Assignment 1

ICS214 > 21545 Prob Stats & Randomness

Salma Almansoori – 202317014

28 Feb 2025

1. Identify Use-Cases:

a. Identify the use-cases for the delivery management system.

A delivery management system helps organize and track delivery orders efficiently, it allows customers to place orders, delivery staff to update statuses, and the system to generate important documents like delivery notes.

Actors (Users) in the System

- Customer : the person who places an order for a delivery, like they provide details such as the recipient's name, contact, and delivery address. Customers can also track their orders by entering a tracking number.
- Delivery Staff : the employee responsible for delivering packages to customers. They update the delivery status in the system, marking packages as "Out for Delivery" or "Delivered."
- Admin : is the person managing the delivery system. The admin assigns delivery staff to new orders and ensures the smooth processing of deliveries, they also have access to customer and order details.
- System : is the automated part of the software that processes orders, generates delivery notes, and keeps records of transactions.

Use-Cases for the Delivery Management System

1. Place a Delivery Order : the customer submits an order by entering recipient details, package weight, and delivery address. Then the system assigns a unique order number and confirms the request.

2. Track a Delivery : the customer checks the current status of their package by entering a tracking number, after that the system fetches and displays the latest information.

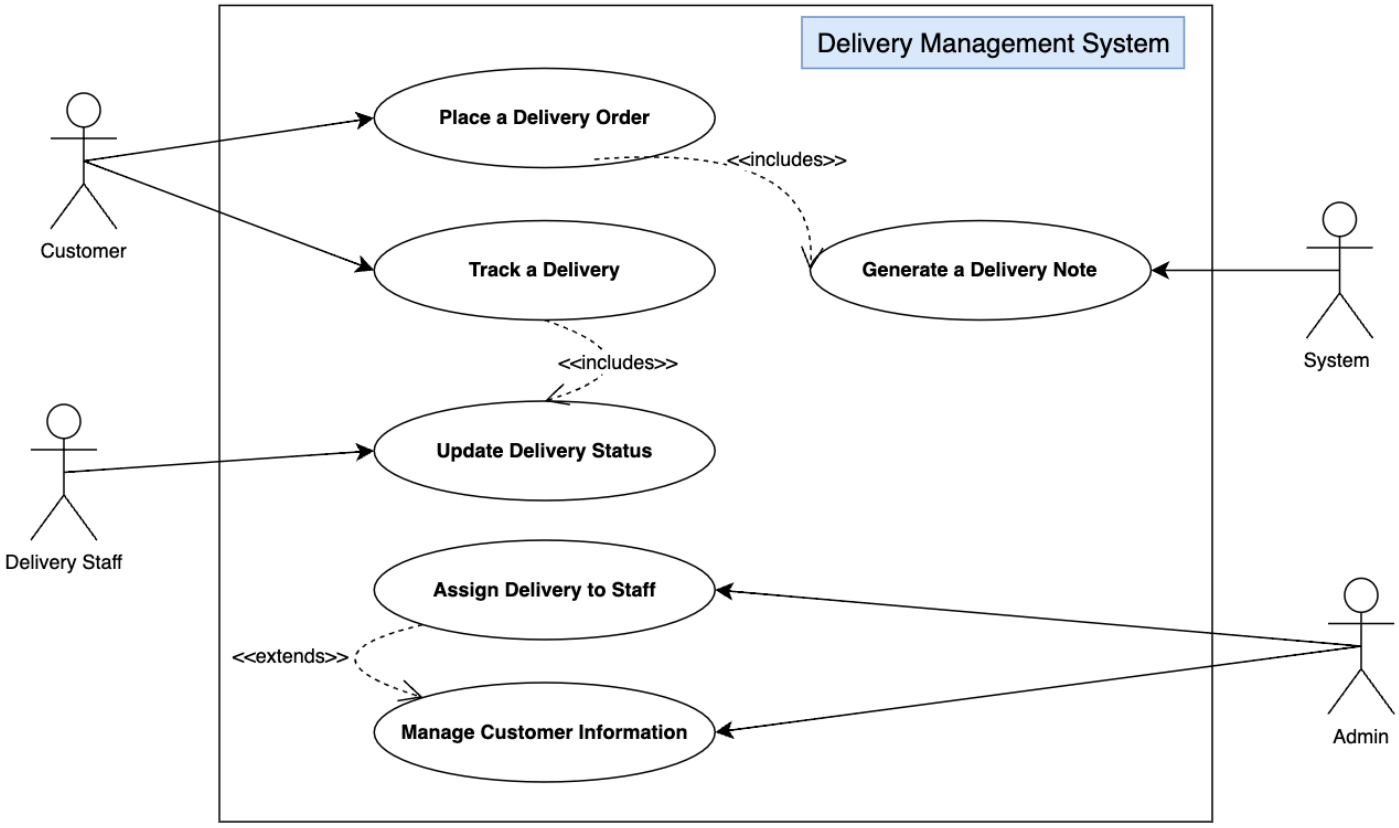
3. Generate a Delivery Note : the system automatically creates a delivery note that includes recipient details, order information, and the total cost. The delivery staff can print this note.

4. Update Delivery Status : the delivery staff will updates the system with the latest order status.

5. Assign a Delivery to Staff : the admin assigns delivery requests to available delivery staff and ensures smooth order processing.

6. Manage Customer Information : the system stores and updates customer details, such as names, contact numbers, and delivery addresses.

b. Draw the UML use-case diagram and include supporting use-case description tables using standard tools.



c. Include at least 3 scenarios. Ensure that "include" and "extend" relationships are added.

TABLE 1

User case name	Place a Delivery Order
Trigger	The customer wants to send a package.
Preconditions	- The customer must be logged into the system - The customer must provide the recipient's name, address, and package details
Main Scenarios	- The customer selects the option to place a new delivery order - The customer enters recipient details, package weight, and delivery address - The system verifies the entered information. - The system assigns a unique order number and saves the order - The system **includes** the process of generating a delivery note - The confirm message will sent to the customer
Exceptions	- If the details are incorrect, the system asks for corrections before proceeding. - If the system is down, the order cannot be placed

TABLE 2

User case name	Track a Delivery
Trigger	The customer wants to check the status of their package
Preconditions	- The order must already be placed in the system - The customer must have the tracking number
Main Scenarios	- The customer selects the (Track Delivery) option - The customer enters their tracking number - The system retrieves the current status of the package - The system **includes** the (Update Delivery Status)use-case to get real-time updates - The system will displays the latest delivery status
Exceptions	- If the tracking number is incorrect, the system will ask the customer to enter a valid one - If the package is lost, the system will notifies the customer and admin

TABLE 3

User case name	Generate a Delivery Note
Trigger	A delivery order has been successfully placed
Preconditions	- The order must be confirmed by the system - The system must have customer and order details
Main Scenarios	- The system automatically creates a delivery note - The note includes the recipient's name, order number, package weight, and total cost - The delivery staff can view and print the note - The system **includes** this use-case in the (Place a Delivery Order) use-case
Exceptions	- If order details are missing, the note cannot be generated

TABLE 4

User case name	Assign Delivery to Staff
Trigger	A new order has been placed and needs to be assigned
Preconditions	- The order must be confirmed - Delivery staff must be available
Main Scenarios	- The admin logs into the system - The admin selects an order from the list of unassigned deliveries - The admin assigns the order to an available delivery staff - The system **extends** the "Manage Customer Information" use-case by allowing the admin to update customer-related details if needed - The assigned staff receives a notification
Exceptions	- If no delivery staff is available, the order remains unassigned until staff becomes available

TABLE 5

User case name	Update Delivery Status
Trigger	The delivery staff wants to update the package's delivery status
Preconditions	- The package must already be assigned to a delivery staff member - The staff must be logged into the system
Main Scenarios	- The delivery staff logs into the system - The staff selects the order they are delivering - The staff updates the status (Out for Delivery, Delivered) - The system saves the update - The system **includes** this use-case in the (Track a Delivery) use-case so customers get real-time tracking updates
Exceptions	- If the staff enters an invalid status, the system asks for confirmation

2. Identify Objects and Classes:

a. Based on the use-case descriptions, identify the objects and their respective classes.

To design an effective Delivery Management System, we first identify the key objects that interact with the system. These objects are then categorized into classes with attributes (data) and methods (functions).

1. Customer Class
Description: will represents the user who places delivery orders and tracks them.
Attributes:

- customerID: unique identifier for each customer
- name: name of the customer
- email: contact email of the customer
- phoneNumber: contact number of the customer
- address: customer's registered address

Methods:

- placeOrder(): allows the customer to create a new delivery request
- trackOrder(): retrieves the current status of an order

2. DeliveryOrder Class
Description: it represents a delivery request placed by a customer
Attributes:

- orderID: unique identifier for the order.
- customerID: links the order to the customer who placed it
- recipientName: name of the person receiving the package
- recipientAddress: delivery location
- packageWeight: weight of the package
- status: current status of the order (Processing, Out for Delivery, Delivered).
- trackingNumber: unique tracking code for the order.

Methods:

- generateDeliveryNote(): creates a delivery note for the package
- updateStatus(): updates the order status as it moves through the delivery process

3. DeliveryStaff Class
Description: represents the employees responsible for delivering packages.
Attributes:

- staffID: unique identifier for each delivery staff member
- name: name of the delivery staff
- phoneNumber: contact number for the staff member
- assignedOrders[]: list of delivery orders assigned to the staff

Methods:

- updateDeliveryStatus(): allows staff to mark an order as (Out for Delivery)or (Delivered)

- viewAssignedOrders(): displays all orders assigned to the staff member.

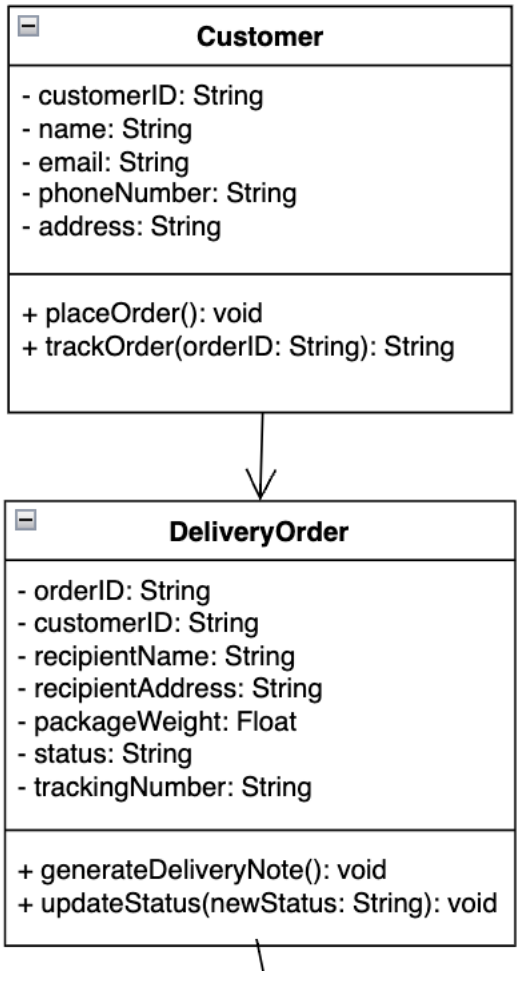
4. Admin Class
Description: is representing the system administrator who manages deliveries and staff assignments
Attributes:

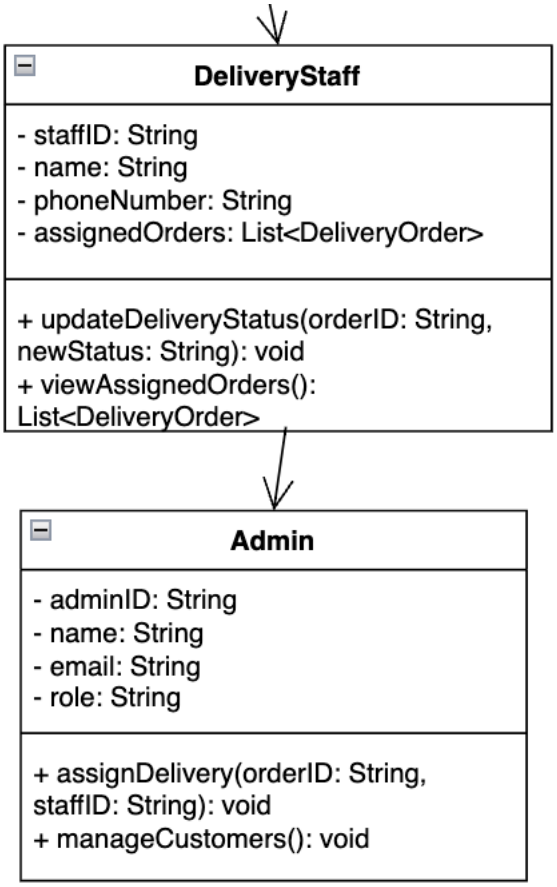
- adminID: unique identifier for the admin
- name: name of the admin
- email: contact email of the admin
- role: defines whether the admin has full system access or limited permissions

Methods:

- assignDelivery(): assigns an order to an available delivery staff member
- manageCustomers(): updates customer details if necessary

b. Draw the UML class diagram and provide supporting descriptions to explain the identified classes.
c. Include at least 4 classes and establish access specifiers for member visibility (e.g., public, private, protected).





3. Create Python Classes and Objects:

For all identified classes, write well-documented Python code to include:
a. Constructor (_ _init_ _) and at least 5 attributes per class. b. Required setter and getter methods. c. Other required function headers in the classes where the function's body is just a pass statement and include a comment to indicate what the function should achieve.

```
from enum import Enum

# Define an Enum for Order Status
class OrderStatus(Enum):
    PROCESSING = "Processing"
    OUT_FOR_DELIVERY = "Out for Delivery"
    DELIVERED = "Delivered"

class Customer:
    """Represents a customer who places delivery orders and tracks
them."""

    def __init__(self, customer_id, name, email, phone, address):
        self.__customer_id = customer_id
        self.__name = name
        self.__email = email
```

```
        self.__phone = phone
        self.__address = address

    # Getter Methods
    def getCustomerID(self):
        return self.__customer_id

    def getName(self):
        return self.__name

    def getEmail(self):
        return self.__email

    def getPhone(self):
        return self.__phone

    def getAddress(self):
        return self.__address

    # Setter Methods
    def setEmail(self, email):
        self.__email = email

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

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

class DeliveryOrder:
    """Represents a delivery request placed by a customer."""

    def __init__(self, order_id, customer_id, recipient_name,
recipient_address, package_weight, status, tracking_number):
        self.__order_id = order_id
        self.__customer_id = customer_id
        self.__recipient_name = recipient_name
        self.__recipient_address = recipient_address
        self.__package_weight = package_weight
        self.__status = status
        self.__tracking_number = tracking_number

    # Getter Methods
    def getOrderID(self):
        return self.__order_id
```

```
    def getCustomerID(self):
        return self.__customer_id

    def getRecipientName(self):
        return self.__recipient_name

    def getRecipientAddress(self):
        return self.__recipient_address

    def getPackageWeight(self):
        return self.__package_weight

    def getStatus(self):
        return self.__status

    def getTrackingNumber(self):
        return self.__tracking_number

    # Setter Methods
    def setStatus(self, status):
        if isinstance(status, OrderStatus):
            self.__status = status
        else:
            raise ValueError("Invalid order status")

class DeliveryStaff:
    """Represents delivery personnel responsible for handling packages."""

    def __init__(self, staff_id, name, phone):
        self.__staff_id = staff_id
        self.__name = name
        self.__phone = phone
        self.__assigned_orders = []

    # Getter Methods
    def getStaffID(self):
        return self.__staff_id

    def getName(self):
        return self.__name

    def getPhone(self):
        return self.__phone
```

```
    def getAssignedOrders(self):
        return self.__assigned_orders

class Admin:
    """Represents an admin responsible for managing orders and delivery
staff."""

    def __init__(self, admin_id, name, email, role):
        self.__admin_id = admin_id
        self.__name = name
        self.__email = email
        self.__role = role

    # Getter Methods
    def getAdminID(self):
        return self.__admin_id

    def getName(self):
        return self.__name

    def getEmail(self):
        return self.__email

    def getRole(self):
        return self.__role
```

4. Use objects to generate a Delivery Note:

a. Create objects of all identified classes.

b. Use these objects and their functions to populate and display the information provided in the figure.

```
from enum import Enum

# Define an Enum for Order Status
class OrderStatus(Enum):
    PROCESSING = "Processing"
    OUT_FOR_DELIVERY = "Out for Delivery"
    DELIVERED = "Delivered"

class Customer:
    """Represents a customer who places delivery orders and tracks
them."""

    def __init__(self, customer_id, name, email, phone, address):
```

```
        self.__customer_id = customer_id
        self.__name = name
        self.__email = email
        self.__phone = phone
        self.__address = address

    # Getter Methods
    def getCustomerID(self):
        return self.__customer_id

    def getName(self):
        return self.__name

    def getEmail(self):
        return self.__email

    def getPhone(self):
        return self.__phone

    def getAddress(self):
        return self.__address

    # Setter Methods
    def setEmail(self, email):
        self.__email = email

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

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

class DeliveryOrder:
    """Represents a delivery request placed by a customer."""

    def __init__(self, order_id, customer_id, recipient_name,
recipient_address, package_weight, status, tracking_number):
        self.__order_id = order_id
        self.__customer_id = customer_id
        self.__recipient_name = recipient_name
        self.__recipient_address = recipient_address
        self.__package_weight = package_weight
        self.__status = status
        self.__tracking_number = tracking_number
```

```
    # Getter Methods
    def getOrderID(self):
        return self.__order_id

    def getCustomerID(self):
        return self.__customer_id

    def getRecipientName(self):
        return self.__recipient_name

    def getRecipientAddress(self):
        return self.__recipient_address

    def getPackageWeight(self):
        return self.__package_weight

    def getStatus(self):
        return self.__status

    def getTrackingNumber(self):
        return self.__tracking_number

    # Setter Methods
    def setStatus(self, status):
        if isinstance(status, OrderStatus):
            self.__status = status
        else:
            raise ValueError("Invalid order status")

class DeliveryStaff:
    """Represents delivery personnel responsible for handling packages."""

    def __init__(self, staff_id, name, phone):
        self.__staff_id = staff_id
        self.__name = name
        self.__phone = phone
        self.__assigned_orders = []

    # Getter Methods
    def getStaffID(self):
        return self.__staff_id

    def getName(self):
        return self.__name
```

```
    def getPhone(self):
        return self.__phone

    def getAssignedOrders(self):
        return self.__assigned_orders

class Admin:
    """Represents an admin responsible for managing orders and delivery
staff."""

    def __init__(self, admin_id, name, email, role):
        self.__admin_id = admin_id
        self.__name = name
        self.__email = email
        self.__role = role

    # Getter Methods
    def getAdminID(self):
        return self.__admin_id

    def getName(self):
        return self.__name

    def getEmail(self):
        return self.__email

    def getRole(self):
        return self.__role

# Use Objects to Generate the Delivery Note
customer1 = Customer("C101", "Salma Almansoori",
"salma.almansoori@example.com", "0501234567", "45 Knowledge Ave, Dubai")
order1 = DeliveryOrder("O2025", customer1.getCustomerID(), "Sarah
Johnson", "45 Knowledge Ave, Dubai", 7.0, OrderStatus.PROCESSING,
"TRK2025001")
staff1 = DeliveryStaff("S201", "Mohamed Almansoori", "0500001234")
staff1.getAssignedOrders().append(order1)

# Print the Delivery Note
print("\n" + "="*40)
print(" " * 10 + "DELIVERY NOTE")
print("="*40)
print("Thank you for using our delivery service!")
print("Please print your delivery receipt and present it upon receiving
your items.\n")
```

```
print("Recipient Details:")
print(f"Name: {order1.getRecipientName()}")
print(f"Contact: {customer1.getEmail()}")
print(f"Delivery Address: {order1.getRecipientAddress()}\n")

print("Delivery Information:")
print(f"Order Number: {order1.getOrderID()}")
print(f"Tracking Number: {order1.getTrackingNumber()}")
print(f"Delivery Date: January 25, 2025")
print(f"Delivery Method: Courier")
print(f"Package Weight: {order1.getPackageWeight()} kg\n")

print("Summary of Items Delivered:")
print(f"{'Item Code':<15}{'Description':<30}{'Quantity':<10}{'Unit Price
(AED)':<20}{'Total Price (AED)':<20}")
print("-"*90)
print(f"{'ITM001':<15}{'Wireless
Keyboard':<30}{'1':<10}{'100.00':<20}{'100.00':<20}")
print(f"{'ITM002':<15}{'Wireless Mouse &
Pad':<30}{'1':<10}{'75.00':<20}{'75.00':<20}")
print(f"{'ITM003':<15}{'Laptop Cooling
Pad':<30}{'1':<10}{'120.00':<20}{'120.00':<20}")
print(f"{'ITM004':<15}{'Camera
Lock':<30}{'3':<10}{'15.00':<20}{'45.00':<20}")
print("-"*90)

subtotal = 270.00
taxes = 13.50
total = subtotal + taxes
print(f"\nSubtotal: AED {subtotal:.2f}")
print(f"Taxes and Fees: AED {taxes:.2f}")
print(f"Total Charges: AED {total:.2f}")
print("="*40)
```

5. GitHub

https://github.com/Salmnns/Delivery-Management-System-

6. Summary of learnings

In this assignment, I learned how to design and implement a Delivery Management System using object-oriented programming (OOP) and UML diagrams. At first, understanding use-case diagrams and class diagrams was difficult, but as I practiced, I saw how they help organize a system before coding. I also learned how to create Python classes with constructors, getters, setters, and methods. Using Encapsulation (private attributes) made the code more secure and organized. At first, I struggled with defining relationships between classes, but breaking down the system into Customer, DeliveryOrder, Admin, and DeliveryStaff helped.

Generating the Delivery Note using Python was fun because I saw how different classes work together in a real system. I also learned how to use GitHub to upload my code and share it in my document. At first, I got confused about formatting my code in Microsoft Word, but I fixed it by using Courier New font and keeping indentation correct.

Overall, this assignment helped me improve my coding, problem-solving, and documentation skills. I also learned how real-world delivery systems work, which makes coding more interesting and useful.