Delivery status is updated in the system and customer is notified

2) Identify Objects and Classes

a) Identifying Objects and Their Respective Classes

Order

- Represents a delivery order in the system
- Handles order creation and payment processing

Customer

- Represents a customer using the delivery service
- Manages recipient details

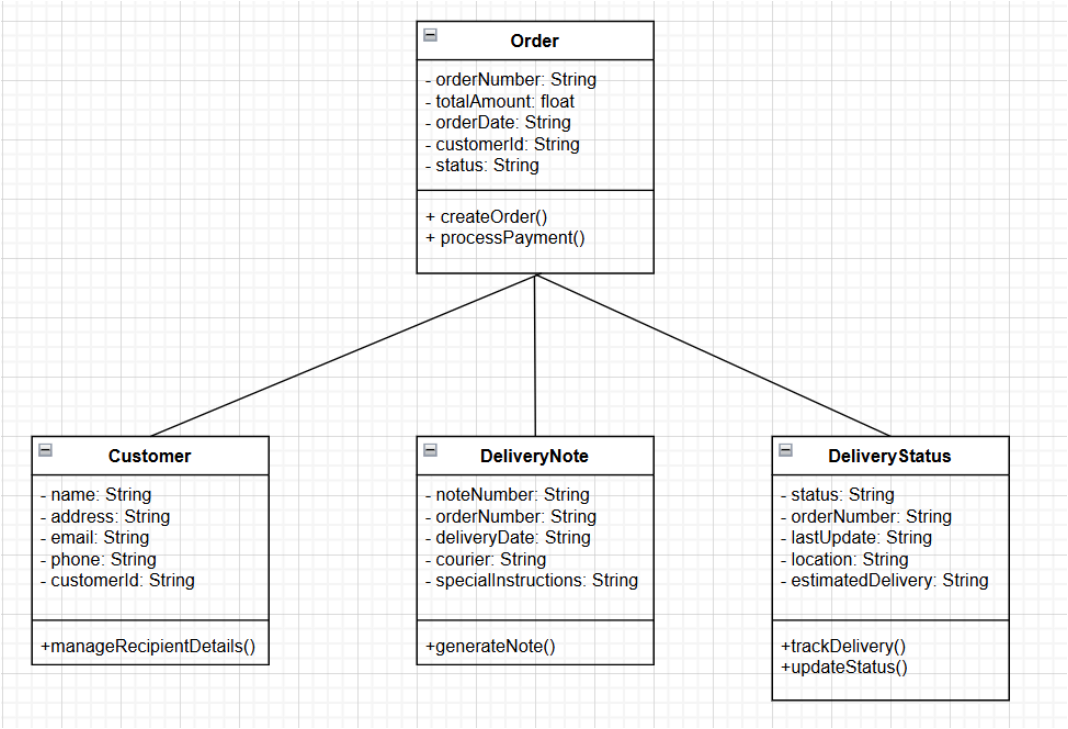
DeliveryNote

- Represents the delivery note for an order
- Handles generation of delivery documentation

DeliveryStatus

- Represents the status of a delivery
- Manages tracking and updating of delivery status

b) Draw UML Class Diagram



c) Supporting Descriptions & Access Specifiers

Class Descriptions:

- Order:
  - Represents a delivery order in the system.
  - Private attributes: orderNumber, totalAmount, orderDate, customerId, status.
  - Public methods: createOrder(), processPayment().
- Customer:

    - Represents a customer using the delivery service.
    - Private attributes: name, address, email, phone, customerId.
    - Public method: manageRecipientDetails().
- DeliveryNote:
    - Represents the delivery note for an order.
    - Private attributes: noteNumber, orderNumber, deliveryDate, courier, specialInstructions.
    - Public method: generateNote().
- DeliveryStatus:
    - Represents the current status of a delivery.
    - Private attributes: status, orderNumber, lastUpdate, location, estimatedDelivery.
    - Public methods: trackDelivery(), updateStatus()

3-4) Create Python Classes and Objects and use objects to generate a Delivery Note:

The code:

```
class Order:
    """Represents a delivery order in the system."""

    def __init__(self, order_number, total_amount, order_date,
customer_id, status): #Initialize order attributes
        self.__order_number = order_number
        self.__total_amount = total_amount
        self.__order_date = order_date
        self.__customer_id = customer_id
        self.__status = status

    # Getter and setter methods for order attributes
    def get_order_number(self):
        return self.__order_number

    def set_order_number(self, order_number):
        self.__order_number = order_number

    def get_total_amount(self):
        return self.__total_amount

    def set_total_amount(self, total_amount):
        self.__total_amount = total_amount

    def get_order_date(self):
        return self.__order_date

    def set_order_date(self, order_date):
```

```
        self.__order_date = order_date

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def get_status(self):
        return self.__status

    def set_status(self, status):
        self.__status = status

    def create_order(self):
        pass #Simulate order creation logic

    def process_payment(self):
        pass #Simulate payment processing logic


class Customer:
    """Represents a customer using the delivery service."""

    def __init__(self, name, address, email, phone, customer_id):
#Initialize customer attributes
        self.__name = name
        self.__address = address
        self.__email = email
        self.__phone = phone
        self.__customer_id = customer_id

    # Getter and setter methods for customer attributes
    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    def get_address(self):
        return self.__address

    def set_address(self, address):
        self.__address = address

    def get_email(self):
        return self.__email

    def set_email(self, email):
```

```
        self.__email = email

    def get_phone(self):
        return self.__phone

    def set_phone(self, phone):
        self.__phone = phone

    def get_customer_id(self):
        return self.__customer_id

    def set_customer_id(self, customer_id):
        self.__customer_id = customer_id

    def manage_recipient_details(self):
        pass #Simulate recipient details management


class DeliveryNote:
    """Represents the delivery note for an order."""

    def __init__(self, note_number, order_number, delivery_date, courier,
special_instructions):
        """Initialize delivery note attributes."""
        self.__note_number = note_number
        self.__order_number = order_number
        self.__delivery_date = delivery_date
        self.__courier = courier
        self.__special_instructions = special_instructions

    # Getter and setter methods for delivery note attributes
    def get_note_number(self):
        return self.__note_number

    def set_note_number(self, note_number):
        self.__note_number = note_number

    def get_order_number(self):
        return self.__order_number

    def set_order_number(self, order_number):
        self.__order_number = order_number

    def get_delivery_date(self):
        return self.__delivery_date

    def set_delivery_date(self, delivery_date):
        self.__delivery_date = delivery_date
```

```
    def get_courier(self):
        return self.__courier

    def set_courier(self, courier):
        self.__courier = courier

    def get_special_instructions(self):
        return self.__special_instructions

    def set_special_instructions(self, special_instructions):
        self.__special_instructions = special_instructions

    def generate_note(self):
        pass #Simulate delivery note generation


class DeliveryStatus:
    """Represents the current status of a delivery."""

    def __init__(self, status, order_number, last_update, location,
estimated_delivery): #Initialize delivery status attributes
        self.__status = status
        self.__order_number = order_number
        self.__last_update = last_update
        self.__location = location
        self.__estimated_delivery = estimated_delivery

    # Getter and setter methods for delivery status attributes
    def get_status(self):
        return self.__status

    def set_status(self, status):
        self.__status = status

    def get_order_number(self):
        return self.__order_number

    def set_order_number(self, order_number):
        self.__order_number = order_number

    def get_last_update(self):
        return self.__last_update

    def set_last_update(self, last_update):
        self.__last_update = last_update

    def get_location(self):
        return self.__location
```

```
    def set_location(self, location):
        self.__location = location

    def get_estimated_delivery(self):
        return self.__estimated_delivery

    def set_estimated_delivery(self, estimated_delivery):
        self.__estimated_delivery = estimated_delivery

    def track_delivery(self):
        pass #Simulate delivery tracking logic

    def update_status(self):
        pass #Simulate updating the delivery status


# Create objects of all identified classes
order = Order("DEL123456789", 283.50, "2025-01-25", "CUST001",
"Processing")
customer = Customer("Sarah Johnson", "45 Knowledge Avenue, Dubai, UAE",
"sarah.johnson@example.com", "+1234567890", "CUST001")
delivery_note = DeliveryNote("DN-2025-001", "DEL123456789", "2025-01-25",
"FastDelivery", "Leave at reception")
delivery_status = DeliveryStatus("In Transit", "DEL123456789",
"2025-01-24 14:30", "Dubai Sorting Center", "2025-01-25")

# Display Delivery Note information
print("Delivery Note")
print("============")
print("Recipient: " + customer.get_name())
print("Contact: " + customer.get_email())
print("Delivery Address: " + customer.get_address())
print("Order Number: " + order.get_order_number())
print("Reference Number: " + delivery_note.get_note_number())
print("Delivery Date: " + delivery_note.get_delivery_date())
print("Total Charges: AED " + str(order.get_total_amount()))
print("Delivery Status: " + delivery_status.get_status())

# Demonstrate method calls
order.create_order()
order.process_payment()
customer.manage_recipient_details()
delivery_note.generate_note()
delivery_status.track_delivery()
delivery_status.update_status()
```

The output:

Delivery Note
==========
Recipient: Sarah Johnson
Contact: sarah.johnson@example.com
Delivery Address: 45 Knowledge Avenue, Dubai, UAE
Order Number: DEL123456789
Reference Number: DN-2025-001
Delivery Date: 2025-01-25
Total Charges: AED 283.5
Delivery Status: In Transit

5) GitHub repository link

https://github.com/MaryamAlraeesi14/Assignment-1.git

6) Summary of learnings

In completing this assignment, I gained valuable insights into modeling real-world systems using object-oriented principles. I learned to break down complex processes like delivery management into manageable classes with clear attributes and methods. The assignment taught me how to represent relationships between classes using UML concepts like "include" and "extend", and how to implement these in Python code. By simulating a real-world process such as generating a delivery note, I better understood the importance of data flow and interaction between objects. I learned to structure code more effectively, using classes to represent distinct entities within a system.

Overall, this assignment significantly improved my object-oriented programming skills and deepened my understanding of system design principles, preparing me for more complex software development challenges in the future.