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 [ ]: