Delivery Management System

Nasser Nabeel Lootah

202221929

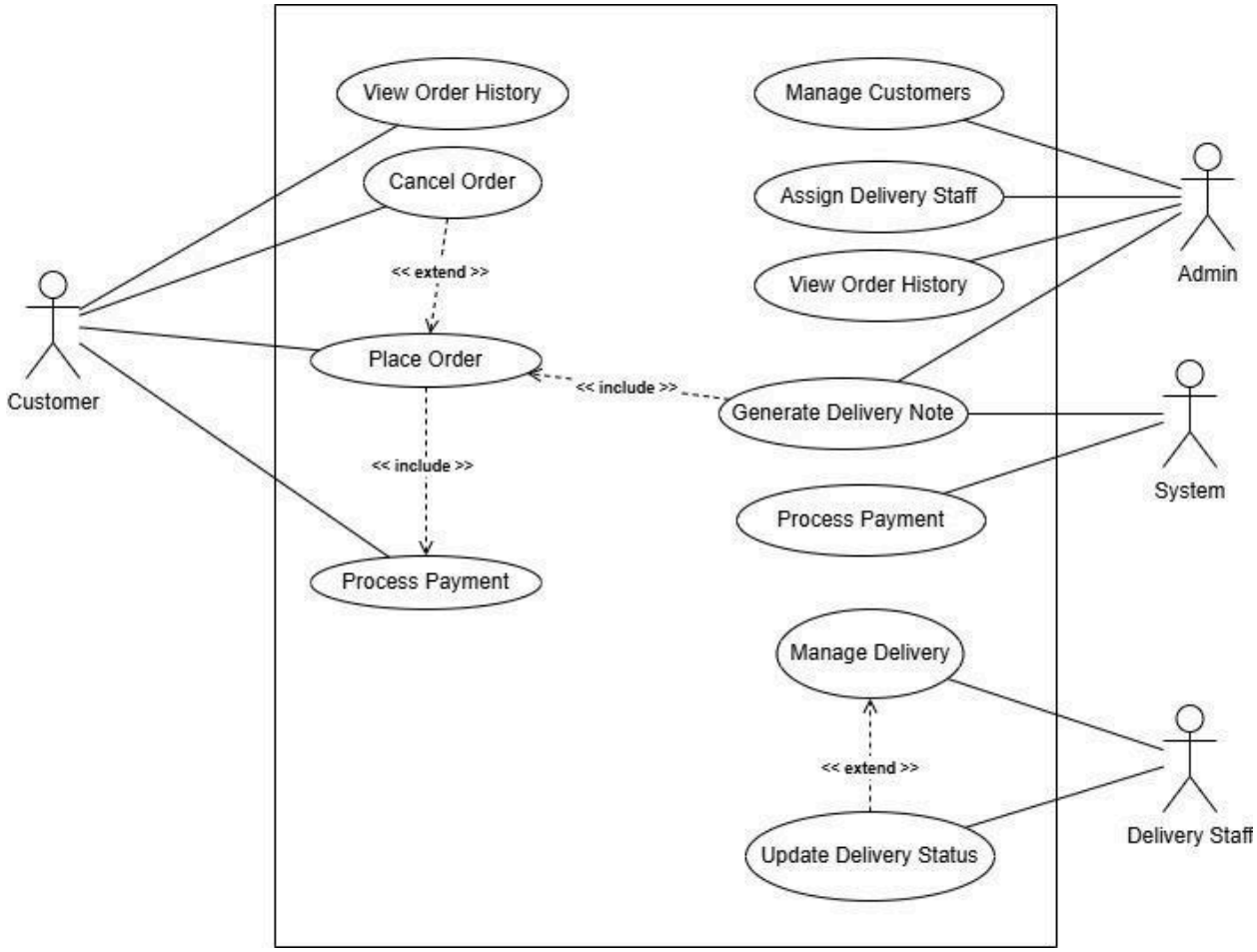
Zayed University

ICS220 > 21383 Program. Fund.

February 28, 2025

Professor Leonce

UML Use-Case



Actors and use cases

Actors represent users or external systems that interact with the system.

- **Customer** (Places orders, cancels orders, makes payments, views history)
- **Delivery Staff** (Manages deliveries, updates status)
- **Admin** (Manages customers, assigns delivery staff, generates reports)
- **System (Automatic)** (Handles payments, generates delivery notes)

Include Relationships:

- Payment is always required when placing an order.
- Delivery note is always generated after placing an order.

Extend Relationship:

- Cancelling is only possible if the order isn't dispatched.
- Updating status is part of delivery but depends on staff actions

Use Case 1: Place Order

Use Case	Place Order
Trigger	The customer wants to place a new delivery order.
Preconditions	The customer must be logged into the system.
Main Scenarios	1. The customer enters the recipient's details (name, address, contact). 2. The customer selects the type of package (size, weight). 3. The system calculates the estimated cost based on package details. 4. The customer confirms the order and selects a payment method. 5. The system generates a unique order ID and provides confirmation.

Exceptions	4a. If mandatory details are missing, the system prompts the user to fill them. 4b. If the entered address is invalid, the system asks for corrections. 5a. If payment fails, the system notifies the customer and requests a retry.

Use Case 2: Manage Delivery

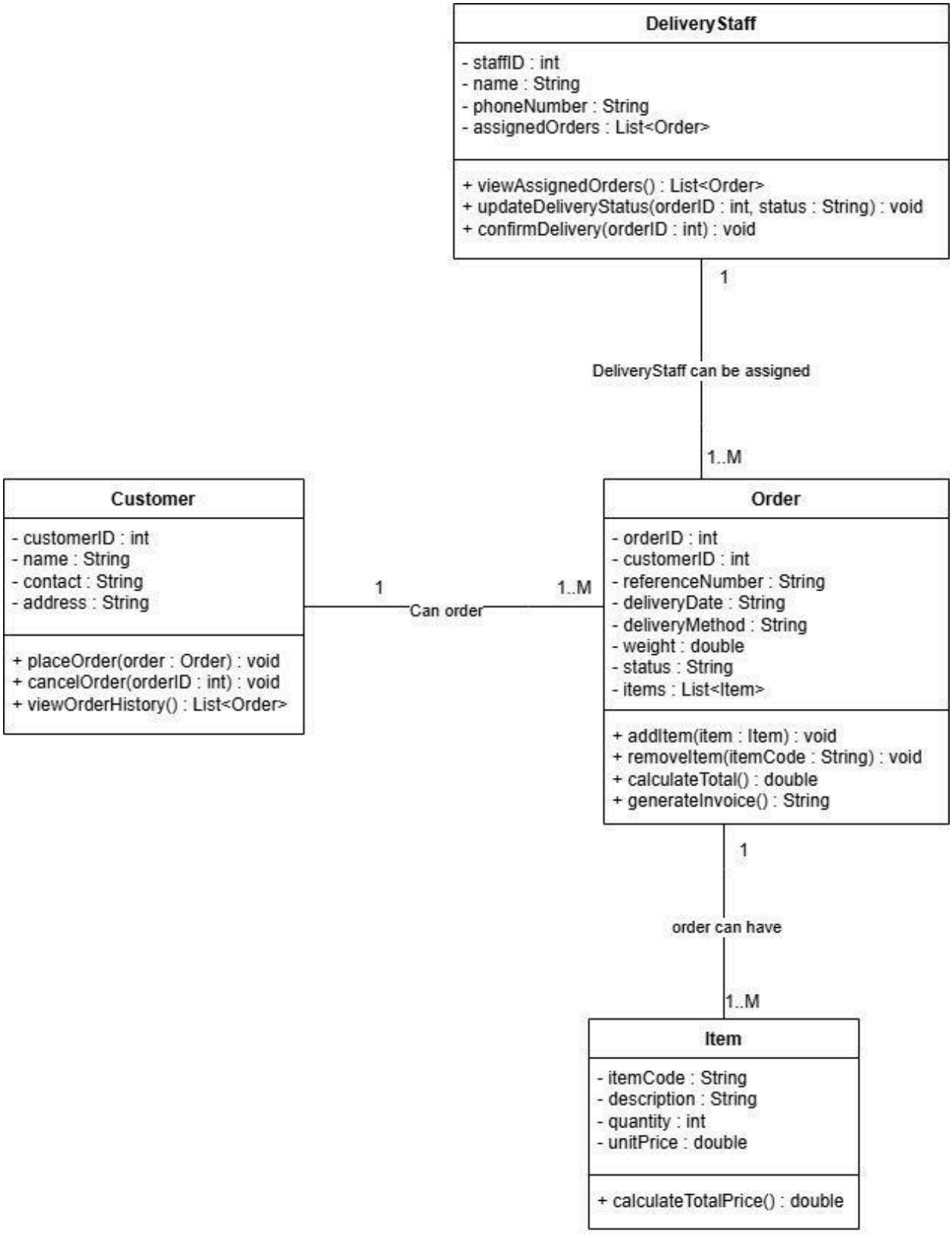
Use Case	Manage Delivery
Trigger	The delivery staff wants to update the status of an assigned order.
Preconditions	The delivery staff must be logged in and assigned to the order.
Main Scenarios	1. The delivery staff views the list of assigned deliveries. 2. The delivery staff selects an order to update. 3. The system displays the current status (Pending, In Transit, Delivered). 4. The delivery staff updates the status based on progress. 5. The system records the update and notifies the customer.
Exceptions	4a. If the order is already marked as delivered, the system prevents changes. 4b. If an error occurs while updating, the system logs the issue and asks for a retry.

Use Case 3: Generate Delivery Note

Use Case	Generate Delivery Note
Trigger	The admin/system generates a delivery note for completed orders.
Preconditions	The order must be marked as "Delivered."

Main Scenarios	1. The system retrieves order details from the database. 2. The system generates a delivery note with order ID, recipient, and delivery details. 3. The admin reviews and approves the generated delivery note. 4. The system saves the note and makes it available for download. 5. The customer receives a notification with the delivery note.
Exceptions	3a. If the order is not marked as "Delivered," the system prevents note generation. 4a. If an error occurs in note generation, the system retries and logs the issue.

UML Class Diagram



UML Class Relationship

1 Customer → places multiple Orders (1-to-Many).

1 Order → contains multiple Items (1-to-Many).

1 Order → is linked to 1 Customer (Many-to-1).

1 Order → is assigned to 1 DeliveryStaff (1-to-1).

1 Order → has multiple Items (1-to-Many).

1 DeliveryStaff → can be assigned multiple Orders (1-to-Many).

Python Code

```
class Customer:
    def __init__(self, customer_id, name, contact, address):
        self.__customer_id = customer_id
        self.__name = name
        self.__contact = contact
        self.__address = address

    def get_customer_details(self):
        """Returns customer details as a formatted string."""
        return f"Customer: {self.__name}, \nContact: {self.__contact},
\nAddress: {self.__address}"



class Order:
    def __init__(self, order_id, reference_number, customer, delivery_date,
delivery_method, weight):
        self.__order_id = order_id
        self.__reference_number = reference_number
        self.__customer = customer
        self.__delivery_date = delivery_date
        self.__delivery_method = delivery_method
        self.__weight = weight
        self.__items = []

    def add_item(self, item):
        """Adds an item to the order."""
        self.__items.append(item)

    def get_order_summary(self):
        """Returns a formatted order summary including all items."""
        summary = f"Order ID: {self.__order_id}, \nReference Number:
{self.__reference_number}, \nDelivery Date: {self.__delivery_date}, \nMethod:
{self.__delivery_method}, \nWeight: {self.__weight}kg\n"
        summary += "\n\nItems:\n"
        for item in self.__items:
```

```
            summary += item.get_item_details() + "\n"
        return summary


class Item:
    def __init__(self, item_code, description, quantity, unit_price):
        self.__item_code = item_code
        self.__description = description
        self.__quantity = quantity
        self.__unit_price = unit_price

    def get_item_details(self):
        """Returns item details including item code."""
        return f"Item Code: {self.__item_code}, Description:
{self.__description}, Qty: {self.__quantity}, Unit Price: AED
{self.__unit_price}, Total: AED {self.__quantity * self.__unit_price}"


class DeliveryStaff:
    def __init__(self, staff_id, name, phone_number):
        self.__staff_id = staff_id
        self.__name = name
        self.__phone_number = phone_number

    def get_staff_details(self):
        """Returns delivery staff details."""
        return f"Delivery Staff: {self.__name}, Contact:
{self.__phone_number}"


# Example Usage
customer = Customer(101, "Sarah Johnson", "sarah.johnson@example.com", "45
Knowledge Avenue, Dubai, UAE")
order = Order("DEL123456789", "DN-2025-001", customer, "January 25, 2025",
"Courier", 7)

# Adding all 4 items as per sample output
item1 = Item("ITM001", "Wireless Keyboard", 1, 100.00)
```

```
item2 = Item("ITM002", "Wireless Mouse & Pad Set", 1, 75.00)
item3 = Item("ITM003", "Laptop Cooling Pad", 1, 120.00)
item4 = Item("ITM004", "Camera Lock", 3, 15.00)

order.add_item(item1)
order.add_item(item2)
order.add_item(item3)
order.add_item(item4)

delivery_staff = DeliveryStaff(201, "John Doe", "+971501234567")

# Printing details
print("Recipient Details: ")
print(customer.get_customer_details())

print("\nDelivery Information:")
print(order.get_order_summary())

print(delivery_staff.get_staff_details())
```

Output

```
Recipient Details:
Customer: Sarah Johnson,
Contact: sarah.johnson@example.com,
Address: 45 Knowledge Avenue, Dubai, UAE

Delivery Information:
Order ID: DEL123456789,
Reference Number: DN-2025-001,
Delivery Date: January 25, 2025,
Method: Courier,
Weight: 7kg


Items:
Item Code: ITM001, Description: Wireless Keyboard, Qty: 1, Unit Price: AED 100.0, Total: AED 100.0
Item Code: ITM002, Description: Wireless Mouse & Pad Set, Qty: 1, Unit Price: AED 75.0, Total: AED 75.0
Item Code: ITM003, Description: Laptop Cooling Pad, Qty: 1, Unit Price: AED 120.0, Total: AED 120.0
Item Code: ITM004, Description: Camera Lock, Qty: 3, Unit Price: AED 15.0, Total: AED 45.0

Delivery Staff: John Doe, Contact: +971501234567
```

GitHub link

https://github.com/NasserLootah/NasserLootahCode?tab=readme-ov-file#nasserlootahcode

Summary of learning

In this assignment, I learned how to design and implement a delivery management system using object-oriented programming (OOP). I understood how to identify important parts of the system, like customers, orders, items, and delivery staff, and represent them as classes with attributes and functions.

Creating the UML diagrams helped me see how different parts of the system connect and work together. Writing Python classes also helped me practice using private attributes and getter methods to keep the data safe and well-organized.

One important thing I learned is that OOP makes coding easier to manage because it allows for reusable and structured code. Overall, this assignment helped me improve my Python skills, problem-solving, and understanding of system design in a simple and practical way.