Assignment 1: Delivery Management System

Maryam A. Nasib

Interdisciplinary Studies, Zayed University

ICS220 - 21383 Programming Fundamentals

Dr. Andrew Leonce

February 28th, 2025

Delivery Management System:

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:
Total Weight: 7 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

A delivery company in your city requires software to manage its delivery system. The system needs to handle delivery orders, manage delivery details, and generate a delivery note for each transaction. A sample delivery note is provided in the figure above to understand the required data fields.

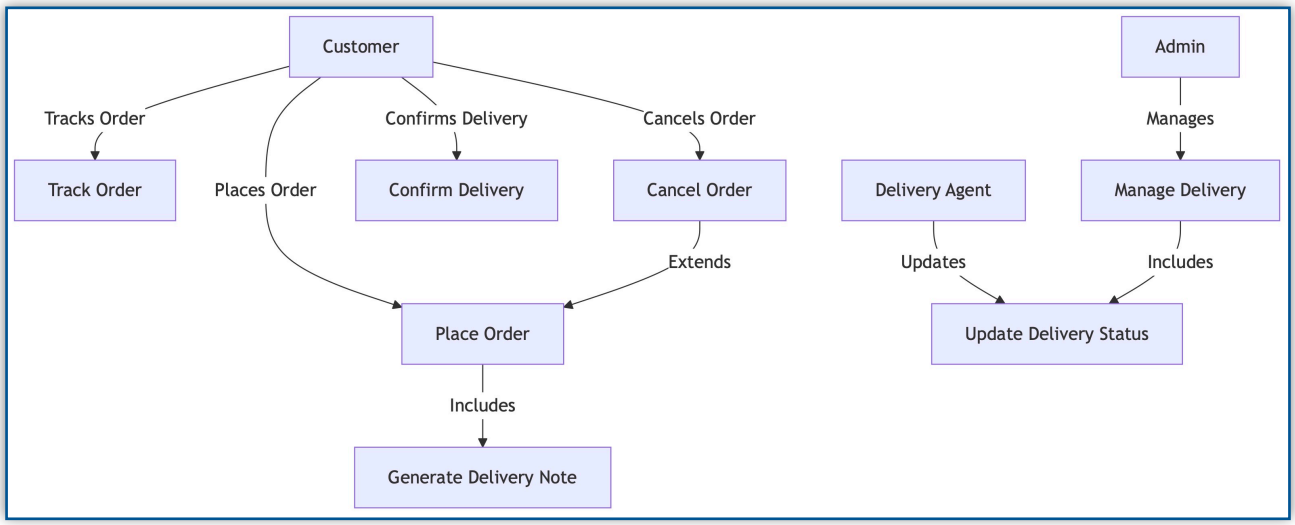
1. UML Use-Case, diagrams and description

a. Identify the use-cases for the delivery management system

1. Place order: The customer chooses an item and places an order for delivery via the delivery company website.
2. Manage delivery: the person in charge chooses a delivery agent, monitors the status, and handles logistics.
3. Generate delivery note: The system generates a delivery note that includes all relevant information.
4. Track order: Customers may get live notifications on the status of orders.
5. Cancel order: A customer can cancel their order before its been sent out for delivery.

6. Confirming delivery: After the delivery is completed, the customer verifies having received the items.

b. Draw the UML use-case diagram and include supporting use-case description tables using standard tools



c. Include at least 3 scenarios. Ensure that "include" and extend" relationships are added

Scenario 1: A customer placing an order (Base Use-Case)

1. The customer logs into the website system.
2. The customer chooses items and adds them to their shopping cart.
3. The customer moves to the checkout page and inputs their delivery information.
4. The system checks the information and calculates the final cost of their order.
5. The order is verified, and a unique order number is assigned.
6. Generate a delivery note (include), a delivery note is generated instantly by the system as soon as an order is submitted.
7. The order is then switched to "pending."

Scenario 2: A customer cancelling an order (Extended Use-Case)

1. The. customer logs into the website system to examine their order history.
2. The customer picks an order from their order history with a status that says "pending.".
3. The system determines if the order is allowed to be cancelled.
4. If so, the system cancels the order and changes its status.
5. Cancel order (extend), this case only applies if the order has already been placed.

Scenario 3: The administrator handles a delivery (Base Use-Case)

1. The administrator logs into the website system.
2. The administrator reviews pending orders and appoints a courier.
3. The designated courier is alerted and keeps the delivery status up-to-date.
4. The administrator confirms that the order has been successfully delivered to the customer in the system.

5. Updating the delivery status (include), the courier has to update the order status at every point during delivery.

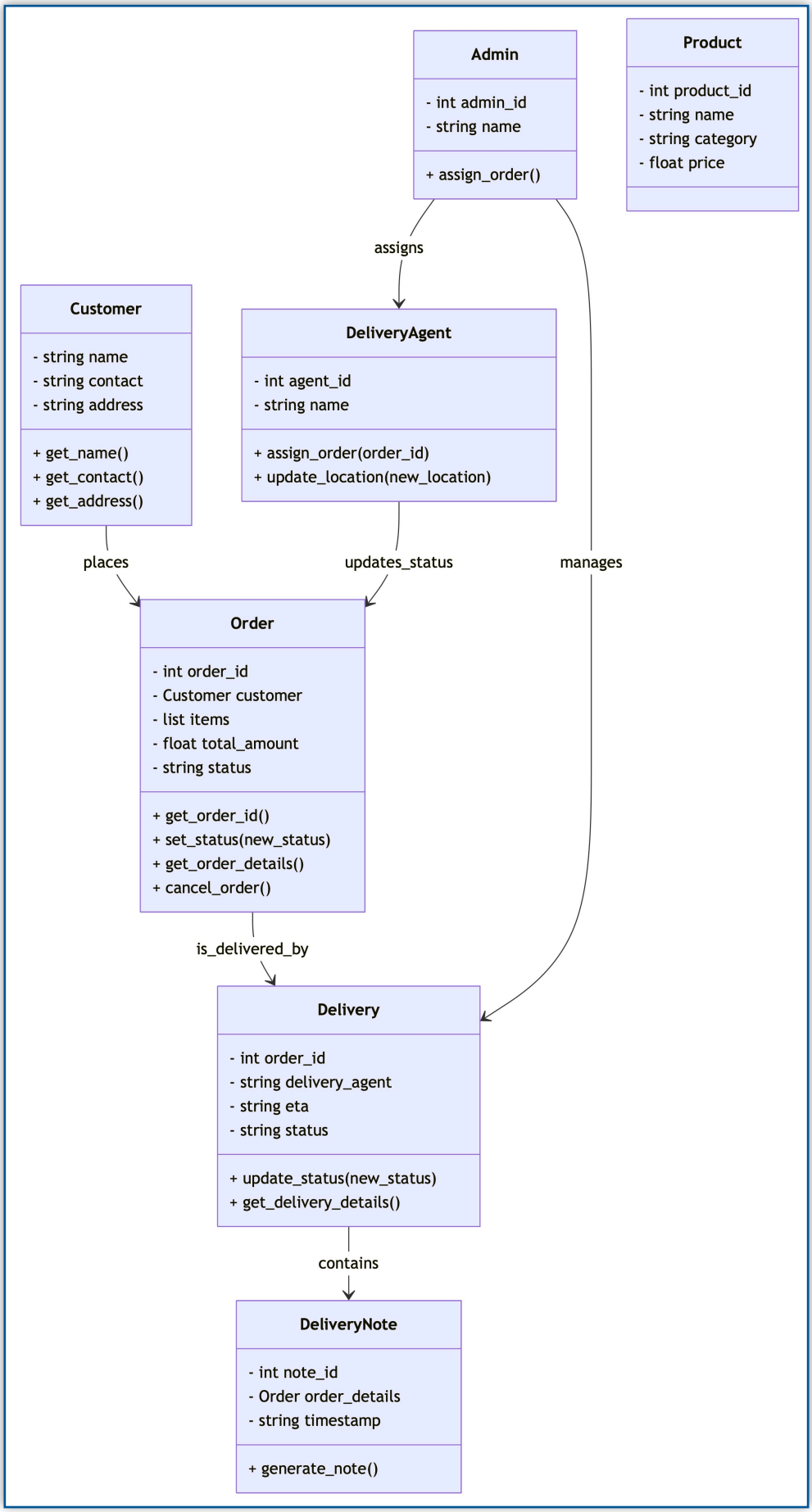
Use-Case	Actors	Preconditions		Postconditions
Place Order	Customer	Customer logs in	1. Select items 2. Adds to cart 3. Proceeds to checkout 4. Offer is confirmed	Order is verified and created/stored in the system
Cancel Order	Customer	Order is pending	1. Customer selects an order 2. Requests cancellation	Order status is modified to “Cancelled” and/or removed
Manage Delivery	Admin	Order is verified and is pending	1. Assigns deliver agent 2. Updates status	Order status is modified to “Out for Delivery”

2. UML Class, diagrams and description

a. Based on the use-case descriptions, identify the objects and their respective classes.

1. Order: contains order information such as order number, customer name, and item code list.
2. Customer: handles customer information such as name, combatant information like their email and delivery address.
3. Deliver: Responsible for delivery logistics, including the assigned courier and anticipated delivery time.
4. Delivery note: A delivery note is generated, which includes structured delivery notes.
5. Admin: handles order assignments and monitors delivery tracking.
6. Courier: In charge of completing orders and reporting status.
7. Product: Contains product information, including name, price, and classification.

b. Draw the UML class diagram and provide supporting descriptions to explain the identified classes.



c. Include at least 4 classes and establish access specifiers for member visibility.

Class: Order

- Which represents a customers order details.
- Attributtes:
  - `-order_id` (private): orders distinctive identification.
  - `-customer` (private): the person who submitted the order.
  - `-items` (private): A list of items ordered.
  - `-total_amount` (private): The overall order cost.
  - `-status` (private): The latest status of the order (pending, delivered, cancelled, etc.)

- Methods:
  - `+get_order_details()`: returns order information in string format.
  - `+update_status(new_status)`: keeps the order status up-to-date.
  - `+cancel_order(new_status)`: a method that is used to change the '__status' of the order to "Cancelled".

Class: Delivery

- It's responsible for order logistics, transportation and delivery aspect of the order.
- Attributtes:
  - `-order_id` (private): orders distinctive identification.
  - `-delivery_agent` (private): the name of the courier.
  - `-eta` (private): expected delivery time.
  - `-status` (private): indicates the present delivery status

- Methods:
  - `+update_status(new_status)`: keeps the order status up-to-date.
  - `+get_delivery_details(self)`: it manages the delivery information.

Class: Delivery Note

- Which contains structured information about an orders delivery.
- Attributtes:
  - `-note_id` (private): a distinctive identification for the delivery note.
  - `-order_details` (private): information about the associated order.
  - `-timestamp` (private): the time when the note was created.

- Methods:
  - `+generate_note()`: turns formatted delivery note details.

3. **Created Python classes and objects:**
a. **Constructor (__init__) and at least 5 attributes per class.**
b. **Required setter and getter methods**
c. **Other required function headers in the classes where the functions body is just a pass statement and include a comment to indicate what the function should achieve.**

4. **Use objects to generate a Delivery Note:**
a. **Create objects of all identified classes.**
b. **Use these objects and their functions to populate and display the information provided in the figure.**

Code:

```
class Order:
    """Represents a customer's purchase order."""
    def __init__(self, order_id, customer, items, total_amount,
status="Pending"):
#3.a.Constructor(__init__) and at least 5 attributes per class
        self.__order_id = order_id
        self.__customer = customer
        self.__items = items
        self.__total_amount = total_amount
        self.__status = status

#3.b.Required setter and getter methods
    def get_order_id(self):
        return self.__order_id

    def set_status(self, new_status):
        self.__status = new_status

    def get_order_details(self):
        return f"Order {self.__order_id}: {self.__items}, Total:
{self.__total_amount}, Status: {self.__status}"

#3.c.Other required function headers in the classes where the
functions body is just a pass statement and include a comment to
indicate what the function should achieve.
    def cancel_order(self):
        """Cancel the order if it's still pending."""
        self.__status = "Cancelled"
        print(f"Order {self.__order_id} has been cancelled.")

class Customer:
    """Represents a customer in the system."""
    def __init__(self, name, contact, address):
        self.__name = name
```

```
        self.__contact = contact
        self.__address = address

    def get_name(self):
        return self.__name

    def get_contact(self):
        return self.__contact

    def get_address(self):
        return self.__address

class Delivery:
    """Handles delivery logistics."""
    def __init__(self, order_id, delivery_agent, eta, status="In
Transit"):
        self.__order_id = order_id
        self.__delivery_agent = delivery_agent
        self.__eta = eta
        self.__status = status

    def update_status(self, new_status):
        self.__status = new_status
        print(f"Delivery status updated to {self.__status} for
Order {self.__order_id}.")

    def get_delivery_details(self):
        return f"Order {self.__order_id}: Assigned to
{self.__delivery_agent}, ETA: {self.__eta}, Status:
{self.__status}"

class DeliveryNote:
    """Stores delivery details and generates notes."""
    def __init__(self, note_id, order_details, timestamp):
        self.__note_id = note_id
        self.__order_details = order_details
        self.__timestamp = timestamp

    def generate_note(self):
        """Generate a formatted delivery note."""
        print(f"Delivery Note {self.__note_id}:
{self.__order_details}, Generated on {self.__timestamp}")


#4.a.Create objects of all identified classes.
#Scenarios with randomly chosen values
```

```
customer1 = Customer("Maryam Abdulsalam", "050-123-4567", "Dubai")
# Updated customer details
order1 = Order(53672, customer1, ["Grilled Cheese Burger Slider",
"Orange Juice"], 35.5, "Pending") #Updated order details
delivery1 = Delivery(53672, "Sadiq", "40 minutes")  # Delivery
details
note1 = DeliveryNote(158, order1.get_order_details(), "2025-5-24
7:46:08")  # Note details

#4.b.Use these objects and their functions to populate and display
the information provided in the figure.
#Call method in order to print out information
print(f"Customer Name: {customer1.get_name()}")
print(f"Customer Address: {customer1.get_address()}")
print(order1.get_order_details().replace('Total: 35.5', 'Total:
35.5dhs')) # Format total amount
delivery1.update_status("Delivered")
note1.generate_note()
```

Output:

```
Customer Name: Maryam Abdulsalam
Customer Address: Dubai
Order 53672: ['Grilled Cheese Burger Slider', 'Orange Juice'],
Total: 35.5dhs, Status: Pending
Delivery status updated to Delivered for Order 53672.
Delivery Note 158: Order 53672: ['Grilled Cheese Burger Slider',
'Orange Juice'], Total: 35.5, Status: Pending, Generated on
2025-5-24 7:46:08
```

Screenshot of code along with its output as proof that it was successfully run :

```
class Order:
    """Represents a customer's purchase order."""
    def __init__(self, order_id, customer, items, total_amount, status="Pending
#3.a.Constructor(__init__) and at least 5 attributes per class
        self.__order_id = order_id
        self.__customer = customer
        self.__items = items
        self.__total_amount = total_amount
        self.__status = status

#3.b.Required setter and getter methods
    def get_order_id(self):
        return self.__order_id

    def set_status(self, new_status):
        self.__status = new_status

    def get_order_details(self):
        return f"Order {self.__order_id}: {self.__items}, Total: {self.__total_

#3.c.Other required function headers in the classes where the functions body is
    def cancel_order(self):
        """Cancel the order if it's still pending."""
        self.__status = "Cancelled"
        print(f"Order {self.__order_id} has been cancelled.")

class Customer:
    """Represents a customer in the system."""
    def __init__(self, name, contact, address):
        self.__name = name
        self.__contact = contact
        self.__address = address

    def get_name(self):
        return self.__name

    def get_contact(self):
        return self.__contact

    def get_address(self):
        return self.__address

class Delivery:
    """Handles delivery logistics."""
    def __init__(self, order_id, delivery_agent, eta, status="In Transit"):
        self.__order_id = order_id
        self.__delivery_agent = delivery_agent
        self.__eta = eta
        self.__status = status

    def update_status(self, new_status):
        self.__status = new_status
        print(f"Delivery status updated to {self.__status} for Order {self.__or

    def get_delivery_details(self):
        return f"Order {self.__order_id}: Assigned to {self.__delivery_agent},

class DeliveryNote:
    """Stores delivery details and generates notes."""
    def __init__(self, note_id, order_details, timestamp):
        self.__note_id = note_id
        self.__order_details = order_details
        self.__timestamp = timestamp

    def generate_note(self):
        """Generate a formatted delivery note."""
        print(f"Delivery Note {self.__note_id}: {self.__order_details}, Generat


#4.a.Create objects of all identified classes.
#Scenarios with randomly chosen values
customer1 = Customer("Maryam Abdulsalam", "050-123-4567", "Dubai")  # Updated c
order1 = Order(53672, customer1, ["Grilled Cheese Burger Slider", "Orange Juice
delivery1 = Delivery(53672, "Sadiq", "40 minutes")  # Delivery details
note1 = DeliveryNote(158, order1.get_order_details(), "2025-5-24 7:46:08")  # N

#4.b.Use these objects and their functions to populate and display the informat
#Call method in order to print out information
print(f"Customer Name: {customer1.get_name()}")
print(f"Customer Address: {customer1.get_address()}")
print(order1.get_order_details().replace('Total: 35.5', 'Total: 35.5dhs')) # Fo
delivery1.update_status("Delivered")
note1.generate_note()
```

```
Customer Name: Maryam Abdulsalam
Customer Address: Dubai
Order 53672: ['Grilled Cheese Burger Slider', 'Orange Juice'], Total: 35.5dhs, Status: Pending
Delivery status updated to Delivered for Order 53672.
Delivery Note 158: Order 53672: ['Grilled Cheese Burger Slider', 'Orange Juice'], Total: 35.5, Status: Pending, Generated on 2025-5-24 7:46:08
```

5. Summary of learnings:

This assignment improved my understanding of UML diagrams used to represent system interactions as well as Python programming with objects and classes. I learned how to utilize use-case diagrams to describe system operations and class diagrams to organize links between them. I also explored encapsulation, getters and setters, and method functionality in Python.

One problem involved making sure that UML relationships (e.g., "Include" and "Extend") and access specifiers were defined correctly. I accomplished this by reviewing the diagrams and adjusting the relationships. Another obstacle was organizing Python classes while sticking to all the guidelines, such as placeholder methods to provide additional functionality.

Using GitHub for version control helped me stay organized and understand how utilizing such a website works. In conclusion, this assignment helped me improve my knowledge of UML use-cases, UML classes, and understanding how class attributes and methods work.

6. GitHub repository link:

https://github.com/MaryamAbdulsalam09/Delivery-Management-System.git

References:

Discuss.python.org. (n.d.). *Best practice: Setter and getter vs passing argument to constructor.* https://discuss.python.org/t/best-practice-setter-and-getter-vs-passing-arg-to-constructor/57405

Stack Overflow. (n.d.). *How to use setter in Python constructor.* https://stackoverflow.com/questions/47507272/how-to-use-setter-in-python-constructor

Stack Overflow. (n.d.). *How do getters and setters work in Python?* https://stackoverflow.com/questions/74956445/how-do-getters-and-setters-work-in-python

UAE Minerva Project. (n.d.). *Course portal.* https://forum.uae.minervaproject.com/app/courses/2165/sections/12203/classes/124095?course_id=2165

UAE Minerva Project. (n.d.). *Course portal.* https://forum.uae.minervaproject.com/app/courses/2165/sections/12203/classes/118033?course_id=2165

UAE Minerva Project. (n.d.). *Course portal.* https://forum.uae.minervaproject.com/app/courses/2165/sections/12203/classes/120020?course_id=2165

Mermaid Live. (n.d.). *Mermaid live editor.* https://mermaid.live/

Google Colab. (n.d.). *Collaboratory.* https://colab.google/