**Delivery Management System**

Student Name   
University Affiliation

Course

Professor

Date

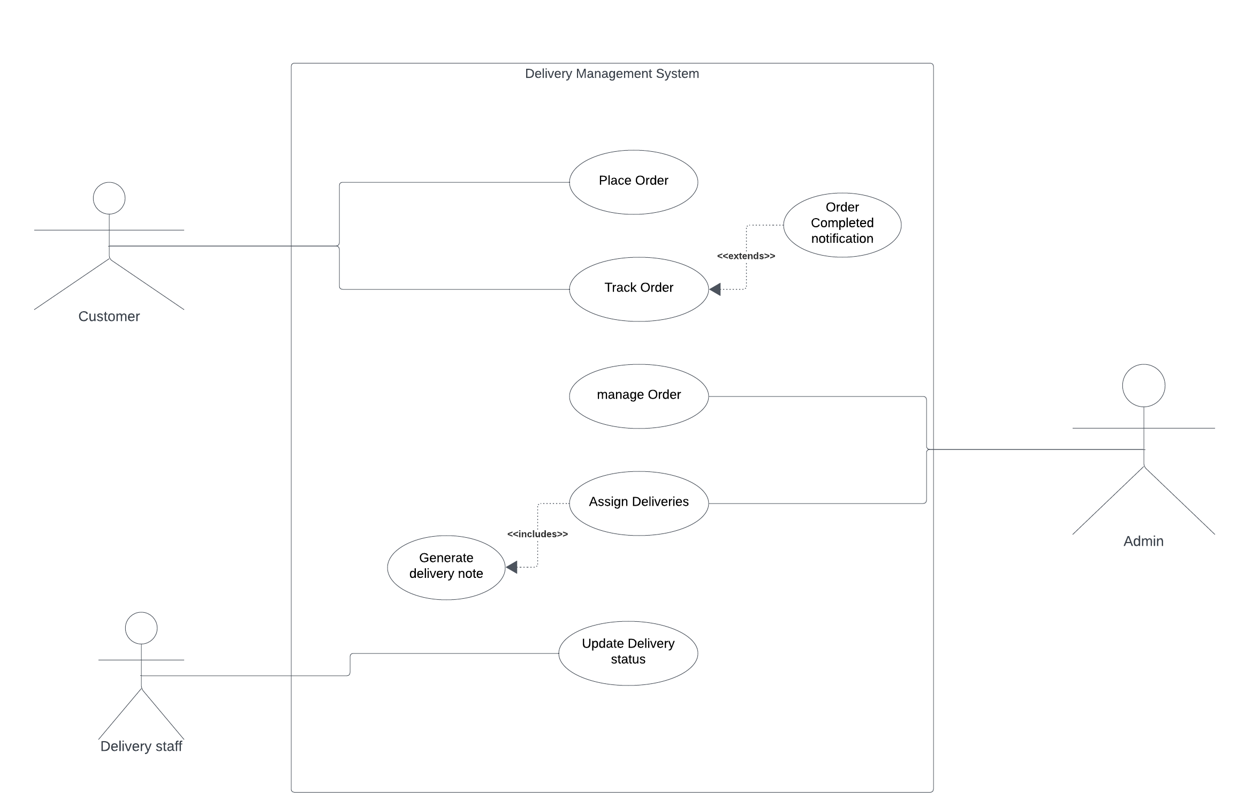
**UML Use- Cases**

1. **Use cases for the delivery management system**

The use cases, according to the provided information, include:

1. Place a delivery order
2. Generate a delivery note
3. Update delivery status
4. Track delivery
5. Manage delivery details
6. Calculate charges
7. **Below is the use case diagram**

The use case diagram



Link: <https://lucid.app/lucidchart/1cb50fda-eed8-4c99-b194-facc6ab1ff87/edit?beaconFlowId=147004213CBEBB35&invitationId=inv_f53b48db-f8bb-4202-ac76-3b1547919c0e&page=0_0#>

Use case description

1. Place order

Actor: Customer

Description: The customer orders by selecting the item, entering the delivery information, and confirming the purchase.

Preconditions: Customer must be logged in to the system.

Postconditions: The order confirmation message is generated.

1. Track Order

Actor: Customer

Description: The Customer checks the current status of the order.

Preconditions: Order must be placed.

Postconditions: The system displays the order status.

1. Manage orders

Actor: Admin

Description: The admin can update, review, and manage the orders.

Preconditions: At least one order must exist.

Postconditions: Order details can be processed or updated further.

1. Assign delivery

Actor: Admin

Description: The admin assigns the order to a delivery staff.

Preconditions: The order must be ready for delivery.

Postconditions: Delivery personnel are assigned and notified.

1. Generate delivery note (included in assign delivery)

Actor: System

Description: The system generates the delivery note after an order has been assigned.

Preconditions: The order is supposed to have been assigned.

Postconditions: The delivery note is generated and stored.

1. Update delivery status

Actor: Delivery staff

Description: The delivery staff updates the order status, such as delivered.

Preconditions: The order must be assigned for delivery.

Postconditions: The order status is updated, and the Customer can view it.

1. Order completed notification (extends track order)

Actor: System

Description: The system sends a notification to the Customer when the order is delivered.

Preconditions: The order must be marked as delivered.

Postconditions: The Customer is notified.

1. **Scenarios**
2. Scenario 1: Placing an order (Customer)

Flow of events

* The Customer logs into the system and selects the option to place an order.
* The system prompts the Customer to enter delivery details such as the recipient's name, address, and contact.
* The system verifies the details <<include>> verify customer information.
* The Customer selects the items and confirms the order.
* The system generates the order number and displays a confirmation message.

1. Scenario 2: Assigning a delivery – Admin

Flow of events

* The admin logs into the system and selects the option to manage orders.
* Review the pending orders and assign the delivery staff.
* The system automatically generates the delivery note. <<include>> generate delivery note.
* The system notifies the delivery staff about the assigned order.

1. Scenario 3: Tracking an order (Customer and delivery staff)

Flow of events

* The Customer logs into the system and clicks on track order.
* The system collects the current order status and displays it.
* The delivery staff updates the order status. <<include>> update delivery status.
* The system reflects the Customer's updated status in real-time.
* If the status is marked as delivered, the system sends a confirmation message to the Customer. <<extend>> order completed notification.

**UML Class diagram**

1. **Objects and classes**
2. Customer class

Attributes: CustomerID, name, email, phone, and address.

1. Order class

Attributes: OrderID, customerID, items, total\_price, order\_date

1. Delivery Class

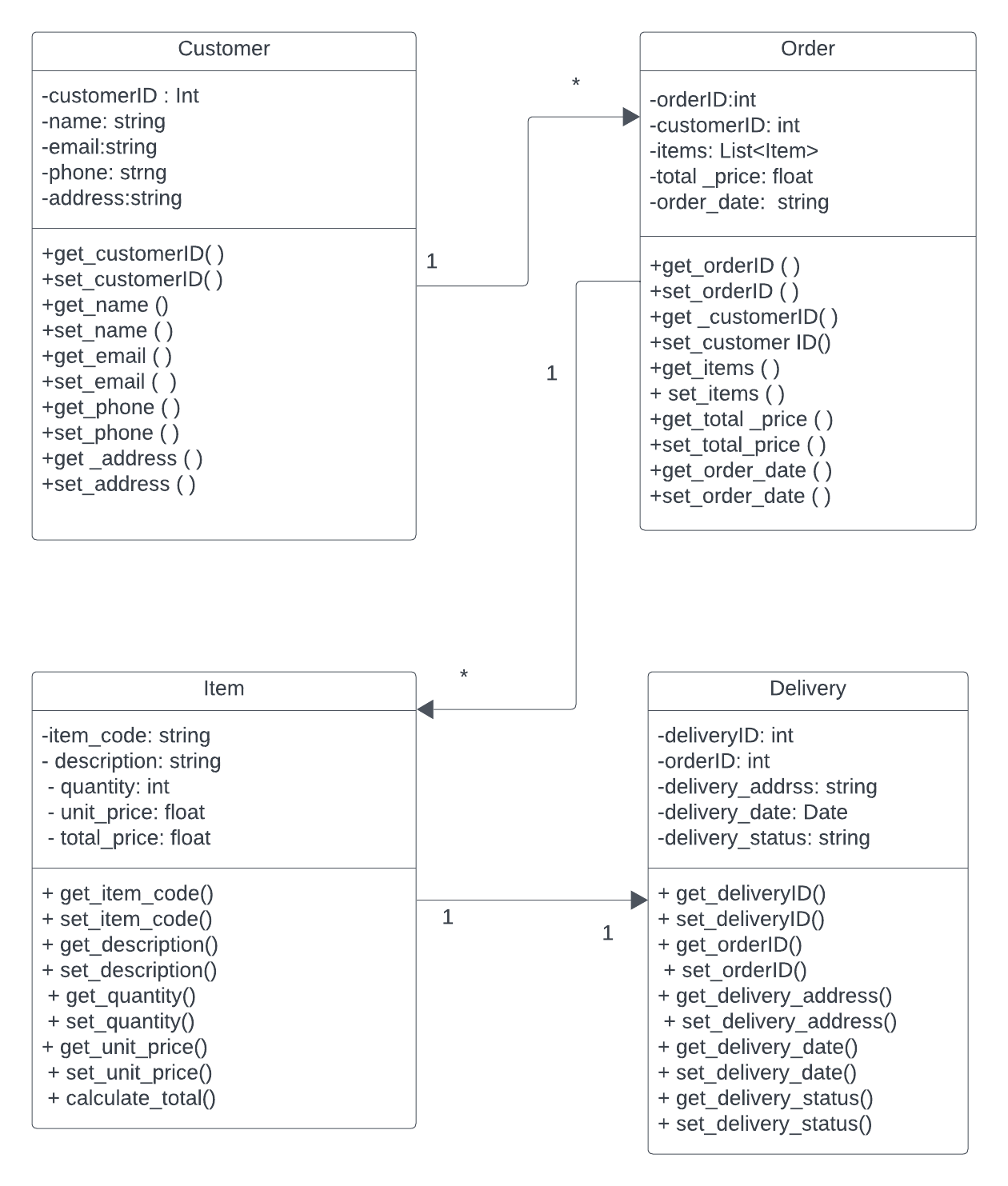
Attributes: DeliveryID, OrderID, delivery\_address, delivery\_date, delivery\_status

1. Item Class

Attributes: item\_code, description, quantity, unit\_price, total\_price

1. **UML class diagram and description**

Below is the class diagram



Link: <https://lucid.app/lucidchart/1cb50fda-eed8-4c99-b194-facc6ab1ff87/edit?beaconFlowId=147004213CBEBB35&invitationId=inv_f53b48db-f8bb-4202-ac76-3b1547919c0e&page=0_0#>

**Description**

1. Customer class

It represents the Customers that are placing the orders. The getters and setters methods for all attributes are provided. It has a one-to-many relationship with the order.

1. Order class

It shows the order placed by the Customer. Getters and setters methods for managing the items are included, such as add\_item ( ), remove\_item, and calculate\_total (). It has a one-to-one relationship with the item and one a one-to-one relationship with delivery.

1. Item

It represents the items in an order. For the getters and setters, calculate\_total () is used to compute the total price of an item.

1. Delivery Class

It represents the delivery information of a given order. The getters and setters for all attributes are included. It has one one-to-one relationship with the order.

**Python**

"""

This script models a simple delivery management system with classes for customers, deliveries, items, and invoices.

It allows tracking customer details, delivery information, items purchased, and final billing.

"""

class Customer:

"""Represents a customer placing an order, including name, contact details, and delivery address."""

def \_\_init\_\_(self, name, contact, address):

self.\_name = name

self.\_contact = contact

self.\_address = address

def get\_name(self):

return self.\_name

def set\_name(self, name):

self.\_name = name

def get\_contact(self):

return self.\_contact

def set\_contact(self, contact):

self.\_contact = contact

def get\_address(self):

return self.\_address

def set\_address(self, address):

self.\_address = address

def get\_details(self):

"""Returns customer details as a dictionary."""

return {

"Name": self.\_name,

"Contact": self.\_contact,

"Delivery Address": self.\_address

}

class Delivery:

"""Stores delivery details including order number, method, and delivery date."""

def \_\_init\_\_(self, order\_number, reference\_number, delivery\_date, method, dimensions, weight):

self.\_order\_number = order\_number

self.\_reference\_number = reference\_number

self.\_delivery\_date = delivery\_date

self.\_method = method

self.\_dimensions = dimensions

self.\_weight = weight

def get\_order\_number(self):

return self.\_order\_number

def set\_order\_number(self, order\_number):

self.\_order\_number = order\_number

def get\_reference\_number(self):

return self.\_reference\_number

def set\_reference\_number(self, reference\_number):

self.\_reference\_number = reference\_number

def get\_delivery\_date(self):

return self.\_delivery\_date

def set\_delivery\_date(self, delivery\_date):

self.\_delivery\_date = delivery\_date

def get\_method(self):

return self.\_method

def set\_method(self, method):

self.\_method = method

def get\_dimensions(self):

return self.\_dimensions

def set\_dimensions(self, dimensions):

self.\_dimensions = dimensions

def get\_weight(self):

return self.\_weight

def set\_weight(self, weight):

self.\_weight = weight

def get\_delivery\_details(self):

"""Returns delivery details as a dictionary."""

return {

"Order Number": self.\_order\_number,

"Reference Number": self.\_reference\_number,

"Delivery Date": self.\_delivery\_date,

"Delivery Method": self.\_method,

"Package Dimensions": self.\_dimensions,

"Total Weight": self.\_weight

}

class Item:

"""Represents an item in the delivery, including its price and quantity."""

def \_\_init\_\_(self, code, description, quantity, unit\_price):

self.\_code = code

self.\_description = description

self.\_quantity = quantity

self.\_unit\_price = unit\_price

self.\_total\_price = self.calculate\_total()

def get\_code(self):

return self.\_code

def set\_code(self, code):

self.\_code = code

def get\_description(self):

return self.\_description

def set\_description(self, description):

self.\_description = description

def get\_quantity(self):

return self.\_quantity

def set\_quantity(self, quantity):

self.\_quantity = quantity

self.\_total\_price = self.calculate\_total()

def get\_unit\_price(self):

return self.\_unit\_price

def set\_unit\_price(self, unit\_price):

self.\_unit\_price = unit\_price

self.\_total\_price = self.calculate\_total()

def get\_total\_price(self):

return self.\_total\_price

def calculate\_total(self):

"""Calculates the total price for the item."""

return self.\_quantity \* self.\_unit\_price

def get\_details(self):

"""Returns item details as a dictionary."""

return {

"Item Code": self.\_code,

"Description": self.\_description,

"Quantity": self.\_quantity,

"Unit Price (AED)": self.\_unit\_price,

"Total Price (AED)": self.\_total\_price

}

class Invoice:

"""Represents the final invoice summarizing all purchased items and total costs."""

def \_\_init\_\_(self, items, taxes\_and\_fees):

self.\_items = items

self.\_taxes\_and\_fees = taxes\_and\_fees

self.\_subtotal = sum(item.get\_total\_price() for item in items)

self.\_total\_charges = self.\_subtotal + self.\_taxes\_and\_fees

def get\_invoice\_details(self):

"""Returns invoice details as a dictionary."""

return {

"Items": [item.get\_details() for item in self.\_items],

"Subtotal (AED)": self.\_subtotal,

"Taxes and Fees (AED)": self.\_taxes\_and\_fees,

"Total Charges (AED)": self.\_total\_charges

}

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

"""Test execution demonstrating the use of the defined classes."""

customer = Customer("Sarah Johnson", "sarah.johnson@example.com", "45 Knowledge Avenue, Dubai, UAE")

delivery = Delivery("DEL123456789", "DN-2025-001", "January 25, 2025", "Courier", "", "7 kg")

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)

]

invoice = Invoice(items, 13.50)

print("Delivery Note")

print("Recipient Details:")

print(customer.get\_details())

print("\nDelivery Information:")

print(delivery.get\_delivery\_details())

print("\nSummary of Items Delivered:")

print(invoice.get\_invoice\_details())

**Reflection**

The process of working on the Delivery Management System assignment brought into clarity the concepts of system design and implementation. By creating UML diagrams specifically for use-case diagrams and class diagrams, I learned to view the relationships that exist between system elements, including customers, admins, delivery staff, and the whole system. This exercise confirmed how fundamental it is to establish preconditions with their corresponding postconditions and flow of events for every use case so systems can be structured. The development of the class diagram helped me better understand object-oriented design concepts, especially class associations, and data encapsulation, which establish effective data organization.

Testing the implementation using Python proved useful for the practical application of these principles. The class structure included Customer alongside Order and Delivery and Item with getter/setter methods for ensuring clean data handling and modularity. Implementing this work enhanced my capacity to process tasks related to order tracking, status updates, and delivery note generation in genuine operational situations. Implementing dictionaries and automated calculations for invoice system operations developed my ability to solve problems effectively. This project allowed me to experience different software development aspects firsthand while modeling systems and writing programs that linked classroom understanding to business practice.