Assignment 1

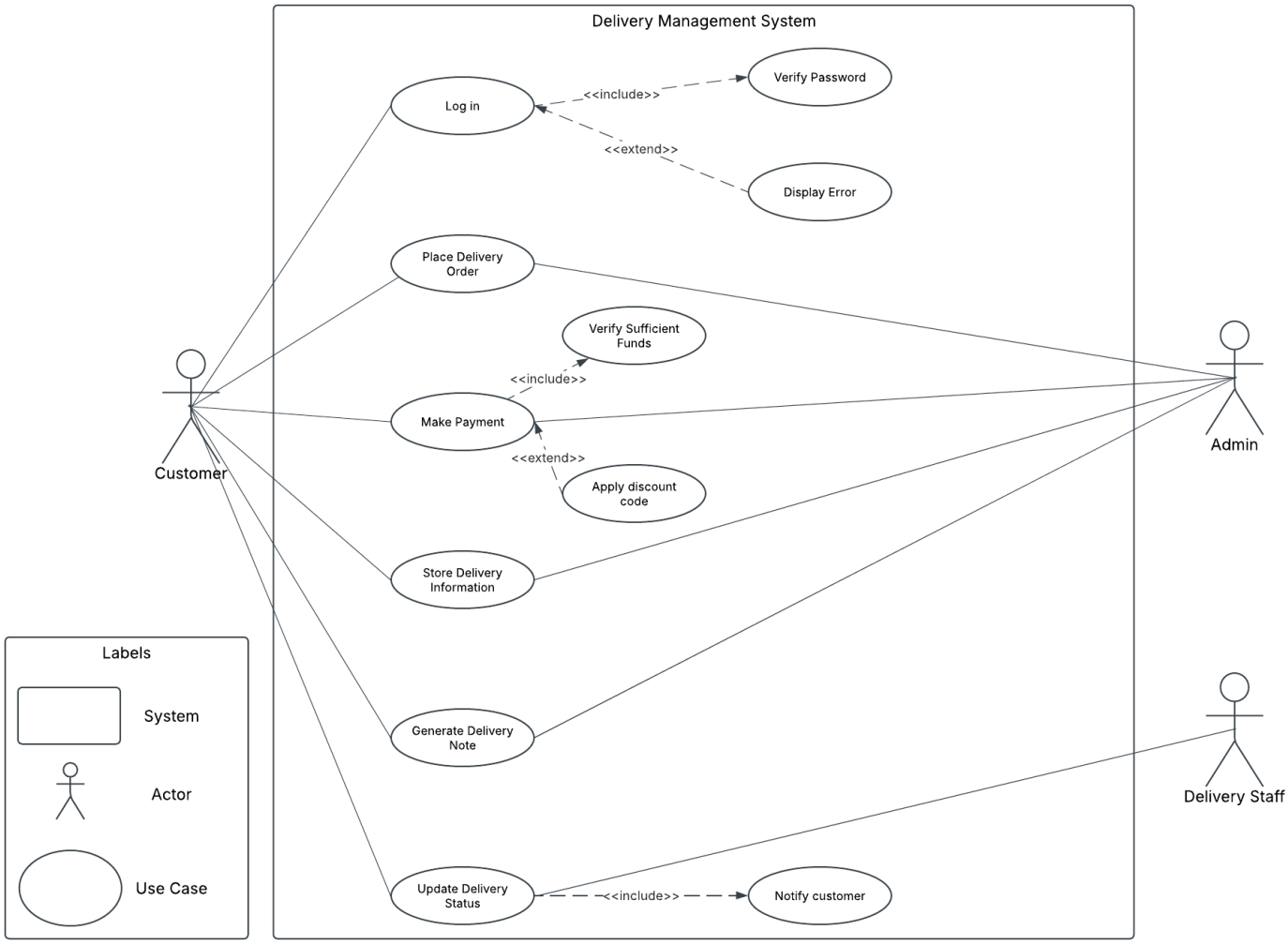
Ahmed Alremeithi - 202221113

ICS220 - Program. Fund.

Professor Afshan Parkar

February 28, 2025

UML Use-Case Diagram:



Link (UML Use-Case Diagram Page #1):

https://lucid.app/lucidchart/35145696-f3bf-4dda-a211-3ef1cbf5ae6a/edit?viewport_loc=-625%2C69%2C1777%2C1460%2C0_0&invitationId=inv_b216f8f1-abeb-45ba-afb6-829eb8428dd0

Use-Case Description Tables:

#1: Log In + Include & Extend

Use Case:	**Log in**
Trigger:	The customer attempts to log in by entering their username and password.
Notes:	Extends “Display Error”
Preconditions:	1. The user must have a valid account in the system 2. The system must be online and operational
Main Scenarios:	1. The system prompts the user to enter login credentials. 2. The user enters their username and password. 3. <<Include>> “Verify Password” 4. The user is logged in and granted access
Exceptions:	1a. The password is incorrect → The system extends “Display Error” 1b. The username does not exist. → The system extends “Display Error”

Use Case:	Verify Password
Trigger:	The system needs to verify the entered password.
Notes:	This use-case is included in “Log In”
Preconditions:	1. The user has entered a password 2. The system must have access to stored credentials
Main Scenarios:	1. The system retrieves the stored password for the username. 2. The system compares the entered password with the stored password. 3. If the password matches, authentication is successful. 4. If the password does not match, an error is triggered.
Exceptions:	1a. The stored password is corrupted or unavailable. → The system cannot verify credentials.

Use Case:	**Display Error**
Trigger:	The system encounters an issue preventing a successful login.
Notes:	This use-case is extended by “Log In”.
Preconditions:	1. The user attempted to log in but encountered an issue. 2. The system is able to detect errors.
Main Scenarios:	1. The system identifies the error type. 2. The system displays an appropriate error message (e.g. “Invalid Password”, “Invalid Username”). 3. The system prompts the user to retry logging in.
Exceptions:	2a. The system fails to retrieve error messages. → A generic error message is displayed.

#2: Place Delivery Order

Use Case:	**Place Delivery Order**
Trigger:	The customer submits a delivery order with recipient details, delivery address, and item list.
Notes:	No include or extends.
Preconditions:	1. The customer must be logged into their account 2. The system must be operational and able to process orders.
Main Scenarios:	1. The system prompts the customer to enter order details. 2. The customer provides recipient details, delivery address, and item list. 3. The system calculates the total cost, including taxes and fees. 4. The customer reviews the order and confirms submission. 5. The system stores the order information and assigns a unique order number. 6. The system generates an order confirmation and displays the order number.
Exceptions:	2a. The customer does not provide mandatory details (e.g. address, recipient information) → The system prompts for missing info.

#3: Make Payment + Include & Extend

Use Case:	**Make Payment**
Trigger:	The customer proceeds to pay for a placed delivery order.
Notes:	Extends “Apply Discount Code”
Preconditions:	1. The customer has a confirmed delivery order. 2. The system is connected to a valid payment processor.
Main Scenarios:	1. The system prompts the customer to select a payment method. 2. The customer enters payment details and confirms payment. 3. <<Include>> “Verify Sufficient Funds” 4. The system processes the payment. 5. The system confirms payment success and generates a receipt.
Exceptions:	1a. The payment method is invalid. → The system prompts the customer to select a different method. 5a. Payment processing fails. → The customer is informed of the failure and asked to retry.

Use Case:	Verify Sufficient Funds
Trigger:	The system needs to confirm that the customer has enough funds to complete the payment.
Notes:	This use-case is included in “Make Payment”.
Preconditions:	1. The customer has entered payment details. 2. The system is able to access the payment method’s balance or credit limit.
Main Scenarios:	1. The system checks the available balance of the selected payment method. 2. If sufficient funds are available, the payment proceeds. 3. If there are insufficient funds, the system declines the payment. 4. The customer reviews the order and confirms submission. 5. The system stores the order information and assigns a unique order number. 6. The system generates an order confirmation and displays the order number.
Exceptions:	1a. The system cannot access the payment balance due to a technical issue. → The system prompts the customer to retry later.

Use Case:	**Apply Discount Code**
Trigger:	The customer chooses to apply a discount code during payment.
Notes:	This use-case is extended by “Make Payment”.
Preconditions:	1. The customer has a valid discount code. 2. The system supports discount application for the current order.
Main Scenarios:	1. The customer enters a discount code. 2. The system checks the validity of the code. 3. If valid, the system applies the discount and recalculates the total price. 4. The system updates the payment amount.
Exceptions:	2a. The discount code is expired or invalid. → The system notifies the customer and does not apply the discount.

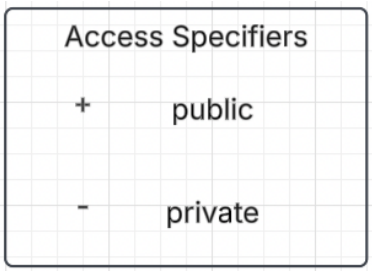
#4: Store Delivery Information

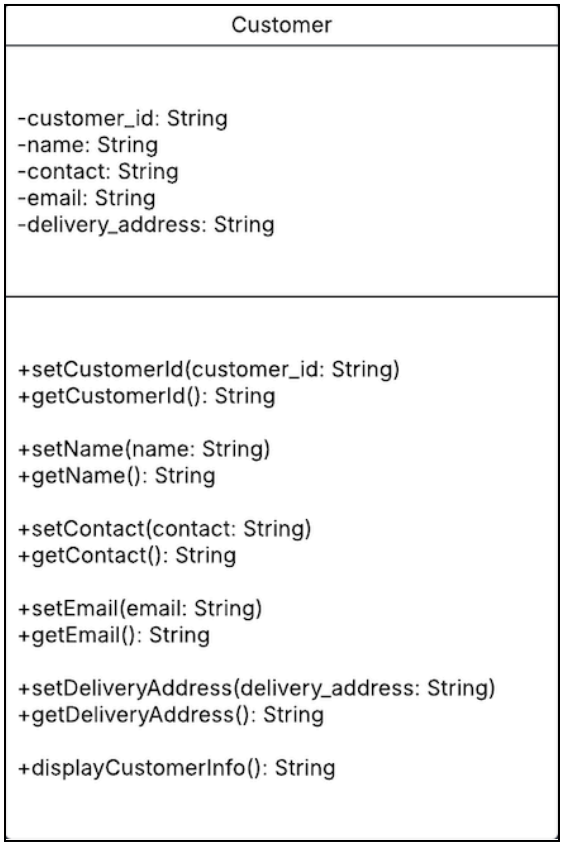
Use Case:	Store Delivery Information
Trigger:	A customer places a delivery order, and the system needs to store delivery details.
Notes:	No include or extends.
Preconditions:	1. The customer has successfully placed an order. 2. The system is operational and connected to the database.
Main Scenarios:	1. The system assigns a unique delivery number to the order. 2. The system stores the recipient details, delivery address, and contact information. 3. The system records the order's reference number, delivery date, and method. 4. The system saves package dimensions and total weight. 5. The system updates the order status to "Processing".
Exceptions:	2a. The system fails to store delivery information due to a database error. → The system notifies the customer and asks them to retry later.

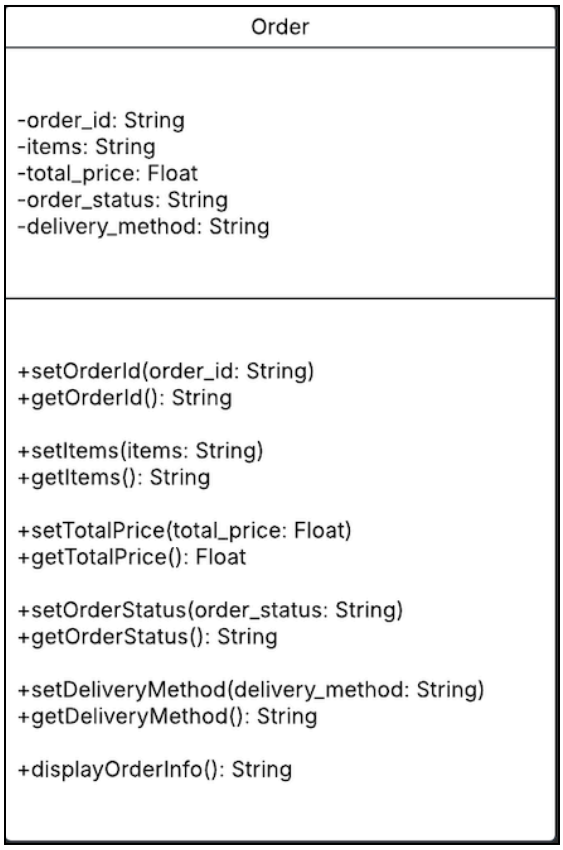
#5: Generate Delivery Note

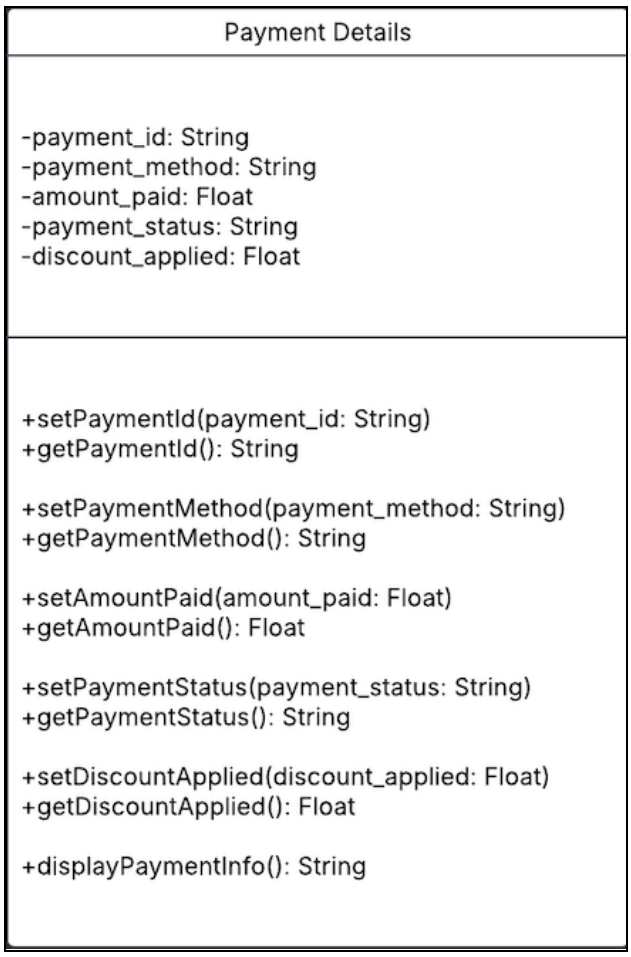
Use Case:	**Generate Delivery Note**
Trigger:	A customer successfully places an order, and the system needs to generate a delivery note.
Notes:	No include or extends.
Preconditions:	1. The order has been placed and stored in the system. 2. The system has access to all necessary order and delivery details.
Main Scenarios:	1. The system retrieves the order details, including recipient information, delivery address, and item list. 2. The system fetches the delivery information, such as delivery number, reference number, delivery date, and package details. 3. The system calculates the total charges, including taxes and fees. 4. The system formats all information into a structured delivery note. 5. The system generates a downloadable and printable delivery note.
Exceptions:	5a. The system encounters an error while generating the delivery note. → The system displays an error message and prompts the user to retry later.

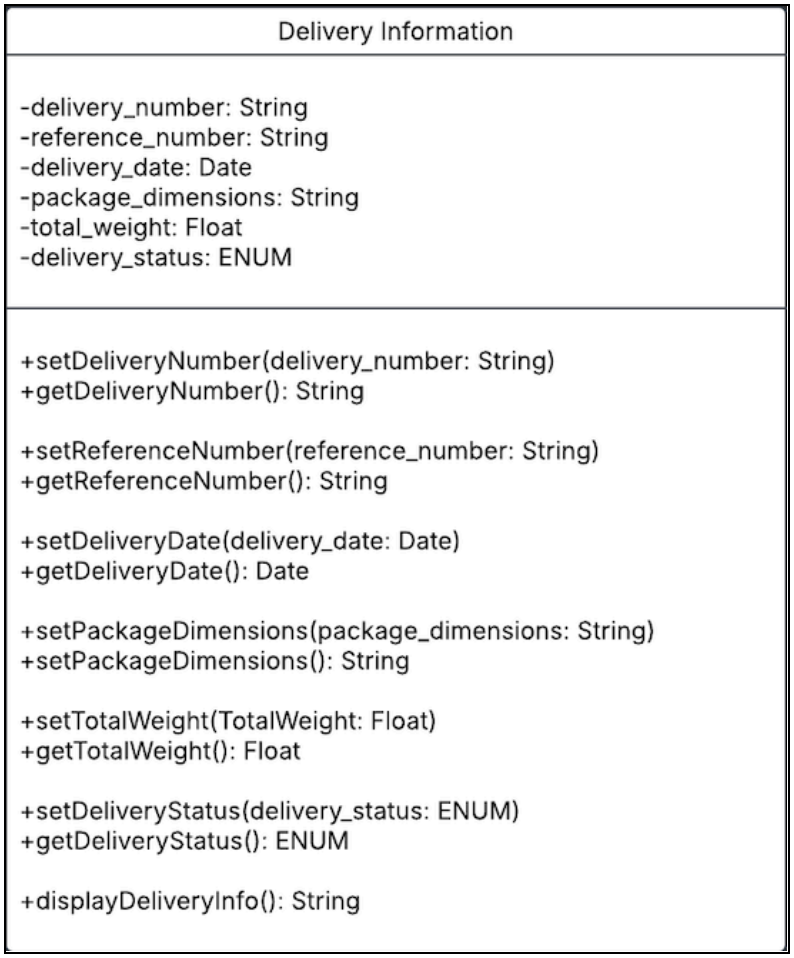
UML Class Diagrams:

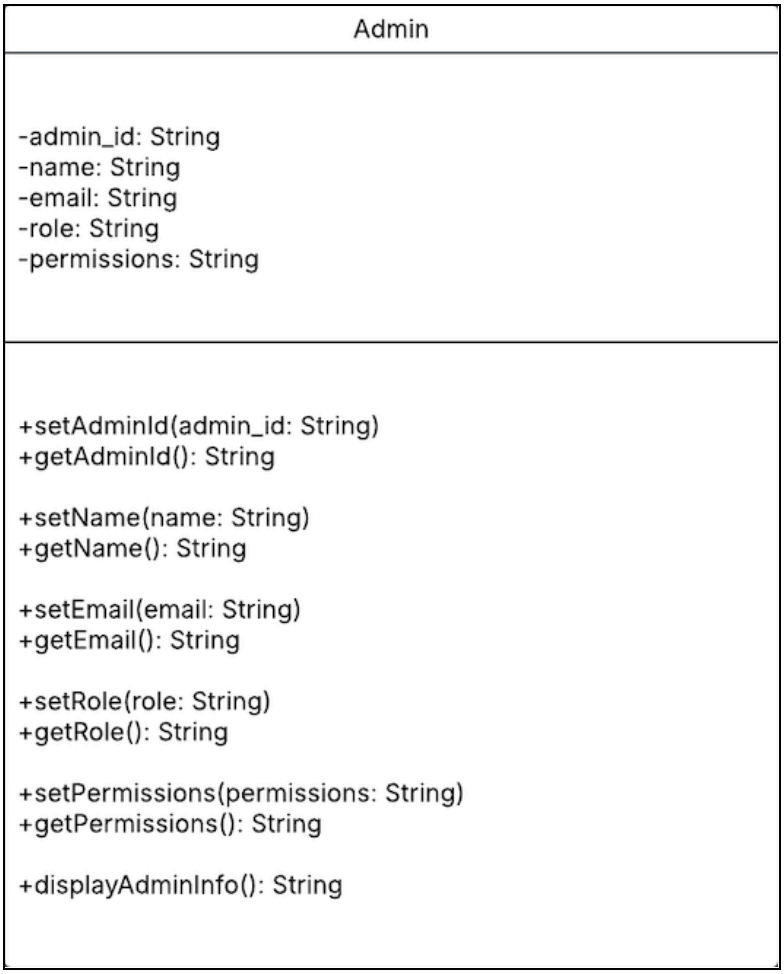







Payment Details
-payment_id: String
-payment_method: String
-amount_paid: Float
-payment_status: String
-discount_applied: Float
+setPaymentId(payment_id: String)
+getPaymentId(): String
+setPaymentMethod(payment_method: String)
+getPaymentMethod(): String
+setAmountPaid(amount_paid: Float)
+getAmountPaid(): Float
+setPaymentStatus(payment_status: String)
+getPaymentStatus(): String
+setDiscountApplied(discount_applied: Float)
+getDiscountApplied(): Float
+displayPaymentInfo(): String


Delivery Information
-delivery_number: String
-reference_number: String
-delivery_date: Date
-package_dimensions: String
-total_weight: Float
-delivery_status: ENUM
+setDeliveryNumber(delivery_number: String)
+getDeliveryNumber(): String
+setReferenceNumber(reference_number: String)
+getReferenceNumber(): String
+setDeliveryDate(delivery_date: Date)
+getDeliveryDate(): Date
+setPackageDimensions(package_dimensions: String)
+setPackageDimensions(): String
+setTotalWeight(TotalWeight: Float)
+getTotalWeight(): Float
+setDeliveryStatus(delivery_status: ENUM)
+getDeliveryStatus(): ENUM
+displayDeliveryInfo(): String



Link (UML Class Diagrams Page #2):

https://lucid.app/lucidchart/35145696-f3bf-4dda-a211-3ef1cbf5ae6a/edit?viewport_loc=-1131%2C-132%2C2389%2C1304%2C.G1WIRqcsUOR&invitationId=inv_b216f8f1-abeb-45ba-afb6-829eb8428dd0

Python Code:

```
"""Delivery Management System"""

from datetime import datetime
from enum import Enum

# ENUM Class for Delivery Status
class DeliveryStatus(Enum):
   ORDER_PLACED = "Order Placed"
   SHIPPED = "Shipped"
   ON_THE_WAY = "On The Way"
   DELIVERED = "Delivered"

# Class: Customer
class Customer:
   """Represents a customer placing a delivery order."""

   def __init__(self, customer_id, name, contact, email,
delivery_address):
       self.__customer_id = customer_id
       self.__name = name
       self.__contact = contact
       self.__email = email
       self.__delivery_address = delivery_address

   # Getters and Setters
   def setCustomerId(self, customer_id):
       self.__customer_id = customer_id

   def getCustomerId(self):
       return self.__customer_id

   def setName(self, name):
       self.__name = name

   def getName(self):
       return self.__name
```

```
    def setContact(self, contact):
        self.__contact = contact

    def getContact(self):
        return self.__contact

    def setEmail(self, email):
        self.__email = email

    def getEmail(self):
        return self.__email

    def setDeliveryAddress(self, delivery_address):
        self.__delivery_address = delivery_address

    def getDeliveryAddress(self):
        return self.__delivery_address

    def displayCustomerInfo(self):
        """Displays customer details."""
        return f"Customer ID: {self.__customer_id}, Name: {self.__name},
Contact: {self.__contact}, Email: {self.__email}, Address:
{self.__delivery_address}"

# Class: Order
class Order:
    """Represents an order placed by the customer."""

    def __init__(self, order_id, items, total_price, order_status,
delivery_method):
        self.__order_id = order_id
        self.__items = items
        self.__total_price = total_price
        self.__order_status = order_status
        self.__delivery_method = delivery_method

    # Getters and Setters
    def setOrderId(self, order_id):
        self.__order_id = order_id
```

```
    def getOrderId(self):
        return self.__order_id

    def setItems(self, items):
        self.__items = items

    def getItems(self):
        return self.__items

    def setTotalPrice(self, total_price):
        self.__total_price = total_price

    def getTotalPrice(self):
        return self.__total_price

    def setOrderStatus(self, order_status):
        self.__order_status = order_status

    def getOrderStatus(self):
        return self.__order_status

    def setDeliveryMethod(self, delivery_method):
        self.__delivery_method = delivery_method

    def getDeliveryMethod(self):
        return self.__delivery_method

    def displayOrderInfo(self):
        """Displays order details."""
        return f"Order ID: {self.__order_id}, Items: {self.__items}, Total
Price: {self.__total_price}, Status: {self.__order_status}, Delivery
Method: {self.__delivery_method}"

# Class: PaymentDetails
class PaymentDetails:
    """Handles payment details related to an order."""

    def __init__(self, payment_id, payment_method, amount_paid,
payment_status, discount_applied):
```

```
        self.__payment_id = payment_id
        self.__payment_method = payment_method
        self.__amount_paid = amount_paid
        self.__payment_status = payment_status
        self.__discount_applied = discount_applied

    # Getters and Setters
    def setPaymentId(self, payment_id):
        self.__payment_id = payment_id

    def getPaymentId(self):
        return self.__payment_id

    def setPaymentMethod(self, payment_method):
        self.__payment_method = payment_method

    def getPaymentMethod(self):
        return self.__payment_method

    def setAmountPaid(self, amount_paid):
        self.__amount_paid = amount_paid

    def getAmountPaid(self):
        return self.__amount_paid

    def setPaymentStatus(self, payment_status):
        self.__payment_status = payment_status

    def getPaymentStatus(self):
        return self.__payment_status

    def setDiscountApplied(self, discount_applied):
        self.__discount_applied = discount_applied

    def getDiscountApplied(self):
        return self.__discount_applied

    def displayPaymentInfo(self):
        """Displays payment details."""
```

```
        return f"Payment ID: {self.__payment_id}, Method:
{self.__payment_method}, Amount Paid: {self.__amount_paid}, Status:
{self.__payment_status}, Discount Applied: {self.__discount_applied}"

# Class: DeliveryInformation
class DeliveryInformation:
    """Stores delivery information for an order."""

    def __init__(self, delivery_number, reference_number, delivery_date,
package_dimensions, total_weight, delivery_status):
        self.__delivery_number = delivery_number
        self.__reference_number = reference_number
        self.__delivery_date = delivery_date
        self.__package_dimensions = package_dimensions
        self.__total_weight = total_weight
        self.__delivery_status = delivery_status

    # Getters and Setters
    def setDeliveryNumber(self, delivery_number):
        self.__delivery_number = delivery_number

    def getDeliveryNumber(self):
        return self.__delivery_number

    def setReferenceNumber(self, reference_number):
        self.__reference_number = reference_number

    def getReferenceNumber(self):
        return self.__reference_number

    def setDeliveryDate(self, delivery_date):
        self.__delivery_date = delivery_date

    def getDeliveryDate(self):
        return self.__delivery_date

    def setPackageDimensions(self, package_dimensions):
        self.__package_dimensions = package_dimensions

    def getPackageDimensions(self):
```

```
        return self.__package_dimensions

    def setTotalWeight(self, total_weight):
        self.__total_weight = total_weight

    def getTotalWeight(self):
        return self.__total_weight

    def setDeliveryStatus(self, delivery_status):
        self.__delivery_status = delivery_status

    def getDeliveryStatus(self):
        return self.__delivery_status

    def displayDeliveryInfo(self):
        """Displays delivery details."""
        return f"Delivery Number: {self.__delivery_number}, Reference:
{self.__reference_number}, Date: {self.__delivery_date}, Package:
{self.__package_dimensions}, Weight: {self.__total_weight}, Status:
{self.__delivery_status.value}"

# Class: Admin
class Admin:
    """Represents an admin who manages the system."""

    def __init__(self, admin_id, name, email, role, permissions):
        self.__admin_id = admin_id
        self.__name = name
        self.__email = email
        self.__role = role
        self.__permissions = permissions

    # Getters and Setters
    def setAdminId(self, admin_id):
        self.__admin_id = admin_id

    def getAdminId(self):
        return self.__admin_id

    def setName(self, name):
```

```
        self.__name = name

    def getName(self):
        return self.__name

    def setEmail(self, email):
        self.__email = email

    def getEmail(self):
        return self.__email

    def setRole(self, role):
        self.__role = role

    def getRole(self):
        return self.__role

    def setPermissions(self, permissions):
        self.__permissions = permissions

    def getPermissions(self):
        return self.__permissions

    def displayAdminInfo(self):
        """Displays admin details."""
        return f"Admin ID: {self.__admin_id}, Name: {self.__name}, Email:
{self.__email}, Role: {self.__role}, Permissions: {self.__permissions}"

# Creating two Customer objects
customer1 = Customer("CUST001", "Ahmed Alremeithi", "0501234567",
"202221113@zu.ac.ae", "Abu Dhabi, UAE")
customer2 = Customer("CUST002", "Thani Alremeithi", "0507654321",
"202221114@zu.ac.ae", "Dubai, UAE")

# Creating two Order objects
order1 = Order("ORD001", "Wireless Keyboard, Laptop Cooling Pad", 220.00,
"Confirmed", "Courier")
order2 = Order("ORD002", "Camera Lock, Wireless Mouse & Pad Set", 90.00,
"Pending", "Courier")
```

```
# Creating two PaymentDetails objects
payment1 = PaymentDetails("PAY001", "Credit Card", 220.00, "Completed",
10.00)
payment2 = PaymentDetails("PAY002", "Debit Card", 90.00, "Processing",
5.00)

# Creating two DeliveryInformation objects
delivery1 = DeliveryInformation("DEL001", "REF12345", "2025-01-25",
"Medium Box", 7.0, DeliveryStatus.SHIPPED)
delivery2 = DeliveryInformation("DEL002", "REF67890", "2025-01-28", "Small
Box", 2.5, DeliveryStatus.ORDER_PLACED)

# Creating two Admin objects
admin1 = Admin("ADM001", "Khalid Alkaabi", "admin1@delivery.com",
"Manager", "Full Access")
admin2 = Admin("ADM002", "Saeed Almansoori", "admin2@delivery.com",
"Supervisor", "Limited Access")


# Printing Customer 1's details together
print("\n" + "-" * 50)
print("CUSTOMER 1: AHMED ALREMEITHI - ORDER DETAILS")
print("-" * 50)
print(customer1.displayCustomerInfo())
print(order1.displayOrderInfo())
print(payment1.displayPaymentInfo())
print(delivery1.displayDeliveryInfo())

# Printing Customer 2's details together
print("\n" + "-" * 50)
print("CUSTOMER 2: THANI ALREMEITHI - ORDER DETAILS")
print("-" * 50)
print(customer2.displayCustomerInfo())
print(order2.displayOrderInfo())
print(payment2.displayPaymentInfo())
print(delivery2.displayDeliveryInfo())

# Printing Admins separately for reference
print("\n" + "-" * 50)
print("ADMIN DETAILS")
```

```
print("-" * 50)
print(admin1.displayAdminInfo())
print(admin2.displayAdminInfo())
```

Output:

CUSTOMER 1: AHMED ALREMEITHI - ORDER DETAILS

Customer ID: CUST001, Name: Ahmed Alremeithi, Contact: 0501234567, Email: 202221113@zu.ac.ae, Address: Abu Dhabi, UAE

Order ID: ORD001, Items: Wireless Keyboard, Laptop Cooling Pad, Total Price: 220.0, Status: Confirmed, Delivery Method: Courier

Payment ID: PAY001, Method: Credit Card, Amount Paid: 220.0, Status: Completed, Discount Applied: 10.0

Delivery Number: DEL001, Reference: REF12345, Date: 2025-01-25, Package: Medium Box, Weight: 7.0, Status: Shipped

---

CUSTOMER 2: THANI ALREMEITHI - ORDER DETAILS

Customer ID: CUST002, Name: Thani Alremeithi, Contact: 0507654321, Email: 202221114@zu.ac.ae, Address: Dubai, UAE

Order ID: ORD002, Items: Camera Lock, Wireless Mouse & Pad Set, Total Price: 90.0, Status: Pending, Delivery Method: Courier

Payment ID: PAY002, Method: Debit Card, Amount Paid: 90.0, Status: Processing, Discount Applied: 5.0

Delivery Number: DEL002, Reference: REF67890, Date: 2025-01-28, Package: Small Box, Weight: 2.5, Status: Order Placed

---

ADMIN DETAILS

Admin ID: ADM001, Name: Khalid Alkaabi, Email: admin1@delivery.com, Role: Manager, Permissions: Full Access

Admin ID: ADM002, Name: Saeed Almansoori, Email: admin2@delivery.com, Role: Supervisor, Permissions: Limited Access

Github Repository Link: https://github.com/AhmedAlremeithi/Coding

Summary of learnings:

During this assignment, I learned how to analyze a real-world system and break it down into clear use cases using UML diagrams. This helped me understand how different parts of a system interact and how to organize information efficiently. I also improved my object-oriented programming (OOP) skills by creating classes with private attributes, getters, and setters, ensuring data encapsulation and security. Writing the display functions allowed me to see how data is stored and retrieved, making it easier to present structured information.

Additionally, I practiced creating multiple objects from each class, which showed me how different components of a system work together. Formatting the output properly also helped me understand the importance of clear and organized data presentation. This assignment reinforced the importance of planning before coding, ensuring that every class and function is structured correctly. Overall, this project gave me valuable experience in system design, OOP principles, and software development best practices.