**Delivery Management System**

Abdulrahman Bin Hulais  
Zayed University

ICS220

Professor Afshan Parkar

28/02/2025

**Introduction**

The Delivery Management System features automation for order placement workflow, order management, and tracking procedures. To benefit administrators and users, the integrated system contains components that manage the ordering process, status information, delivery allocation, and fee calculation. The solution defines system structure and essential entity relationships through UML diagrams such as use-case and class diagrams. The program's implementation with Python enables the execution of object-oriented principles that produce maintainable modular code with scalable features. This paper studies how the Delivery Management System optimizes order fulfillment and logistics operations through its design and actual usage.

**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

A diagram of a delivery management system

Description automatically generated

Link: <https://lucid.app/lucidchart/2ec89605-1b28-4c90-83c0-bde8309c8736/edit?viewport_loc=-570%2C12%2C2220%2C1232%2C0_0&invitationId=inv_1f18566c-cd29-4bdc-b6f1-15c2dadc2647>

Use case description

**Use case 1 : Login User**

|  |  |
| --- | --- |
| Ues case name | Login |
| Actor | Customer |
| Trigger | customer want to have order |
| Notes | Extend display loing error |
| Preconditions | Customer must be logged in to the system. |
| Main scenario |  |
| 1 | The Customer enters their username and password. |
| 2 | The system invokes Verify the Login. |
| 3 | The system grants access to the Customer. |
| **Exceptions:** | 2a. If the login fails, the system extends Display Login Error. |

**Ues case 2 : Manage Orders**

|  |  |
| --- | --- |
| Actors | Admin |
| Trigger | The Customer places an order. |
| Notes | Includes Verify Item and extends Display Unavailable Item. |
| Preconditions | The Customer must be logged in.  The Item must be available. |
| Main Scenarios:  1 | The Customer places an order. |
| 2 | The system invokes Verify Item to check availability. |
| 3 | If available, the order is processed successfully. |
| Exceptions | 2a. If the item is unavailable, the system extends Display Unavailable Item. |

**Ues case 3 : Purchase**

|  |  |
| --- | --- |
| Actors | Customer, Admin |
| Trigger | The Customer proceeds to purchase an item. |
| Notes | Includes Check Balance and extends Display Failed Payment. |
| Preconditions: | The Customer must have placed an order. |
| Main Scenarios:  1 | The Customer initiates a purchase. |
| 2 | The system invokes Check Balance to verify sufficient funds. |
| 3 | If funds are sufficient, the purchase is completed. |
| Exceptions: | 2a. If payment fails, the system extends Display Failed Payment. |

**Use Case 4: Assign Deliveries**

|  |  |
| --- | --- |
| Actors | Admin, Delivery Staff |
| Trigger | An order is ready for delivery. |
| Preconditions | The order must be successfully placed.  Delivery staff must be available. |
| **Main Scenarios:** | The system assigns an available delivery staff member. |
| 2 | The assigned delivery staff receives delivery details. |
| 3 | The delivery staff confirms and proceeds with the delivery. |
| **Exceptions:** | 1a. If no delivery staff is available, the system holds the assignment until one becomes available. |

**Scenarios**

**i. Scenario 1: Logging in (Customer)**

**Flow of events:**

* The Customer opens the system and selects the login option.
* The system prompts the Customer to enter their username and password.
* The system verifies the login credentials **<<include>> Verify the Login**.
* If the credentials are correct, the Customer is granted access to their account.
* The Customer is redirected to the main dashboard.
* If the credentials are incorrect, the system **<<extend>> Display Login Error** and prompts the Customer to retry.

**ii. Scenario 2: Managing Orders (Admin)**

**Flow of events:**

* The Admin logs into the system and selects the option to manage orders.
* The system displays a list of pending orders.
* The Admin selects an order and **<<include>> Verify Item** to check stock availability.
* If the item is available, the system processes the order and updates its status.
* If the item is unavailable, the system **<<extend>> Display Unavailable Item** and notifies the Customer.

**iii. Scenario 3: Purchasing an Item (Customer & Admin)**

**Flow of events:**

* The Customer selects an order and chooses the payment option.
* The system **<<include>> Check Balance** to verify available funds.
* If the payment is successful, the system updates the order status to "Purchased" and sends a confirmation message.
* If the payment fails, the system **<<extend>> Display Failed Payment** and prompts the Customer to retry with another payment method.

**iv. Scenario 4: Assigning Deliveries (Admin & Delivery Staff)**

**Flow of events:**

* The Admin logs into the system and selects the option to assign deliveries.
* The system displays a list of confirmed orders awaiting delivery.
* The Admin assigns an available Delivery Staff to the order.
* The system updates the order status and notifies the assigned Delivery Staff.
* The Delivery Staff confirms the assignment and prepares for delivery.
* If no Delivery Staff is available, the system holds the assignment until one becomes available.

**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

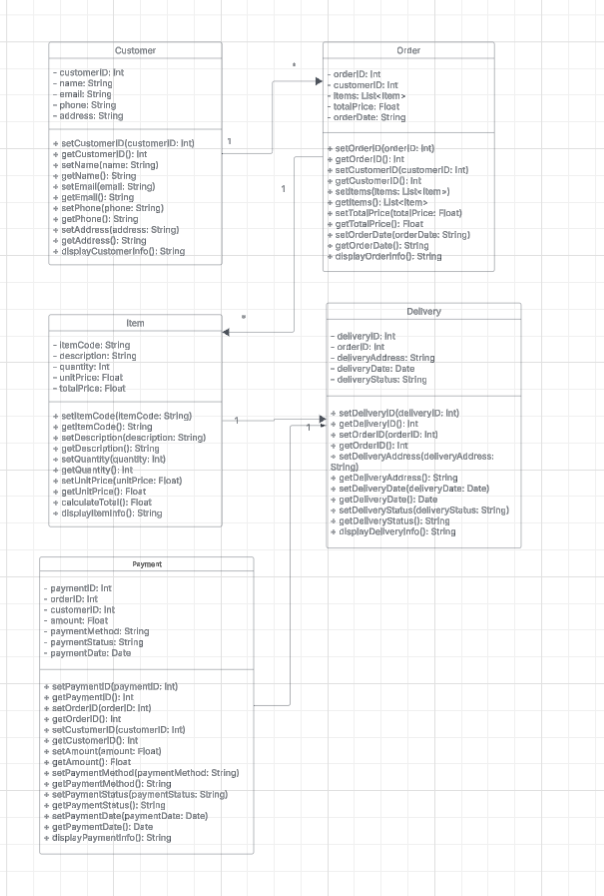
Attributes: DeliveryID, OrderID, delivery\_address, delivery\_date, delivery\_status

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/2ec89605-1b28-4c90-83c0-bde8309c8736/edit?viewport_loc=-797%2C-799%2C7842%2C4062%2C0_0&invitationId=inv_1f18566c-cd29-4bdc-b6f1-15c2dadc2647>

**Description**

1. Customer Class

The Customer class represents users who place orders in the delivery management system. It contains personal information such as their name, email, phone number, and address. It also provides methods for retrieving and updating this information.

Attributes:

customerID: A unique identifier for the customer.

name: The full name of the customer.

email: The email address for communication.

phone: The contact number of the customer.

address: The customer's delivery address.

Methods:

setCustomerID(customerID: Int) / getCustomerID(): Sets and retrieves the customer ID.

setName(name: String) / getName(): Sets and retrieves the customer’s name.

setEmail(email: String) / getEmail(): Sets and retrieves the email.

setPhone(phone: String) / getPhone(): Sets and retrieves the phone number.

setAddress(address: String) / getAddress(): Sets and retrieves the delivery address.

displayCustomerInfo(): Displays all customer details.

2. Order Class

The Order class represents a purchase made by a customer. It contains details about the items purchased, the total price, and the order date. Orders are linked to customers.

Attributes:

orderID: A unique identifier for each order.

customerID: The ID of the customer who placed the order.

items: A list of items included in the order.

totalPrice: The total cost of all items in the order.

orderDate: The date when the order was placed.

Methods:

setOrderID(orderID: Int) / getOrderID(): Sets and retrieves the order ID.

setCustomerID(customerID: Int) / getCustomerID(): Sets and retrieves the customer ID linked to the order.

setItems(items: List<Item>) / getItems(): Adds or retrieves the items in the order.

setTotalPrice(totalPrice: Float) / getTotalPrice(): Sets and retrieves the total cost.

setOrderDate(orderDate: String) / getOrderDate(): Sets and retrieves the order date.

displayOrderInfo(): Displays all order details.

3. Item Class

The Item class represents products available for purchase in the system. It stores item details like description, quantity, unit price, and total price.

Attributes:

itemCode: A unique identifier for the item.

description: A short description of the item.

quantity: The number of units ordered.

unitPrice: The price per unit.

totalPrice: The total cost (calculated as quantity \* unitPrice).

Methods:

setItemCode(itemCode: String) / getItemCode(): Sets and retrieves the item code.

setDescription(description: String) / getDescription(): Sets and retrieves the item description.

setQuantity(quantity: Int) / getQuantity(): Sets and retrieves the quantity.

setUnitPrice(unitPrice: Float) / getUnitPrice(): Sets and retrieves the unit price.

calculateTotal(): Computes the total price (quantity \* unitPrice).

displayItemInfo(): Displays all item details.

4. Delivery Class

The Delivery class represents the shipping details of an order. It keeps track of which order is being delivered, where it is being sent, and its current status.

Attributes:

deliveryID: A unique identifier for the delivery.

orderID: The order associated with this delivery.

deliveryAddress: The shipping address where the order is to be delivered.

deliveryDate: The date the order is scheduled for delivery.

deliveryStatus: The current status of the delivery (e.g., "Pending," "Shipped," "Delivered").

Methods:

setDeliveryID(deliveryID: Int) / getDeliveryID(): Sets and retrieves the delivery ID.

setOrderID(orderID: Int) / getOrderID(): Links the delivery to an order.

setDeliveryAddress(deliveryAddress: String) / getDeliveryAddress(): Sets and retrieves the delivery address.

setDeliveryDate(deliveryDate: Date) / getDeliveryDate(): Sets and retrieves the delivery date.

setDeliveryStatus(deliveryStatus: String) / getDeliveryStatus(): Updates and retrieves the delivery status.

displayDeliveryInfo(): Displays all delivery details.

5. Payment Class

The Payment class handles financial transactions for the orders. It stores payment-related details such as the amount, method, status, and date.

Attributes:

paymentID: A unique identifier for the payment transaction.

orderID: The order linked to this payment.

customerID: The ID of the customer making the payment.

amount: The total amount paid.

paymentMethod: The method used for payment (e.g., "Credit Card," "PayPal").

paymentStatus: The current status of the payment (e.g., "Completed," "Failed").

paymentDate: The date the payment was made.

Methods:

setPaymentID(paymentID: Int) / getPaymentID(): Sets and retrieves the payment ID.

setOrderID(orderID: Int) / getOrderID(): Links the payment to an order.

setCustomerID(customerID: Int) / getCustomerID(): Associates the payment with a customer.

setAmount(amount: Float) / getAmount(): Sets and retrieves the payment amount.

setPaymentMethod(paymentMethod: String) / getPaymentMethod(): Sets and retrieves the payment method.

setPaymentStatus(paymentStatus: String) / getPaymentStatus(): Updates and retrieves the payment status.

setPaymentDate(paymentDate: Date) / getPaymentDate(): Sets and retrieves the payment date.

displayPaymentInfo(): Displays all payment details.

**Python**

class Customer:

def \_\_init\_\_(self, customerID, name, email, phone, address):

self.\_customerID = customerID

self.\_name = name

self.\_email = email

self.\_phone = phone

self.\_address = address

def get\_customerID(self): return self.\_customerID

def set\_customerID(self, customerID): self.\_customerID = customerID

def get\_name(self): return self.\_name

def set\_name(self, name): self.\_name = name

def get\_email(self): return self.\_email

def set\_email(self, email): self.\_email = email

def get\_phone(self): return self.\_phone

def set\_phone(self, phone): self.\_phone = phone

def get\_address(self): return self.\_address

def set\_address(self, address): self.\_address = address

def display(self):

print(f"Customer ID: {self.\_customerID}")

print(f"Name: {self.\_name}")

print(f"Email: {self.\_email}")

print(f"Phone: {self.\_phone}")

print(f"Address: {self.\_address}")

print("=" \* 30)

class Item:

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

self.\_item\_code = item\_code

self.\_description = description

self.\_quantity = quantity

self.\_unit\_price = unit\_price

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

def get\_item\_code(self): return self.\_item\_code

def set\_item\_code(self, item\_code): self.\_item\_code = item\_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() # Update total when quantity changes

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() # Update total when price changes

def get\_total\_price(self): return self.\_total\_price

def calculate\_total(self): return self.\_quantity \* self.\_unit\_price

def display(self):

print(f"Item Code: {self.\_item\_code}")

print(f"Description: {self.\_description}")

print(f"Quantity: {self.\_quantity}")

print(f"Unit Price: {self.\_unit\_price}")

print(f"Total Price: {self.\_total\_price}")

print("-" \* 30)

class Order:

def \_\_init\_\_(self, orderID, customer, items, order\_date):

self.\_orderID = orderID

self.\_customer = customer

self.\_items = items

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

self.\_order\_date = order\_date

def get\_orderID(self): return self.\_orderID

def set\_orderID(self, orderID): self.\_orderID = orderID

def get\_customer(self): return self.\_customer

def set\_customer(self, customer): self.\_customer = customer

def get\_items(self): return self.\_items

def set\_items(self, items): self.\_items = items

def get\_total\_price(self): return self.\_total\_price

def set\_total\_price(self, total\_price): self.\_total\_price = total\_price

def get\_order\_date(self): return self.\_order\_date

def set\_order\_date(self, order\_date): self.\_order\_date = order\_date

def display(self):

print(f"Order ID: {self.\_orderID}")

print(f"Order Date: {self.\_order\_date}")

print(f"Total Price: {self.\_total\_price}")

print("\nCustomer Details:")

self.\_customer.display()

print("\nItems Ordered:")

for item in self.\_items:

item.display()

class Delivery:

def \_\_init\_\_(self, deliveryID, order, delivery\_address, delivery\_date, delivery\_status):

self.\_deliveryID = deliveryID

self.\_order = order

self.\_delivery\_address = delivery\_address

self.\_delivery\_date = delivery\_date

self.\_delivery\_status = delivery\_status

def get\_deliveryID(self): return self.\_deliveryID

def set\_deliveryID(self, deliveryID): self.\_deliveryID = deliveryID

def get\_order(self): return self.\_order

def set\_order(self, order): self.\_order = order

def get\_delivery\_address(self): return self.\_delivery\_address

def set\_delivery\_address(self, delivery\_address): self.\_delivery\_address = delivery\_address

def get\_delivery\_date(self): return self.\_delivery\_date

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

def get\_delivery\_status(self): return self.\_delivery\_status

def set\_delivery\_status(self, delivery\_status): self.\_delivery\_status = delivery\_status

def display(self):

print(f"Delivery ID: {self.\_deliveryID}")

print(f"Delivery Address: {self.\_delivery\_address}")

print(f"Delivery Date: {self.\_delivery\_date}")

print(f"Delivery Status: {self.\_delivery\_status}")

print("\nOrder Details:")

self.\_order.display()

# Creating Objects and Printing Details

customer = Customer(1, "John Doe", "john.doe@example.com", "1234567890", "123 Street, NY")

items = [

Item("ITM001", "Laptop", 1, 1500),

Item("ITM002", "Mouse", 2, 50)

]

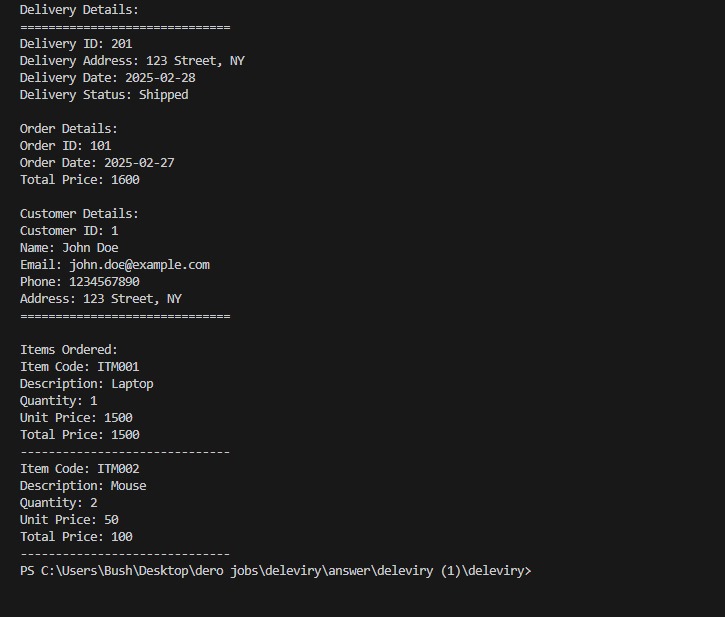
order = Order(101, customer, items, "2025-02-27")

delivery = Delivery(201, order, "123 Street, NY", "2025-02-28", "Shipped")

print("\nDelivery Details:\n" + "=" \* 30)

delivery.display()

**output**



**Github link**

<https://github.com/h2qz/delivery-management-system-.git>

**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.