Assignment 1

Moza Abdulla Alhemeiri

College of Interdisciplinary Studies, Zayed University

ICS 220 - 22111 Programming Fundamentals

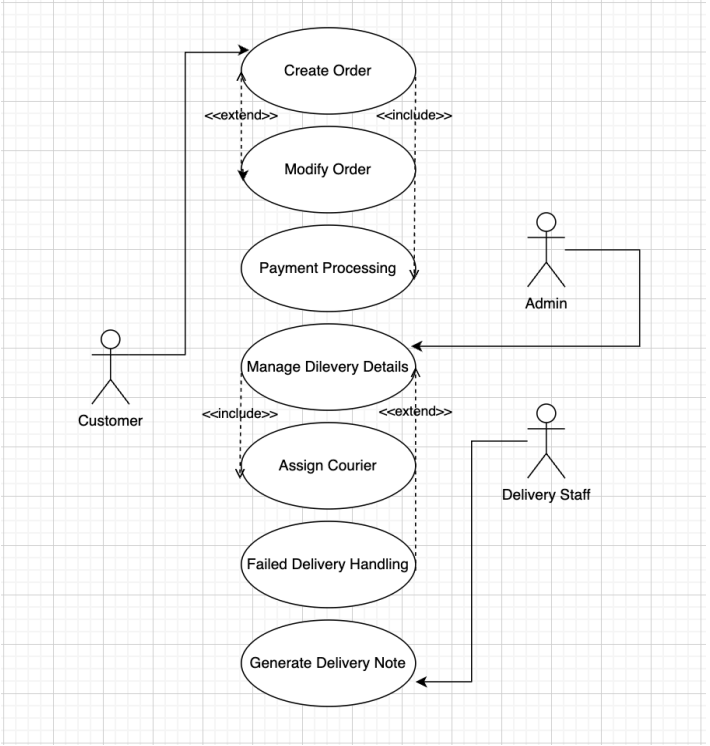
Prof. Sujith Mathew

February 28, 2025

1. Identify Use-Cases

- Create Order
- Manage Delivery Details
- Generate Delivery Note
- Payment Processing
- Assign Courier
- Modify Order
- Failed Delivery Handling

UML Use-Case Diagram



Use-Case	Description
Create Order	Customer places an order by providing delivery details and items.
Modify Order	Customer modifies the order (e.g., updates address, adds items).
Payment Processing	Payment is processed for the order (verify payment success).
Manage Delivery Details	Admin or delivery staff track and update delivery status and details.
Assign Courier	Admin assigns a courier to the order based on location or availability.
Failed Delivery Handling	Delivery staff handle failed deliveries by retrying or modifying the delivery plan.
Generate Delivery Note	Admin generates and assigns the delivery note to the courier.

2. Identify Objects and Classes

Customer:

Represents the customer placing an order. This object manages customer-related details like name, contact, and address.

Order:

Represents an order made by the customer, which consists of multiple items. The Order class manages the order number, items, and total price.

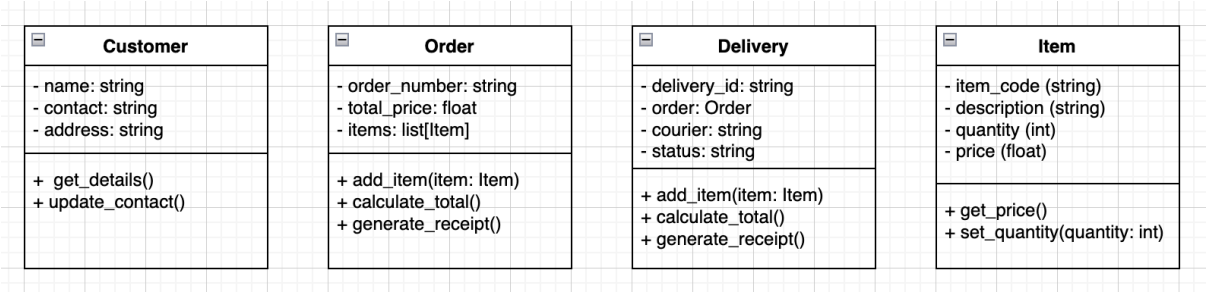
Delivery:

Represents the delivery of the order, with details like the delivery ID, the order being delivered, courier assigned, and delivery status.

Item:

Represents a product that can be included in an order. Each item has details like item code, description, quantity, and price.

UML Class Diagram



Relationships:

- Customer is associated with Order (one customer can place many orders).
- Order contains multiple Items (one order can have multiple items).
- Order is associated with Delivery (one order can have one delivery).
- Delivery involves Item(s) (delivery will involve the items present in the order).

Private (__): Used for attributes that should not be directly accessible from outside the class, e.g., __name, __contact, __order_number.

Public (public): Used for methods and attributes that need to be accessed by other classes, e.g., get_details(), add_item().

3. Create Python Classes and Objects

```
# Class Definitions

class Customer:
    """
    This class represents a customer in the delivery management system.
    """

    def __init__(self, name, contact, address, email, customer_id):
        """
        Constructor to initialize customer details.

        Args:
        - name (str): Name of the customer.
        - contact (str): Contact number of the customer.
        - address (str): Address of the customer.
        - email (str): Email of the customer.
        - customer_id (str): Unique ID for the customer.
        """
        self.__name = name
        self.__contact = contact
        self.__address = address
        self.__email = email
        self.__customer_id = customer_id

    # Getter and Setter Methods
    def get_name(self):
        """Returns the name of the customer."""
        return self.__name

    def set_name(self, name):
        """Sets the name of the customer."""
        self.__name = name

    def get_contact(self):
        """Returns the contact number of the customer."""
        return self.__contact

    def set_contact(self, contact):
        """Sets the contact number of the customer."""
        self.__contact = contact
```

```
    def get_address(self):
        """Returns the address of the customer."""
        return self.__address

    def set_address(self, address):
        """Sets the address of the customer."""
        self.__address = address

    def get_email(self):
        """Returns the email address of the customer."""
        return self.__email

    def set_email(self, email):
        """Sets the email address of the customer."""
        self.__email = email

    def get_customer_id(self):
        """Returns the unique customer ID."""
        return self.__customer_id

    def set_customer_id(self, customer_id):
        """Sets the unique customer ID."""
        self.__customer_id = customer_id

    def update_contact(self, new_contact):
        """Updates the contact number of the customer."""
        self.__contact = new_contact

    def get_details(self):
        """Returns a formatted string of customer details."""
        return f"Customer: {self.__name}, Contact: {self.__contact}, Address: {self.__address},
Email: {self.__email}"


class Item:
    """
    This class represents an item in an order for the delivery management system.
    """

    def __init__(self, item_code, description, quantity, price, weight):
        """
        Constructor to initialize item details.

        Args:
```

```
    - item_code (str): Unique identifier for the item.
    - description (str): Description of the item.
    - quantity (int): Quantity of the item in the order.
    - price (float): Price per unit of the item.
    - weight (float): Weight of the item.
    """
    self.__item_code = item_code
    self.__description = description
    self.__quantity = quantity
    self.__price = price
    self.__weight = weight

# Getter and Setter Methods
def get_item_code(self):
    """Returns the item code."""
    return self.__item_code

def set_item_code(self, item_code):
    """Sets the item code."""
    self.__item_code = item_code

def get_description(self):
    """Returns the description of the item."""
    return self.__description

def set_description(self, description):
    """Sets the description of the item."""
    self.__description = description

def get_quantity(self):
    """Returns the quantity of the item."""
    return self.__quantity

def set_quantity(self, quantity):
    """Sets the quantity of the item."""
    self.__quantity = quantity

def get_price(self):
    """Returns the price of the item."""
    return self.__price

def set_price(self, price):
    """Sets the price of the item."""
    self.__price = price
```

```
    def get_weight(self):
        """Returns the weight of the item."""
        return self.__weight

    def set_weight(self, weight):
        """Sets the weight of the item."""
        self.__weight = weight

    def calculate_total_price(self):
        """Calculates the total price for this item based on quantity."""
        return self.__price * self.__quantity

    def calculate_total_weight(self):
        """Calculates the total weight for this item based on quantity."""
        return self.__weight * self.__quantity


class Order:
    """
    This class represents an order made by the customer in the delivery management system.
    """

    def __init__(self, order_number, customer, order_date, delivery_date, status):
        """
        Constructor to initialize order details.

        Args:
        - order_number (str): Unique identifier for the order.
        - customer (Customer): Customer who placed the order.
        - order_date (str): Date when the order was placed.
        - delivery_date (str): Date when the order is expected to be delivered.
        - status (str): Current status of the order.
        """
        self.__order_number = order_number
        self.__customer = customer
        self.__order_date = order_date
        self.__delivery_date = delivery_date
        self.__status = status
        self.__items = []
        self.__total_price = 0.0

    # Getter and Setter Methods
    def get_order_number(self):
```

```
    """Returns the order number."""
    return self.__order_number

def set_order_number(self, order_number):
    """Sets the order number."""
    self.__order_number = order_number

def get_order_date(self):
    """Returns the order date."""
    return self.__order_date

def set_order_date(self, order_date):
    """Sets the order date."""
    self.__order_date = order_date

def get_delivery_date(self):
    """Returns the delivery date."""
    return self.__delivery_date

def set_delivery_date(self, delivery_date):
    """Sets the delivery date."""
    self.__delivery_date = delivery_date

def get_status(self):
    """Returns the order status."""
    return self.__status

def set_status(self, status):
    """Sets the order status."""
    self.__status = status

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

def get_total_price(self):
    """Returns the total price for the order."""
    return self.__total_price

def generate_receipt(self):
    """Generates the receipt for the order."""
    print(f"Order Number: {self.__order_number}")
    print(f"Customer: {self.__customer.get_name()}")
```

```
        print(f"Order Date: {self.__order_date}")
        for item in self.__items:
            print(f"{item.get_description()} x {item.get_quantity()} =
{item.calculate_total_price()} AED")
        print(f"Total Price: {self.__total_price} AED")

    def cancel_order(self):
        """Cancels the order (sets status to 'Cancelled')."""
        self.__status = "Cancelled"


class Delivery:
    """
    This class represents the delivery of an order in the delivery management system.
    """

    def __init__(self, delivery_id, order, delivery_date, courier, status):
        """
        Constructor to initialize delivery details.

        Args:
        - delivery_id (str): Unique identifier for the delivery.
        - order (Order): The order that is being delivered.
        - delivery_date (str): Date of delivery.
        - courier (str): Name of the assigned courier.
        - status (str): Current delivery status.
        """
        self.__delivery_id = delivery_id
        self.__order = order
        self.__delivery_date = delivery_date
        self.__courier = courier
        self.__status = status

    # Getter and Setter Methods
    def get_delivery_id(self):
        """Returns the delivery ID."""
        return self.__delivery_id

    def set_delivery_id(self, delivery_id):
        """Sets the delivery ID."""
        self.__delivery_id = delivery_id

    def get_order(self):
        """Returns the associated order."""
```

```
    return self.__order

def set_order(self, order):
    """Sets the associated order."""
    self.__order = order

def get_delivery_date(self):
    """Returns the delivery date."""
    return self.__delivery_date

def set_delivery_date(self, delivery_date):
    """Sets the delivery date."""
    self.__delivery_date = delivery_date

def get_courier(self):
    """Returns the name of the assigned courier."""
    return self.__courier

def set_courier(self, courier):
    """Sets the name of the assigned courier."""
    self.__courier = courier

def get_status(self):
    """Returns the status of the delivery."""
    return self.__status

def set_status(self, status):
    """Sets the status of the delivery."""
    self.__status = status

def update_status(self, new_status):
    """Updates the status of the delivery."""
    self.__status = new_status

def assign_courier(self, courier_name):
    """Assigns a courier to the delivery."""
    self.__courier = courier_name

def mark_as_delivered(self):
    """Marks the delivery as completed."""
    self.__status = "Delivered"
```

4. **Use objects to generate a Delivery Note:**

Creating objects for the classes

Customer Object

customer1 = Customer(name="John Doe", contact="0501234567", address="123 Main St, Cityville", email="john.doe@email.com", customer_id="C001")

Item Objects

item1 = Item(item_code="I001", description="Laptop", quantity=1, price=5000.00, weight=2.5)

item2 = Item(item_code="I002", description="Wireless Mouse", quantity=2, price=150.00, weight=0.1)

Order Object

order1 = Order(order_number="O12345", customer=customer1, order_date="2025-02-28", delivery_date="2025-03-02", status="Processing")

order1.add_item(item1)

order1.add_item(item2)

Delivery Object

delivery1 = Delivery(delivery_id="D001", order=order1, delivery_date="2025-03-02", courier="Jane Smith", status="Out for Delivery")

Generating the Delivery Note

def generate_delivery_note():

```
    """
    Generates and prints a formatted delivery note using the customer, order, and delivery
details.
    """
    print("########################## DELIVERY NOTE
##########################")
    print(f"Delivery ID: {delivery1.get_delivery_id()}")
    print(f"Order Number: {order1.get_order_number()}")
    print(f"Customer Name: {customer1.get_name()}")
    print(f"Customer Contact: {customer1.get_contact()}")
    print(f"Customer Email: {customer1.get_email()}")
    print(f"Delivery Date: {delivery1.get_delivery_date()}")
    print(f"Courier: {delivery1.get_courier()}")
    print(f"Delivery Status: {delivery1.get_status()}")
    print("\nItems:")

    # Displaying item details in the order
    for item in order1.get_items():
        print(f"{item.get_description()} (x{item.get_quantity()}) -
{item.calculate_total_price()} AED")

    print(f"Total Order Price: {order1.get_total_price()} AED")
    print("\nShipping Address:")
    print(f"{customer1.get_address()}")
    print("\n#########################################################################")

# Call the function to generate the delivery note
generate_delivery_note()
```

5. Summary of learnings

In this assignment, I learned how to use Object-Oriented Programming (OOP) principles to create a delivery management system. By creating classes for customers, orders, items, and deliveries, I improved my skills in organizing and managing data. I also learned how to use UML diagrams to plan the structure of the system and how different components interact. Writing the Python code helped me understand how to design and use classes, methods, and attributes effectively. Overall, this assignment taught me how to break down a system into smaller parts and build it step by step.