Task 4

Code:

```
from datetime import date  # Import date module to handle delivery dates
from enum import Enum  # Import Enum for defining delivery statuses

# Define an Enum class for Delivery Status
class DeliveryStatus(Enum):
    PENDING = "Pending"
    DISPATCHED = "Dispatched"
    DELIVERED = "Delivered"
    CANCELLED = "Cancelled"

# Class to store recipient details
class Recipient:
    def __init__(self, name, contact, address):
        self.name = name  # Customer name
        self.contact = contact  # Customer contact email/phone
        self.address = address  # Delivery address

    def get_details(self):
        """Returns recipient details as a formatted string"""
```

```
        return f"{self.name}, {self.contact}, {self.address}"


# Class to represent an item in the order
class Item:
    def __init__(self, item_code, description, quantity, unit_price):
        self.item_code = item_code  # Unique code for the item
        self.description = description  # Item name/description
        self.quantity = quantity  # Number of items ordered
        self.unit_price = unit_price  # Price per unit
        self.total_price = self.calculate_total_price()  # Total price of item


    def calculate_total_price(self):
        """Calculates the total price of the item"""
        return self.quantity * self.unit_price


# Class to manage the order details
class Order:
    def __init__(self, order_number, order_date, recipient, items, total_weight, delivery_method):
        self.order_number = order_number  # Unique order number
        self.order_date = order_date  # Order date
        self.recipient = recipient  # Recipient object
        self.items = items  # List of ordered items
        self.total_weight = total_weight  # Weight of the package
```

```
        self.delivery_method = delivery_method  # Delivery type (e.g., Courier)
        self.delivery_status = DeliveryStatus.PENDING  # Initial status is pending

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

# Class to handle charge calculations
class ChargeCalculator:

    def calculate_subtotal(items):
        """Calculates subtotal by summing up the total price of all items"""
        return sum(item.total_price for item in items)


    def calculate_taxes(subtotal):
        """Calculates 5% tax on the subtotal"""
        return round(subtotal * 0.05, 2)  # Assuming 5% VAT


    def calculate_total(subtotal, taxes):
        """Calculates total charges including taxes"""
        return round(subtotal + taxes, 2)
```

```
# Class to generate the delivery note
class DeliveryNote:
    def __init__(self, reference_number, order):
        self.reference_number = reference_number  # Unique reference number
        self.order = order  # Associated order object
        self.subtotal = ChargeCalculator.calculate_subtotal(order.items)  # Subtotal calculation
        self.taxes_and_fees = ChargeCalculator.calculate_taxes(self.subtotal)  # Taxes calculation
        self.total_charges = ChargeCalculator.calculate_total(self.subtotal, self.taxes_and_fees)  #
Total amount
        self.delivery_date = date(2025, 1, 25)  # Fixed delivery date for now

    def generate_note(self):
        """Generates a formatted delivery note"""
        note = f"""
        ---------------------------------------------
        DELIVERY NOTE
        Reference Number: {self.reference_number}
        Order Number: {self.order.order_number}
        Delivery Date: {self.delivery_date}
        Delivery Method: {self.order.delivery_method}

        Recipient Details:
```

```
        {self.order.recipient.get_details()}


        Items Delivered:

        ------------------------------------------

        Item Code | Description              | Qty | Unit Price (AED) | Total Price (AED)

        ----------------------------------------------------------------

        """

        # Loop through each item and add it to the delivery note

        for item in self.order.items:

            note += f"{item.item_code:9} | {item.description:25} | {item.quantity:3} |
{item.unit_price:15.2f} | {item.total_price:16.2f}\n"


        # Add financial details at the end of the note

        note += f"""

        ------------------------------------------

        Subtotal:          AED {self.subtotal:.2f}

        Taxes and Fees:    AED {self.taxes_and_fees:.2f}

        Total Charges:     AED {self.total_charges:.2f}

        ------------------------------------------

        """

        return note  # Return the formatted delivery note


# Class to simulate sending the delivery note via email
```

```
class EmailService:

    def send_delivery_note(recipient_email, delivery_note):
        """Simulates sending the delivery note to the recipient via email"""
        print(f" Sending delivery note to {recipient_email}...\n")
        print(delivery_note)
        return True  # Simulating successful email delivery

# ------------------ CREATE OBJECTS & GENERATE DELIVERY NOTE ------------------

# Create recipient object
recipient = Recipient("Sarah Johnson", "sarah.johnson@example.com", "45 Knowledge Avenue,
Dubai, UAE")

# Create item objects for each ordered product
items = [
    Item("ITM001", "Wireless Keyboard", 1, 100.00),
    Item("ITM002", "Wireless Mouse & Pad Set", 1, 75.00),
    Item("ITM003", "Laptop Cooling Pad", 1, 120.00),
    Item("ITM004", "Camera Lock", 3, 15.00)
]

# Create an order object
```

```
order = Order(order_number="DEL123456789", order_date=date.today(), recipient=recipient,
items=items, total_weight=7.0, delivery_method="Courier")

# Create a delivery note object
delivery_note = DeliveryNote(reference_number="DN-2025-001", order=order)

# Print the generated delivery note
print(delivery_note.generate_note())

# Optionally, simulate sending the delivery note via email
EmailService.send_delivery_note(recipient.contact, delivery_note.generate_note())
```

Output:

```
--------------------------------------------
DELIVERY NOTE
Reference Number: DN-2025-001
Order Number: DEL123456789
Delivery Date: 2025-01-25
Delivery Method: Courier

Recipient Details:
```

Sarah Johnson, sarah.johnson@example.com, 45 Knowledge Avenue, Dubai, UAE

Items Delivered:

Item Code | Description | Qty | Unit Price (AED) | Total Price (AED)

ITM001 | Wireless Keyboard | 1 | 100.00 | 100.00

ITM002 | Wireless 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

📩 Sending delivery note to sarah.johnson@example.com...