# Assignment 1: Delivery Management System

Asma M. Aldahmani 202309162

Zayed University, Collage of Interdisciplinary Studies

IDS220: Programing fundamentals

Dr. Areej Abdulfattah

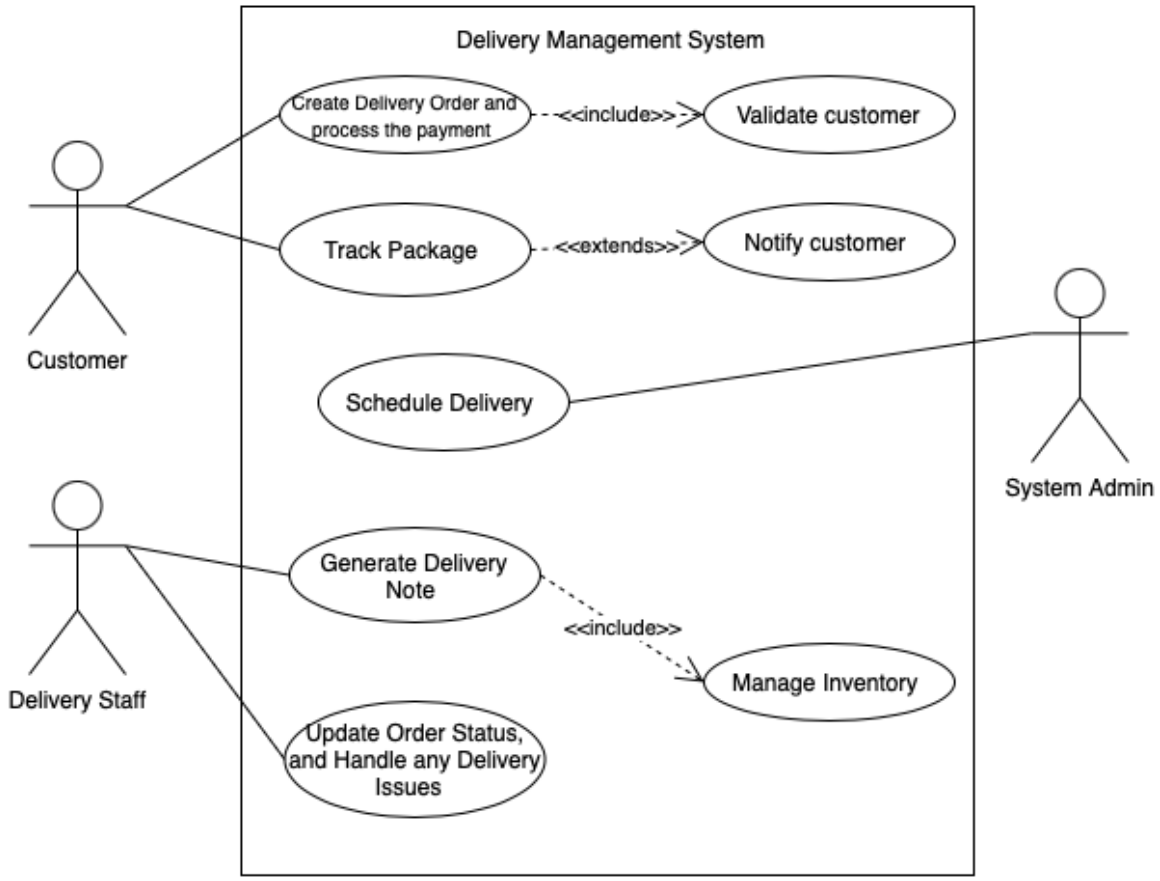
February 27, 2025

1. **Identify Use-Cases**

**a. Use Cases for the Delivery Management System:**

1. Create Delivery Order and Process the Payment
2. (include) Validate Customer
3. Track Package
4. (extend) notify customer
5. Schedule Delivery
6. Generate Delivery Note
7. (include) Manage inventory
8. Update Order Status and Handle any Delivery Issues

**b. UML Use-Case Diagram Visualization:**




Link: [Assignment 1 task 1 .drawio-2.html]

The diagram shows who uses the delivery system and what they can do with it. Customers can place delivery orders, process payment and track packages, while delivery staff handle order updates, and delivery issues and create delivery receipts. Some functions automatically include others, like checking customer details during order creation. Other features are optional, such as sending notifications when tracking updates happen.

**c. Use-Case Description Tables for Three Main Scenarios:**

Use Case 1: Create Delivery Order

| Aspects | Description |
|---|---|
| Use Case | Create Delivery Order and the Process Payment |
| Trigger | 1. Customer initiates a new delivery order request<br>2. confirm the payment process |
| Precondition | 1. Customer is registered in the system<br>2. Items are available for delivery<br>3. customer payment method validity |
| Actors | Customer<br>System |
| Main scenarios | 1. Customer logs into the system<br>2. Customer selects items for delivery<br>3. System validates item availability<br>4. Customer submits their delivery address together with relevant contact information to the system.<br>5. The customer chooses both their preferred date along with delivery method.<br>6. Total cost and delivery fees get computed by the system.<br>7. Customer confirms order details<br>8. The system appears a new delivery order before it distributes an order number to it.<br>9. Customer choose their payment method<br>10. System redirects to payment processing<br>11. The customer pay for the order<br>12. The system indicates successful processing of the payment. |
| Exceptional Scenarios | 1. Items are out of stock: the system will notify buyers and share suitable replacement options.<br>2. Invalid delivery address: The system requires customers to fix the information.<br>3. Delivery date unavailable: The system presents alternative time slots to the customer.<br>4. Payment method is invalid/ expired: System reject the order, appearing a message to retry. |
| Includes | Validate Customer |

Use Case 2: Generate Delivery Note

| Aspects | Description |
|---|---|
| Use Case | Generate Delivery Note |
| Trigger | Order is confirmed and payment is processed |
| Precondition | 1.Order has been created<br>2. Payment has been successfully processed<br>3. Order is scheduled for delivery |
| Actors | Delivery Staff<br>System |
| Main scenarios | 1. The delivery staff chooses which order needs processing at this time.<br>2. The system retrieves every detail needed for a specific order including customer information together with items and delivery specifications.<br>3. System applies final calculations to produce costs after tax along with related fees.<br>4. Letting the system create a delivery note which receives its own particular reference number.<br>5. System displays the delivery note<br>6. The delivery staff obtains the delivery note by printing it from the system.<br>7. System updates the order status to “Ready for Delivery” through its operations. |
| Exceptional Scenarios | 1. Order information incomplete: System prompts personnel to finish all missing information.<br>2. Printing error: System offers option to email the delivery note<br>3. Item availability changed: The system acts as an alert system with suggestion features for staff adjustments. |
| Includes | Manage Inventory |

Use Case 3: Track Packag

| Aspects | Description |
|---|---|
| Use Case | Track Package |
| Trigger | Customer or delivery staff requests package tracking information |
| Precondition | 1. Valid order exists in the system<br>2. Order has been processed |
| Actors | Customer<br>Delivery Staff<br>System |
| Main scenarios | 1. User enters order number/reference number<br>2. System validates the order number<br>3. The system obtains status and location data at the present moment.<br>4. The system shows tracking information that contains delivery time predictions.<br>5. User views the tracking details |
| Exceptional Scenarios | 1. Invalid order number: System displays error message<br>2. Tracking information unavailable: The system presents the previously recorded status from an unknown time<br>3. Delivery delayed: System provides reason for delay and updated estimate |
| Extends | Notify Customer (optional notification when tracking status changes) |

Use Case 4: Update Order Status and Handle Delivery Issues

| Aspects | Description |
|---|---|
| Use Case | Update Order Status and Handle Delivery Issues |
| Trigger | 1. Status change in delivery process<br>2. Delivery issue reported by customer or staff |
| Precondition | 1. Valid order exists in the system<br>2. User has appropriate access rights |
| Actors | Delivery Staff<br>Customer<br>System |
| Main scenarios | 1. The delivery staff member uses the system to access the platform.<br>2. Staff searches and picks a particular order from the system database.<br>3. The system presents current order data as well as comprehensive order information.<br>4. The staff team updates the status descriptions (e.g., in transit, delivered, delayed).<br>5. Staff members take note of any problems that arise during their work.<br>6. Staff applies appropriate resolution action<br>7. System updates the order record<br>8. The system provides status notification to the customer. |
| Exceptional Scenarios | 1. System unavailable: Staff records update offline and syncs later<br>2. Unresolvable issue: Staff reaches to the supervisor<br>3. Item damaged: the staff member starts the return procedure or determines necessary replacement requirements.<br>4. Delivery at wrong address: requires staff members to organize another delivery attempt. |

Use Case 5: Schedule Delivery

| Aspects | Description |
|---|---|
| Use Case | Schedule Delivery |
| Trigger | Order confirmed and ready for delivery scheduling |
| Precondition | 1. Order has been created and payment processed<br>2. Items are in stock and ready for shipment |
| Actors | Delivery Staff<br>Customer<br>System |
| Main scenarios | 1. The system produces a list containing new orders that need scheduling assignment.<br>2. Staff examines delivery details by reviewing the location information as well as size parameters and priority levels.<br>3. The staff conducts checks for delivery slots and available resources.<br>4. Staff members assign delivery dates and times together with courier services.<br>5. The system generates an optimized delivery plan that includes the best route.<br>6. The system successfully switches order status to read "Scheduled."<br>7. The system automatically sends delivery schedule information to customers regarding their delivery date together with time. |
| Exceptional Scenarios | 1. No suitable delivery slots: Staff works with customers to find replacement delivery date<br>2. Resource shortage: Staff members provide priority service to deliveries while shifting lower-priority orders to different times.<br>3. Remote location unreachable: Staff arranges special delivery method<br>4. Last-minute cancellation: Staff reschedules resources for other deliveries. |

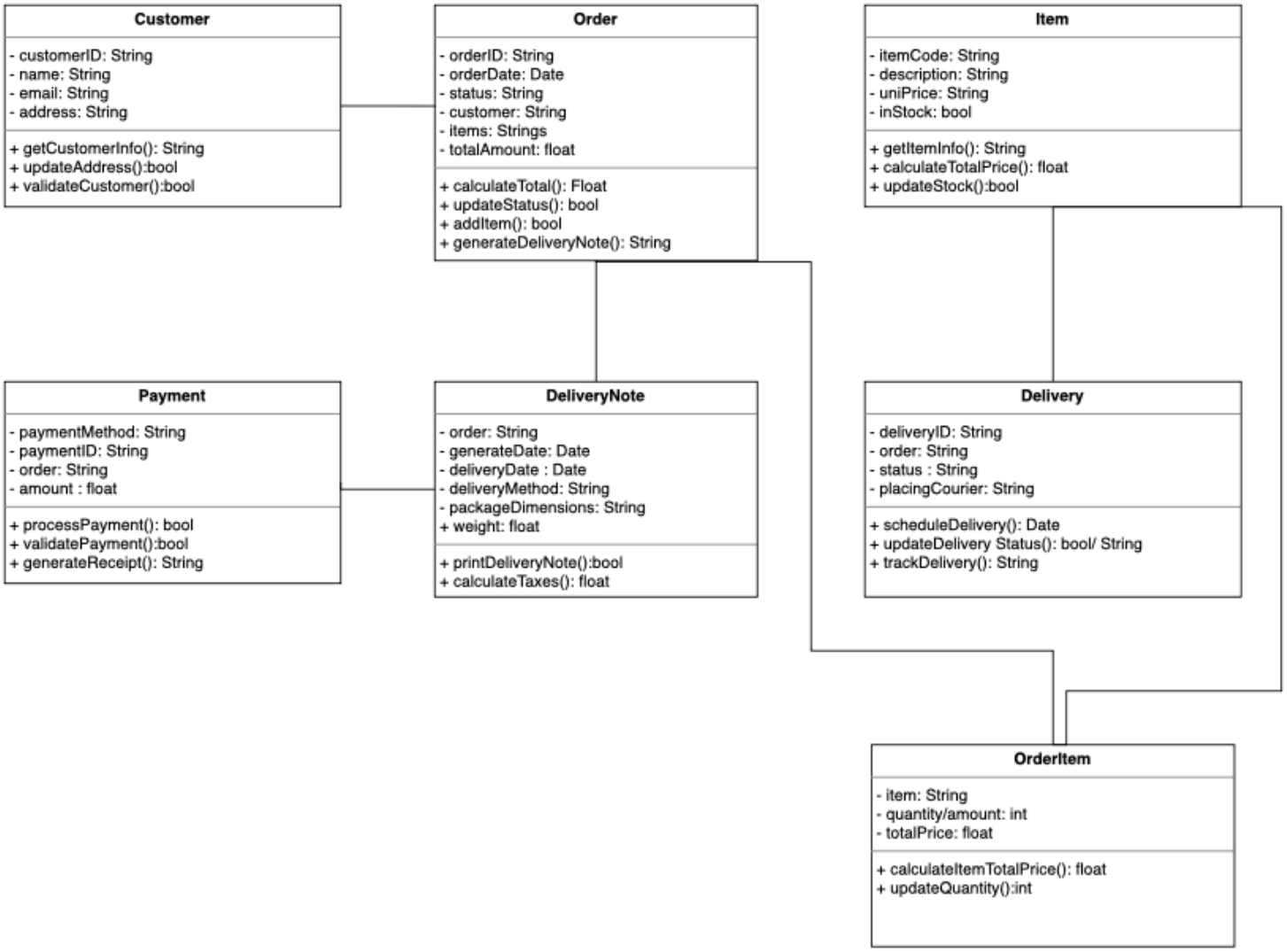
2. **Identify Objects and Classes:**

a. Based on the use-case descriptions those are the main classes:

1. Customer - Represents the recipient of the delivery
2. Order - Represents the delivery order with its details
3. Item - Represents products being delivered
4. DeliveryNote - Represents the delivery receipt document
5. Payment - Handles payment processing information
6. Delivery - Manages delivery scheduling and tracking information
7. OrderItem – calculates the total prices and the amount of available stocks

b. UML use-case diagram

Link: [Assignment 1, task 2 .drawio-2.htmlh]



**c. The supporting descriptions for the classes and the access specifiers for member visibility:**

| Class | Attributes (Private) - | Methods (Public) + |
|---|---|---|
| Customer | - customerID: Unique identifier for each customer<br>- name: Full name of the customer<br>- email: Email address for contact<br>- address: Delivery address | - getCustomerInfo(): Returns all customer information<br>- updateAddress(): Updates customer's address<br>- validateCustomer(): Validates customer information |
| Order | - orderID: Unique identifier for each order (DEL123456789)<br>- orderDate: Date when the order was placed<br>- status: Current status of the order<br>- customer: Reference to the Customer object<br>- items: List of OrderItem objects<br>- totalAmount: Total order amount including taxes | - calculateTotal(): Calculates the total order amount<br>- updateStatus(): Updates the order status<br>- addItem(): Adds an item to the order<br>- generateDeliveryNote(): Creates a delivery note |
| Item | - itemCode: Unique identifier for each item (ITM001)<br>- description: Description of the item<br>- unitPrice: Price per unit of the item<br>- inStock: Availability status | - getItemInfo(): Returns all item information<br>- calculateTotalPrice(): Calculates price based on quantity<br>- updateStock(): Updates inventory when items are ordered |
| Delivery Note | - order: Associated Order object<br>- generationDate: Date when note was generated<br>- deliveryDate: Scheduled delivery date<br>- deliveryMethod: Method of delivery (Courier)<br>- packageDimensions: Size information of package<br>- weight: Weight of package in kg | - printDeliveryNote(): Prints the delivery note<br>- calculateTaxes(): Calculates applicable taxes |
| Payment | - paymentMethod: Method of payment<br>- paymentID: Unique identifier for payment<br>- order: Associated Order object<br>- amount: Payment amount | - processPayment(): Processes the payment<br>- validatePayment(): Validates payment information<br>- generateReceipt(): Generates payment receipt |

| | | |
|---|---|---|
| Delivery | **- deliveryID: Unique identifier for delivery**<br>**- order: Associated Order object**<br>**- status: Current delivery status**<br>**- placingCourier: Courier assigned for delivery** | **- scheduleDelivery(): Schedules delivery date and time**<br>**- updateDeliveryStatus(): Updates delivery status**<br>**- trackDelivery(): Provides tracking information** |
| OrderItem | - item: Reference to an Item object<br>- quantity: Number of items ordered<br>- totalPrice: Total price for the item quantity | - calculateItemTotalPrice(): Calculates total price<br>- updateQuantity(): Updates item quantity |

3. **Create Python Classes and Objects:**

For this code part, for each class I conduct with exactly 5 attributes, I have getter methods for all attributes, also, setter methods for mutable attributes, the required function headers with pass statements with their comments to indicate what the function should achieve. Finally, I have maintained the necessary Enum classes for statuses and methods.

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

From this step, I've learned how to create objects from different classes like Customer, Item, OrderItem, Order, and DeliveryNote, giving them specific attributes to represent real-life things. discovering how to build connections between these objects by passing them as parameters to other objects' constructors, forming a linked structure. Moreover, accessing data from these objects while using getter methods and how to use setter methods to change values. This helps to calculate totals, like total weight and prices. Finally, I've learned how to create a function that uses these connected objects to generate a well-organized delivery note with clear spacing.

**Link (in case) for both part 3 and 4:** https://www.programiz.com/online-compiler/3eYGvnA248Usu

I have merged the **PDFs**, please scroll all the way down for the code.

**Summary of learnings:**

This assignment was a real learning experience for me. At first, I felt overwhelmed trying to identify all the use-cases, but once I started thinking about how a delivery system works, it became more manageable. Drawing the UML diagrams helped me visualize everything, and it made me feel more confident in how the system should be structured. Writing the Python code was a bit tricky at first, especially figuring out how to set up the classes with constructors and methods, but it felt great when I finally got everything working. The most rewarding part was seeing the delivery note come together using all the pieces I built. Overall, I feel like I've learned a lot about system design, UML, and coding in Python, and I'm definitely more comfortable with object-oriented programming now.

The PDF code for both **part 3 and 4:**

In [3]:
```python
from enum import Enum
from datetime import datetime
from typing import List

# Enum representing various order statuses
class OrderStatus(Enum):
    CREATED = "Created"
    PROCESSING = "Processing"
    READY_FOR_DELIVERY = "Ready for Delivery"
    IN_TRANSIT = "In Transit"
    DELIVERED = "Delivered"
    CANCELLED = "Cancelled"

# Enum representing various delivery statuses
class DeliveryStatus(Enum):
    SCHEDULED = "Scheduled"
    PICKED_UP = "Picked Up"
    IN_TRANSIT = "In Transit"
    OUT_FOR_DELIVERY = "Out for Delivery"
    DELIVERED = "Delivered"
    DELAYED = "Delayed"
    FAILED = "Failed"

# Enum representing payment methods
class PaymentMethod(Enum):
    CREDIT_CARD = "Credit Card"
    DEBIT_CARD = "Debit Card"
    PAYPAL = "PayPal"
    BANK_TRANSFER = "Bank Transfer"
    CASH_ON_DELIVERY = "Cash on Delivery"

# Enum representing delivery methods
class DeliveryMethod(Enum):
    STANDARD = "Standard"
    EXPRESS = "Express"
    OVERNIGHT = "Overnight"
```

```
    SAME_DAY = "Same Day"
    PICKUP = "Pickup"

# Class representing a customer
class Customer:
    def __init__(self, customer_id, name, email, address, phone):
        self.__customer_id = customer_id  # Unique customer identifier
        self.__name = name  # Customer's name
        self.__email = email  # Customer's email address
        self.__address = address  # Customer's delivery address
        self.__phone = phone  # Customer's phone number

    # Getter methods
    def get_customer_id(self):
        return self.__customer_id  # Return customer ID

    def get_name(self):
        return self.__name  # Return customer name

    def get_email(self):
        return self.__email  # Return customer email

    def get_address(self):
        return self.__address  # Return customer address

    def get_phone(self):
        return self.__phone  # Return customer phone number

    # Setter methods
    def set_name(self, name):
        self.__name = name  # Update customer name

    def set_email(self, email):
        self.__email = email  # Update customer email

    def set_address(self, address):
        self.__address = address  # Update customer address
```

```
    def set_phone(self, phone):
        self.__phone = phone  # Update customer phone number

    def get_customer_info(self):
        # Return customer info - to be implemented
        pass

    def update_address(self, new_address):
        # Update customer address - to be implemented
        pass

    def validate_customer(self):
        # Validate customer information - to be implemented
        pass

# Class representing an item
class Item:
    def __init__(self, item_code, description, unit_price, in_stock, weight):
        self.__item_code = item_code  # Unique item identifier
        self.__description = description  # Description of the item
        self.__unit_price = unit_price  # Price per unit
        self.__in_stock = in_stock  # Availability status
        self.__weight = weight  # Weight of the item

    # Getter methods
    def get_item_code(self):
        return self.__item_code  # Return item code

    def get_description(self):
        return self.__description  # Return item description

    def get_unit_price(self):
        return self.__unit_price  # Return unit price

    def get_in_stock(self):
        return self.__in_stock  # Return stock status

    def get_weight(self):
```

```
        return self.__weight  # Return weight

    # Setter methods
    def set_description(self, description):
        self.__description = description  # Update item description

    def set_unit_price(self, unit_price):
        self.__unit_price = unit_price  # Update unit price

    def set_in_stock(self, in_stock):
        self.__in_stock = in_stock  # Update stock availability

    def set_weight(self, weight):
        self.__weight = weight  # Update item weight

    def get_item_info(self):
        # Return all item information - to be implemented
        pass

    def calculate_total_price(self, quantity):
        # Calculate total price for given quantity - to be implemented
        pass

    def update_stock(self, quantity_change):
        # Update inventory - to be implemented
        pass

    def is_available(self, quantity):
        # Check if requested quantity is available - to be implemented
        pass

# Class representing an order item
class OrderItem:
    def __init__(self, item, quantity, unit_price, discount, total_price):
        self.__item = item  # Reference to the item
        self.__quantity = quantity  # Quantity ordered
        self.__unit_price = unit_price  # Price per unit
        self.__discount = discount  # Discount applied
```

```
        self.__total_price = total_price  # Total price after discount

    # Getter methods
    def get_item(self):
        return self.__item  # Return the associated item

    def get_quantity(self):
        return self.__quantity  # Return quantity ordered

    def get_unit_price(self):
        return self.__unit_price  # Return unit price

    def get_discount(self):
        return self.__discount  # Return discount applied

    def get_total_price(self):
        return self.__total_price  # Return total price

    # Setter methods
    def set_quantity(self, quantity):
        self.__quantity = quantity  # Update quantity ordered

    def set_discount(self, discount):
        self.__discount = discount  # Update discount applied

    def set_total_price(self, total_price):
        self.__total_price = total_price  # Update total price

    def calculate_item_total_price(self):
        # Calculate total price with discount - to be implemented
        pass

    def update_quantity(self, quantity):
        # Update quantity and recalculate price - to be implemented
        pass

# Class representing an order
class Order:
```

```
    def __init__(self, order_id, order_date, status, customer, total_amount):
        self.__order_id = order_id  # Unique order identifier
        self.__order_date = order_date  # Date when order was placed
        self.__status = status  # Current order status
        self.__customer = customer  # Reference to the associated customer
        self.__total_amount = total_amount  # Total amount for the order
        self.__items = []  # List of items in the order

    # Getter methods
    def get_order_id(self):
        return self.__order_id  # Return order ID

    def get_order_date(self):
        return self.__order_date  # Return order date

    def get_status(self):
        return self.__status  # Return order status

    def get_customer(self):
        return self.__customer  # Return associated customer

    def get_total_amount(self):
        return self.__total_amount  # Return total order amount

    def get_items(self):
        return self.__items  # Return list of order items

    # Setter methods
    def set_status(self, status):
        self.__status = status  # Update order status

    def set_total_amount(self, total_amount):
        self.__total_amount = total_amount  # Update total order amount

    def calculate_total(self):
        # Calculate total order amount - to be implemented
        pass
```

```
    def update_status(self, status):
        # Update order status - to be implemented
        pass

    def add_item(self, item, quantity, discount=0.0):
        # Add item to order - to be implemented
        pass

    def remove_item(self, item_code):
        # Remove item from order - to be implemented
        pass

    def generate_delivery_note(self):
        # Create delivery note - to be implemented
        pass

# Class representing a payment
class Payment:
    def __init__(self, payment_id, order, amount, payment_method, status):
        self.__payment_id = payment_id  # Unique payment identifier
        self.__order = order  # Reference to the associated order
        self.__amount = amount  # Payment amount
        self.__payment_method = payment_method  # Method of payment
        self.__status = status  # Current payment status

    # Getter methods
    def get_payment_id(self):
        return self.__payment_id  # Return payment ID

    def get_order(self):
        return self.__order  # Return associated order

    def get_amount(self):
        return self.__amount  # Return payment amount

    def get_payment_method(self):
        return self.__payment_method  # Return payment method
```

```
    def get_status(self):
        return self.__status  # Return payment status

    # Setter methods
    def set_amount(self, amount):
        self.__amount = amount  # Update payment amount

    def set_payment_method(self, payment_method):
        self.__payment_method = payment_method  # Update payment method

    def set_status(self, status):
        self.__status = status  # Update payment status

    def process_payment(self):
        # Process the payment - to be implemented
        pass

    def validate_payment(self):
        # Validate payment information - to be implemented
        pass

    def generate_receipt(self):
        # Generate payment receipt - to be implemented
        pass

# Class representing a delivery note
class DeliveryNote:
    def __init__(self, note_id, order, generation_date, delivery_date, delivery_method):
        self.__note_id = note_id  # Unique delivery note identifier
        self.__order = order  # Reference to the associated order
        self.__generation_date = generation_date  # Date note was generated
        self.__delivery_date = delivery_date  # Scheduled delivery date
        self.__delivery_method = delivery_method  # Method of delivery

    # Getter methods
    def get_note_id(self):
        return self.__note_id  # Return delivery note ID
```

```
    def get_order(self):
        return self.__order  # Return associated order

    def get_generation_date(self):
        return self.__generation_date  # Return generation date

    def get_delivery_date(self):
        return self.__delivery_date  # Return delivery date

    def get_delivery_method(self):
        return self.__delivery_method  # Return delivery method

    # Setter methods
    def set_delivery_date(self, delivery_date):
        self.__delivery_date = delivery_date  # Update delivery date

    def set_delivery_method(self, delivery_method):
        self.__delivery_method = delivery_method  # Update delivery method

    def print_delivery_note(self):
        # Format and print delivery note - to be implemented
        pass

    def calculate_taxes(self):
        # Calculate applicable taxes - to be implemented
        pass

    def email_delivery_note(self, email=None):
        # Email delivery note to customer - to be implemented
        pass

# Class representing a delivery
class Delivery:
    def __init__(self, delivery_id, order, status, tracking_number, courier):
        self.__delivery_id = delivery_id  # Unique delivery identifier
        self.__order = order  # Reference to the associated order
        self.__status = status  # Current delivery status
        self.__tracking_number = tracking_number  # Tracking number for delivery
```

```
        self.__courier = courier  # Courier assigned for delivery

    # Getter methods
    def get_delivery_id(self):
        return self.__delivery_id  # Return delivery ID

    def get_order(self):
        return self.__order  # Return associated order

    def get_status(self):
        return self.__status  # Return delivery status

    def get_tracking_number(self):
        return self.__tracking_number  # Return tracking number

    def get_courier(self):
        return self.__courier  # Return assigned courier

    # Setter methods
    def set_status(self, status):
        self.__status = status  # Update delivery status

    def set_courier(self, courier):
        self.__courier = courier  # Update assigned courier

    def schedule_delivery(self, date, time):
        # Schedule delivery date and time - to be implemented
        pass

    def update_delivery_status(self, status):
        # Update delivery status - to be implemented
        pass

    def track_delivery(self):
        # Get current tracking information - to be implemented
        pass

    def notify_customer(self, message):
```

```
        # Send notification to customer - to be implemented
        pass
```

In [10]:
```
# fourth part - using objects to generating a delivery note as the figure given

from enum import Enum
from datetime import datetime
from typing import List

# All the class definitions are from the provided code for part 3.

# Use objects to generate a Delivery Note:
if __name__ == "__main__":
    # Create a customer object
    customer = Customer(
        customer_id="CUST001",
        name="Sarah Johnson",
        email="sarah.johnson@example.com",
        address="45 Knowledge Avenue, Dubai, UAE",
        phone="Not provided"  # Phone wasn't in the figure instructions, but I add it optionally in the class
    )

    # Create item objects
    wireless_keyboard = Item(
        item_code="ITM001",
        description="Wireless Keyboard",
        unit_price=100.00,
        in_stock=10,  # Assuming the stock available
        weight=0.5  # And weight in kg
    )

    wireless_mouse = Item(
        item_code="ITM002",
        description="Wirless Mouse & Pad Set",
        unit_price=75.00,
        in_stock=15,
        weight=0.3
```

```
    )

    cooling_pad = Item(
        item_code="ITM003",
        description="Laptop Cooling Pad",
        unit_price=120.00,
        in_stock=8,
        weight=1.0
    )

    camera_lock = Item(
        item_code="ITM004",
        description="Camera Lock",
        unit_price=15.00,
        in_stock=20,
        weight=0.2
    )

    # Create order items
    keyboard_item = OrderItem(
        item=wireless_keyboard,
        quantity=1,
        unit_price=100.00,
        discount=0,
        total_price=100.00
    )

    mouse_item = OrderItem(
        item=wireless_mouse,
        quantity=1,
        unit_price=75.00,
        discount=0,
        total_price=75.00
    )

    cooling_pad_item = OrderItem(
        item=cooling_pad,
        quantity=1,
```

```
        unit_price=120.00,
        discount=0,
        total_price=120.00
    )

    camera_lock_item = OrderItem(
        item=camera_lock,
        quantity=3,
        unit_price=15.00,
        discount=0,
        total_price=45.00
    )

    # Create order
    order = Order(
        order_id="DEL123456789",  # The order ID from the figure
        order_date=datetime.strptime("2025-01-25", "%Y-%m-%d"),  # The delivery date
        status=OrderStatus.READY_FOR_DELIVERY,
        customer=customer,
        total_amount=283.50  # Total amount of the order
    )

    # Add items to order
    order._Order__items = [keyboard_item, mouse_item, cooling_pad_item, camera_lock_item]

    # Create delivery note
    delivery_note = DeliveryNote(
        note_id="DN-2025-001",  # Reference Number
        order=order,
        generation_date=datetime.now(),
        delivery_date=datetime.strptime("2025-01-25", "%Y-%m-%d"),
        delivery_method=DeliveryMethod.STANDARD  # Assuming delivery method as "Courier"
    )

    # Create delivery
    delivery = Delivery(
        delivery_id="DEL123456789",  # Same as order ID
        order=order,
```

```
        status=DeliveryStatus.SCHEDULED,
        tracking_number="Not provided",  # Not in the figure but I added it in class so I have to mentione it her
        courier="Courier"  # As mentioned in the figure
    )

# Function to print the delivery note
    def generate_delivery_note(note):
        order = note.get_order() # Get the order from the note
        customer = order.get_customer()  # Get customer details
        items = order.get_items() # Get items in the order

        # Total weight as required
        total_weight = 7.0

        # Printing the header
        print("\n" + "="*60)
        print(f"{'Delivery Note':^60}")
        print("="*60)
        print("Thank you for using our delivery service! Please print your delivery receipt and present it")
        print("upon receiving your items.")
        print()

        # printing Recipient Details
        print("Recipient Details:")
        print(f"Name: {customer.get_name()}")
        print(f"Contact: {customer.get_email()}")
        print(f"Delivery Address: {customer.get_address()}")
        print("\n" + "-"*60)

        # Printing Delivery Information
        print("Delivery Information:")
        print(f"Order Number: {order.get_order_id()}")
        print(f"Reference Number: {note.get_note_id()}")
        print(f"Delivery Date: {note.get_delivery_date().strftime('%B %d, %Y')}")
        print(f"Delivery Method: {delivery.get_courier()}")
        print(f"Package Dimensions: Not specified")
        print(f"Total Weight: {total_weight} kg")
        print("\n" + "-"*60)
```

```
        # Summary of Items Delivered
        print("Summary of Items Delivered:")
        print(f"{'Item Code':<10} {'Description':<25} {'Quantity':<10} {'Unit Price (AED)':<20} {'Total Price (AE

        # Using the fixed subtotal of 270.00 as shown
        subtotal = 270.00

        for item in items:
            product = item.get_item() # Get product details
            quantity = item.get_quantity() # Get quantity
            unit_price = item.get_unit_price() # Get unit price
            total_price = item.get_total_price() # Get total price

            print(f"{product.get_item_code():<10} {product.get_description():<25} {quantity:<10} {unit_price:<20.

        # Use fixed values for calculations
        taxes = 13.50
        total = 283.50

        print("\n")
        print(f"Subtotal: AED {subtotal:.2f}")
        print(f"Taxes and Fees: AED {taxes:.2f}")
        print(f"Total Charges: AED {total:.2f}")
        print("="*60)

    # Generate and display the delivery note
    generate_delivery_note(delivery_note)
```

```
============================================================
                      Delivery Note
============================================================
Thank you for using our delivery service! Please print your delivery receipt and present it
upon receiving your items.

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

------------------------------------------------------------
Delivery Information:
Order Number: DEL123456789
Reference Number: DN-2025-001
Delivery Date: January 25, 2025
Delivery Method: Courier
Package Dimensions: Not specified
Total Weight: 7.0 kg

------------------------------------------------------------
Summary of Items Delivered:
Item Code  Description               Quantity   Unit Price (AED)     Total Price (AED)
ITM001     Wireless Keyboard         1          100.00               100.00
ITM002     Wirless Mouse & Pad Set   1          75.00                75.00
ITM003     Laptop Cooling Pad        1          120.00               120.00
ITM004     Camera Lock               3          15.00                45.00


Subtotal: AED 270.00
Taxes and Fees: AED 13.50
Total Charges: AED 283.50
============================================================
```

In [ ]: