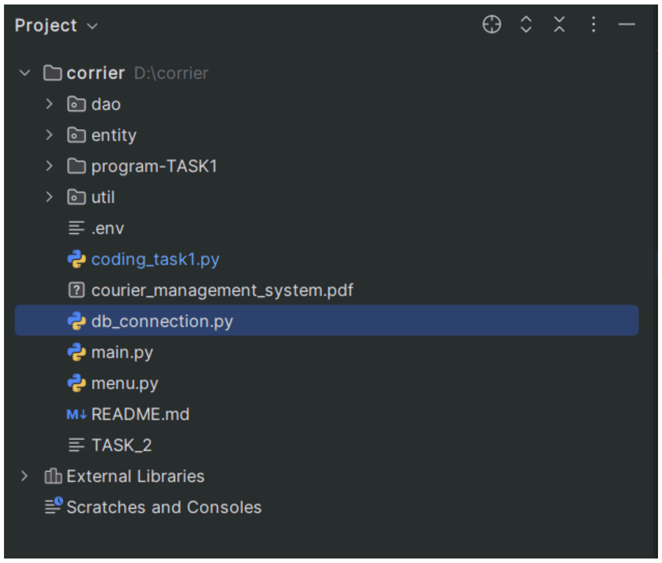
CODING

TASK 1

Connecting the database

Created a python file named as db_connection for connecting database

```
def get_db_connection():
    import mysql.connector

    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Vineeth1246@",
            database="courier_db"
        )
        return connection
    except mysql.connector.Error as err:
        print(f"Database Connection Error: {err}")
        return None
```



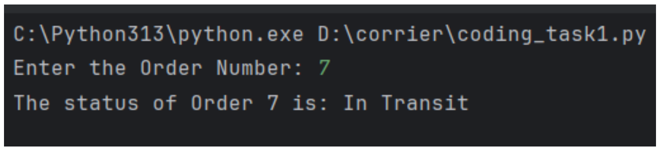
```
Project
corrier D:\corrier
  dao
  entity
  program-TASK1
  util
  .env
  coding_task1.py
  courier_management_system.pdf
  db_connection.py
  main.py
  menu.py
  README.md
  TASK_2
External Libraries
Scratches and Consoles
```

Control Flow Statements

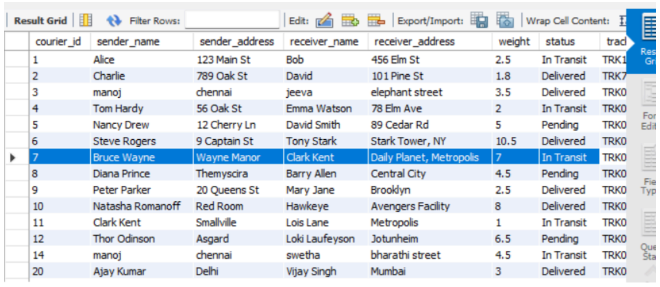
1. Write a program that checks whether a given order is delivered or not based on its status (e.g., "Processing," "Delivered," "Cancelled"). Use if-else statements for this.

```
def get_db_connection():
    import mysql.connector

    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Vineeth1246@",
            database="courier_db"
        )
        return connection
    except mysql.connector.Error as err:
        print(f"Database Connection Error: {err}")
        return None
```

output / database



```
C:\Python313\python.exe D:\corrier\coding_task1.py
Enter the Order Number: 7
The status of Order 7 is: In Transit
```



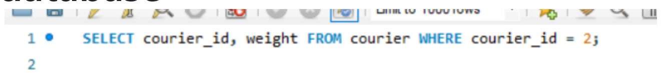
courier_id	sender_name	sender_address	receiver_name	receiver_address	weight	status	trac
1	Alice	123 Main St	Bob	456 Elm St	2.5	In Transit	TRK1
2	Charlie	789 Oak St	David	101 Pine St	1.8	Delivered	TRK7
3	manoj	chennai	jeeva	elephant street	3.5	Delivered	TRK0
4	Tom Hardy	56 Oak St	Emma Watson	78 Elm Ave	2	In Transit	TRK0
5	Nancy Drew	12 Cherry Ln	David Smith	89 Cedar Rd	5	Pending	TRK0
6	Steve Rogers	9 Captain St	Tony Stark	Stark Tower, NY	10.5	Delivered	TRK0
7	Bruce Wayne	Wayne Manor	Clark Kent	Daily Planet, Metropolis	7	In Transit	TRK0
8	Diana Prince	Themyscira	Barry Allen	Central City	4.5	Pending	TRK0
9	Peter Parker	20 Queens St	Mary Jane	Brooklyn	2.5	Delivered	TRK0
10	Natasha Romanoff	Red Room	Hawkeye	Avengers Facility	8	Delivered	TRK0
11	Clark Kent	Smallville	Lois Lane	Metropolis	1	In Transit	TRK0
12	Thor Odinson	Asgard	Loki Laufeyson	Jotunheim	6.5	Pending	TRK0
14	manoj	chennai	swetha	bharathi street	4.5	In Transit	TRK0
20	Ajay Kumar	Delhi	Vijay Singh	Mumbai	3	Delivered	TRK0

2. Implement a switch-case statement to categorize parcels based on their weight into "Light," "Medium," or "Heavy."

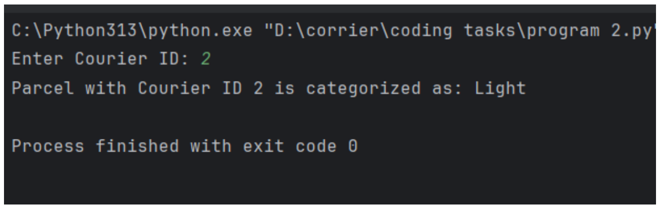
```
from db_connection import get_db_connection

def categorize_parcel(weight):
    """Categorize parcel based on weight."""
    categories = {
        "Light": lambda w: w < 2,
        "Medium": lambda w: 2 <= w < 5,
        "Heavy": lambda w: w >= 5
    }

    for category, condition in categories.items():
        if condition(weight):
            return category
    return "Unknown"

def get_parcel_weight(courier_id):
    """Fetch weight from the database based on courier_id."""
    conn = get_db_connection()
    cursor = conn.cursor()

    query = "SELECT weight FROM courier WHERE courier_id = %s"
    cursor.execute(query, (courier_id,))
    result = cursor.fetchone()

    conn.close()

    if result:
        return result[0]  # Extract weight value from result tuple
    else:
        return None

# Example usage
courier_id = int(input("Enter Courier ID: "))
```

```
weight = get_parcel_weight(courier_id)

if weight is not None:
    category = categorize_parcel(weight)
    print(f"Parcel with Courier ID {courier_id} is categorized as: {category}")
else:
    print("Courier ID not found!")
```

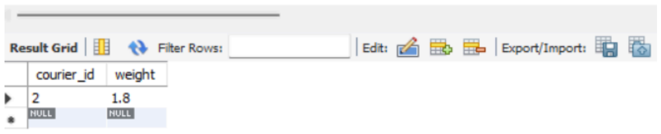
output / database



```
SELECT courier_id, weight FROM courier WHERE courier_id = 2;
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 2.py"
Enter Courier ID: 2
Parcel with Courier ID 2 is categorized as: Light

Process finished with exit code 0
```

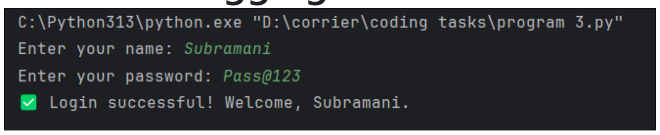


courier_id	weight
2	1.8
NULL	NULL

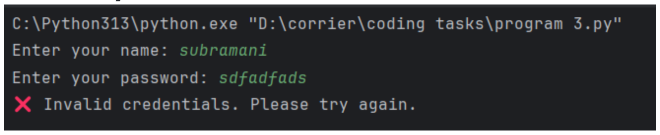
3. Implement User Authentication 1. Create a login system for employees and customers using Java control flow statements.

```
def get_db_connection():
    import mysql.connector

    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Vineeth1246@",
            database="courier_db"
        )
        return connection
    except mysql.connector.Error as err:
        print(f"Database Connection Error: {err}")
        return None

# Function to authenticate employee login
def employee_login(name, password):
    conn = get_db_connection()
    cursor = conn.cursor()

    query = "SELECT * FROM employee WHERE name = %s AND password = %s"
    cursor.execute(query, (name, password))
    result = cursor.fetchone()

    cursor.close()
    conn.close()

    if result:
        print(f" Login successful! Welcome, {name}.")
    else:
        print(" Invalid credentials. Please try again.")

# Example usage
if __name__ == "__main__":
    emp_name = input("Enter your name: ")
    emp_password = input("Enter your password: ")
    employee_login(emp_name, emp_password)
```

logging in with valid credentials / invalid credentials



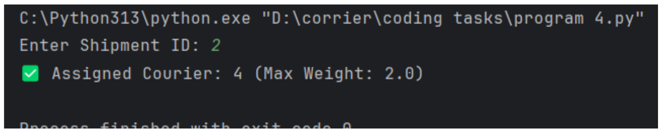
```
C:\Python313\python.exe "D:\corrier\coding tasks\program 3.py"
Enter your name: Subramani
Enter your password: Pass@123
✅ Login successful! Welcome, Subramani.
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 3.py"
Enter your name: subramani
Enter your password: sdfadfads
❌ Invalid credentials. Please try again.
```

4. Implement Courier Assignment Logic 1. Develop a mechanism to assign couriers to shipments based on predefined criteria (e.g., proximity, load capacity) using loops.

```
import mysql.connector

# Establish database connection
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="Vineeth1246@",
        database="courier_db"
    )

# Function to assign a courier to a shipment
def assign_courier(shipment_id):
    conn = get_db_connection()
    cursor = conn.cursor()

    # Fetch shipment details (location and weight)
    cursor.execute("SELECT location_id, weight FROM shipment WHERE shipment_id = %s", (ship
    shipment = cursor.fetchone()

    if not shipment:
        print(" Shipment not found.")
        return

    shipment_location, shipment_weight = shipment

    # Fetch a suitable courier (fixing query)
    query = """
        SELECT courier_id, weight
        FROM courier
        WHERE weight >= %s
        ORDER BY weight ASC
        LIMIT 1;
    """
    cursor.execute(query, (shipment_weight,))
    assigned_courier = cursor.fetchone()

    if assigned_courier:
        courier_id, courier_weight = assigned_courier
        print(f" Assigned Courier: {courier_id} (Max Weight: {courier_weight})")

        # Update shipment with assigned courier
        cursor.execute("UPDATE shipment SET courier_id = %s WHERE shipment_id = %s", (couri
        conn.commit()
    else:
        print(" No suitable courier available.")

    cursor.close()
    conn.close()

# Example usage
if __name__ == "__main__":
    shipment_id = int(input("Enter Shipment ID: "))
    assign_courier(shipment_id)
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 4.py"
Enter Shipment ID: 2
✅ Assigned Courier: 4 (Max Weight: 2.0)

Process finished with exit code 0
```

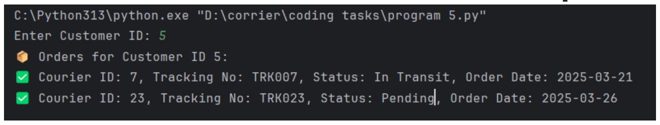
TASK 2

Loops and Iteration

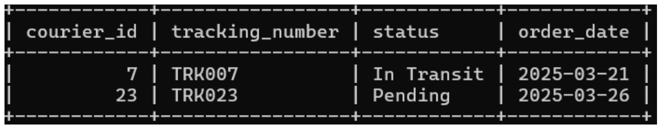
5. Write a python program that uses a for loop to display all the orders for a specific customer.

```
import mysql.connector

# Establish database connection
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="Vineeth1246@",
        database="courier_db"
    )

# Function to display all orders for a specific customer
def display_orders(customer_id):
    conn = get_db_connection()
    cursor = conn.cursor()

    query = """
        SELECT courier_id, tracking_number, status, order_date
        FROM courier
        WHERE customer_id = %s
    """
    cursor.execute(query, (customer_id,))
    orders = cursor.fetchall()

    if not orders:
        print(" No orders found for this customer.")
    else:
        print(f" Orders for Customer ID {customer_id}:")
        for order in orders:
            print(f"Courier ID: {order[0]}, Tracking No: {order[1]}, Status: {order[2]}, Or

    cursor.close()
    conn.close()

# Example usage
if __name__ == "__main__":
    customer_id = int(input("Enter Customer ID: "))
    display_orders(customer_id)
```

output / database



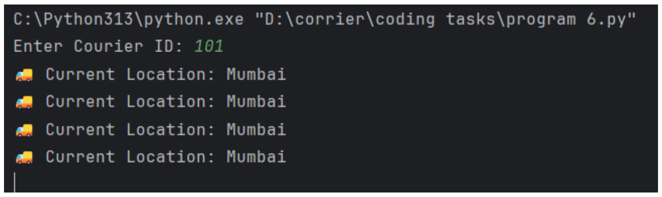
```
C:\Python313\python.exe "D:\corrier\coding tasks\program 5.py"
Enter Customer ID: 5
📦 Orders for Customer ID 5:
✅ Courier ID: 7, Tracking No: TRK007, Status: In Transit, Order Date: 2025-03-21
✅ Courier ID: 23, Tracking No: TRK023, Status: Pending, Order Date: 2025-03-26
```



```
+------------+-----------------+------------+------------+
| courier_id | tracking_number | status     | order_date |
+------------+-----------------+------------+------------+
|          7 | TRK007          | In Transit | 2025-03-21 |
|         23 | TRK023          | Pending    | 2025-03-26 |
+------------+-----------------+------------+------------+
```

6. Implement a while loop to track the real-time location of a courier until it reaches its destination.

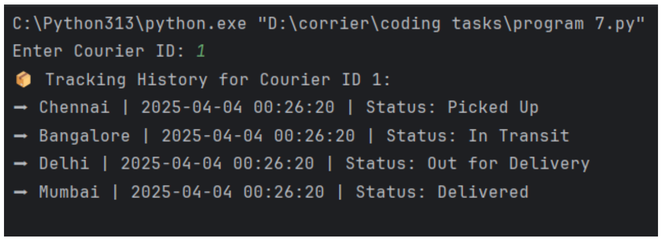
```
import mysql.connector
import time


# Establish database connection
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="Vineeth1246@",
        database="courier_db"
    )


# Function to track courier location in real-time
def track_courier(courier_id):
    conn = get_db_connection()
    cursor = conn.cursor()

    while True:
        # Fetch current location of the courier
        cursor.execute("""
            SELECT location_name FROM location
            WHERE location_id = (SELECT location_id FROM courier WHERE courier_id = %s)
        """, (courier_id,))

        location = cursor.fetchone()

        if not location:
            print(" Courier not found.")
            break

        print(f"Current Location: {location[0]}")

        # Check if courier has reached its destination
        cursor.execute("""
            SELECT status FROM courier WHERE courier_id = %s
        """, (courier_id,))
        status = cursor.fetchone()

        if status and status[0].lower() == "delivered":
            print(" Courier has reached its destination!")
            break

        time.sleep(5)  # Simulate real-time tracking (refreshes every 5 seconds)

    cursor.close()
    conn.close()


# Example usage
if __name__ == "__main__":
    courier_id = int(input("Enter Courier ID: "))
    track_courier(courier_id)
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 6.py"
Enter Courier ID: 101
🚚 Current Location: Mumbai
🚚 Current Location: Mumbai
🚚 Current Location: Mumbai
🚚 Current Location: Mumbai
```

7. Create an array to store the tracking history of a parcel, where each entry represents a location update.

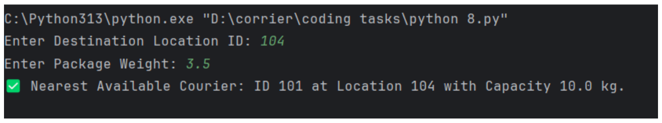
```
import mysql.connector

# Establish database connection
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="Vineeth1246@",
        database="courier_db"
    )

# Function to fetch tracking history
def track_courier(courier_id):
    conn = get_db_connection()
    cursor = conn.cursor()

    query = """
    SELECT t.tracking_id, c.courier_id, l.location_name, t.update_time, t.status
    FROM tracking t
    JOIN location l ON t.location_id = l.location_id
    JOIN courier c ON t.courier_id = c.courier_id
    WHERE c.courier_id = %s
    ORDER BY t.update_time ASC;
    """
    cursor.execute(query, (courier_id,))
    tracking_data = cursor.fetchall()

    if tracking_data:
        print(f" Tracking History for Courier ID {courier_id}:")
        for row in tracking_data:
            print(f"➡ {row[2]} | {row[3]} | Status: {row[4]}")
    else:
        print(" No tracking history found.")

    cursor.close()
    conn.close()

# Example usage
if __name__ == "__main__":
    courier_id = input("Enter Courier ID: ")
    track_courier(courier_id)
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 7.py"
Enter Courier ID: 1
📦 Tracking History for Courier ID 1:
➡ Chennai | 2025-04-04 00:26:20 | Status: Picked Up
➡ Bangalore | 2025-04-04 00:26:20 | Status: In Transit
➡ Delhi | 2025-04-04 00:26:20 | Status: Out for Delivery
➡ Mumbai | 2025-04-04 00:26:20 | Status: Delivered
```

8. Implement a method to find the nearest available courier for a new order using an array of couriers.

```
import mysql.connector


# Establish database connection
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="Vineeth1246@",
        database="courier_db"
    )


# Function to find the nearest available courier
def find_nearest_courier(destination_location, weight):
    conn = get_db_connection()
    cursor = conn.cursor()

    # Fetch couriers who are at the nearest location and have enough capacity
    query = """
    SELECT courier_id, location_id, weight
    FROM courier
    WHERE location_id = %s AND weight >= %s
    ORDER BY weight ASC
    LIMIT 1;
    """

    cursor.execute(query, (destination_location, weight))
    courier = cursor.fetchone()

    if courier:
        courier_id, location_id, courier_weight = courier
        print(
            f" Nearest Available Courier: ID {courier_id} at Location {location_id} with Ca
    else:
        print(" No suitable courier available.")

    cursor.close()
    conn.close()


# Example usage
if __name__ == "__main__":
    destination_location = input("Enter Destination Location ID: ")
    weight = float(input("Enter Package Weight: "))
    find_nearest_courier(destination_location, weight)
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\python 8.py"
Enter Destination Location ID: 104
Enter Package Weight: 3.5
✅ Nearest Available Courier: ID 101 at Location 104 with Capacity 10.0 kg.
```

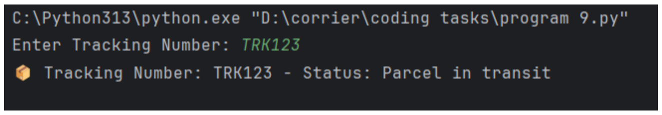
TASK 4

Strings,2d Arrays, user defined functions,Hashmap

9. *Parcel Tracking* Create a program that allows users to input a parcel tracking number.Store the tracking number and Status in 2d String Array. Initialize the array with values. Then, simulate the

tracking process by displaying messages like "Parcel in transit," "Parcel out for delivery," or "Parcel delivered" based on the tracking number's status.

```
# Initialize a 2D array with tracking numbers and statuses
parcel_tracking_data = [
    ["TRK123", "Parcel in transit"],
    ["TRK456", "Parcel out for delivery"],
    ["TRK789", "Parcel delivered"],
    ["TRK101", "Parcel in transit"],
    ["TRK202", "Parcel out for delivery"],
    ["TRK303", "Parcel delivered"]
]

# Function to track parcel status
def track_parcel(tracking_number):
    for parcel in parcel_tracking_data:
        if parcel[0] == tracking_number:
            print(f" Tracking Number: {tracking_number} - Status: {parcel[1]}")
            return
    print(" Tracking number not found.")

# Get user input
tracking_number = input("Enter Tracking Number: ").strip().upper()
track_parcel(tracking_number)
```

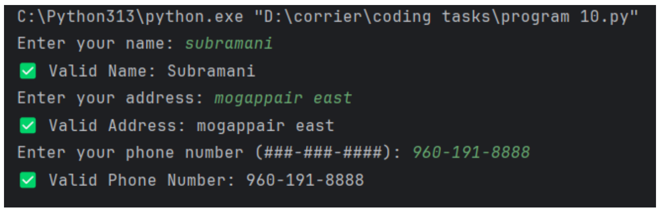


```
C:\Python313\python.exe "D:\corrier\coding tasks\program 9.py"
Enter Tracking Number: TRK123
📦 Tracking Number: TRK123 - Status: Parcel in transit
```

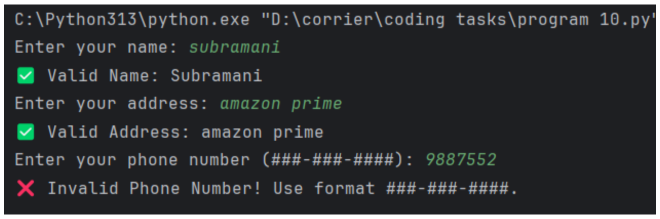
10. *Customer Data Validation*: Write a function which takes 2 parameters, data-denotes the data and detail-denotes if it is name addtress or phone number.Validate customer information based on following critirea. Ensure that names contain only letters and are properly capitalized, addresses do not contain special characters, and phone numbers follow a specific format (e.g., ###-###-####).

```
import re

# Function to validate customer data
def validate_customer_data(data, detail):
    if detail == "name":
        if re.fullmatch(r"[A-Za-z ]+", data):
            return f"Valid Name: {data.title()}"
        else:
            return " Invalid Name! Only letters and spaces are allowed."

    elif detail == "address":
        if re.fullmatch(r"[A-Za-z0-9 ,.-]+", data):
            return f"Valid Address: {data}"
        else:
            return " Invalid Address! Special characters are not allowed."

    elif detail == "phone":
        if re.fullmatch(r"\d{3}-\d{3}-\d{4}", data):
            return f" Valid Phone Number: {data}"
        else:
            return " Invalid Phone Number! Use format ###-###-####."

    else:
        return " Invalid detail type! Use 'name', 'address', or 'phone'."

# Get user input and validate
name = input("Enter your name: ")
print(validate_customer_data(name, "name"))

address = input("Enter your address: ")
print(validate_customer_data(address, "address"))

phone = input("Enter your phone number (###-###-####): ")
print(validate_customer_data(phone, "phone"))
```

valid / invalid format



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 10.py"
Enter your name: subramani
✅ Valid Name: Subramani
Enter your address: mogappair east
✅ Valid Address: mogappair east
Enter your phone number (###-###-####): 960-191-8888
✅ Valid Phone Number: 960-191-8888
```



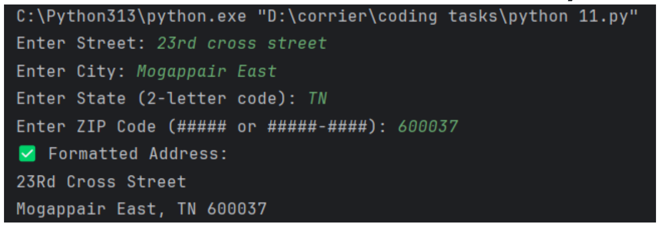
```
C:\Python313\python.exe "D:\corrier\coding tasks\program 10.py"
Enter your name: subramani
✅ Valid Name: Subramani
Enter your address: amazon prime
✅ Valid Address: amazon prime
Enter your phone number (###-###-####): 9887552
❌ Invalid Phone Number! Use format ###-###-####.
```

11. *Address Formatting*: Develop a function that takes an address as input (street, city, state, zip code) and formats it correctly, including capitalizing the first letter of each word and properly formatting the zip code.

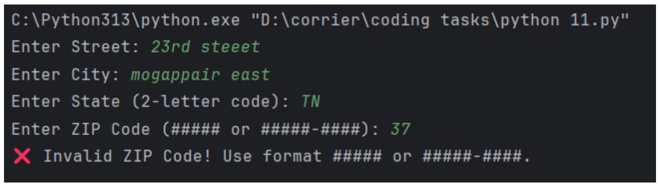
```
import re

# Function to format address
def format_address(street, city, state, zip_code):
    if not re.fullmatch(r"\d{6}(-\d{4})?", zip_code):  # Validates ZIP (##### or #####-####
        return " Invalid ZIP Code! Use format ##### or #####-####."

    formatted_street = street.title()  # Capitalizes first letter of each word
    formatted_city = city.title()
    formatted_state = state.upper()  # Converts state to uppercase

    return f" Formatted Address:\n{formatted_street}\n{formatted_city}, {formatted_state} {

# Get user input and format address
street = input("Enter Street: ")
city = input("Enter City: ")
state = input("Enter State (2-letter code): ")
zip_code = input("Enter ZIP Code (##### or #####-####): ")
```

```
print(format_address(street, city, state, zip_code))
```

valid / invalid format



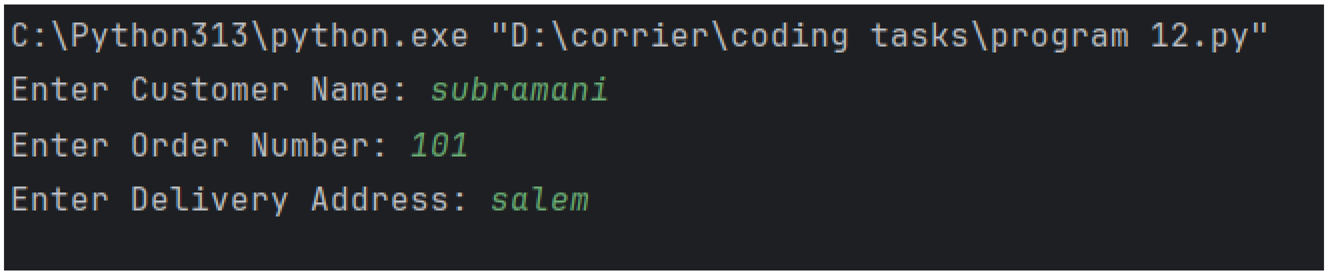
```
C:\Python313\python.exe "D:\corrier\coding tasks\python 11.py"
Enter Street: 23rd cross street
Enter City: Mogappair East
Enter State (2-letter code): TN
Enter ZIP Code (##### or #####-####): 600037
✅ Formatted Address:
23Rd Cross Street
Mogappair East, TN 600037
```



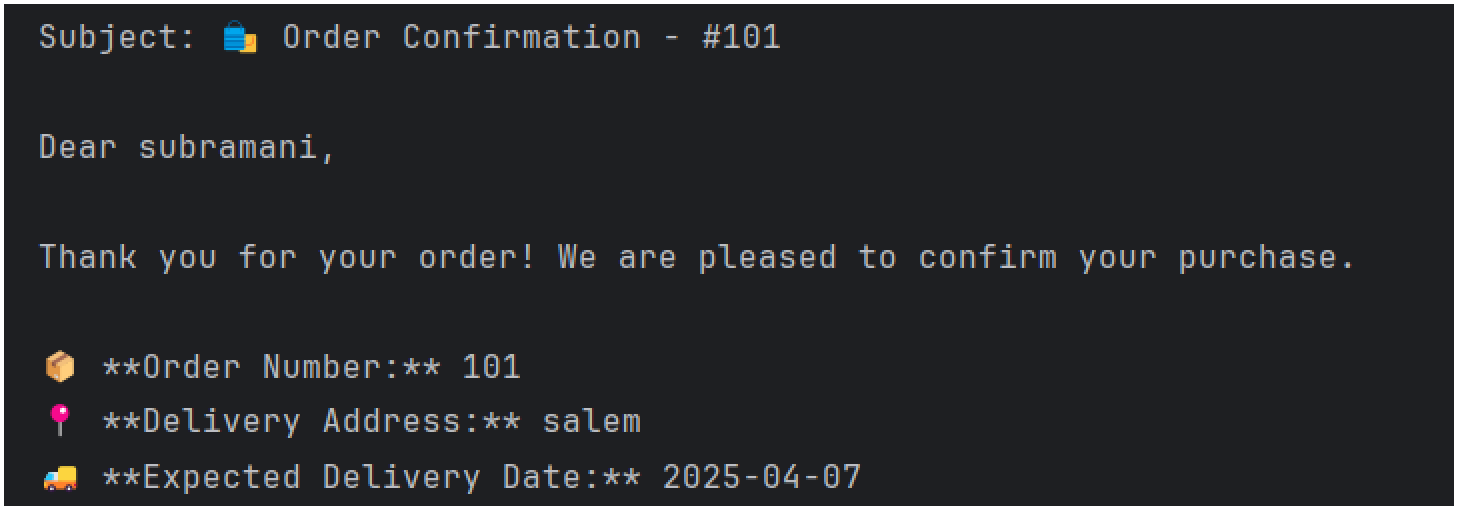
```
C:\Python313\python.exe "D:\corrier\coding tasks\python 11.py"
Enter Street: 23rd steeet
Enter City: mogappair east
Enter State (2-letter code): TN
Enter ZIP Code (##### or #####-####): 37
❌ Invalid ZIP Code! Use format ##### or #####-####.
```

12. *Order Confirmation Email*: Create a program that generates an order confirmation email. The email should include details such as the customer's name, order number, delivery address, and expected delivery date.

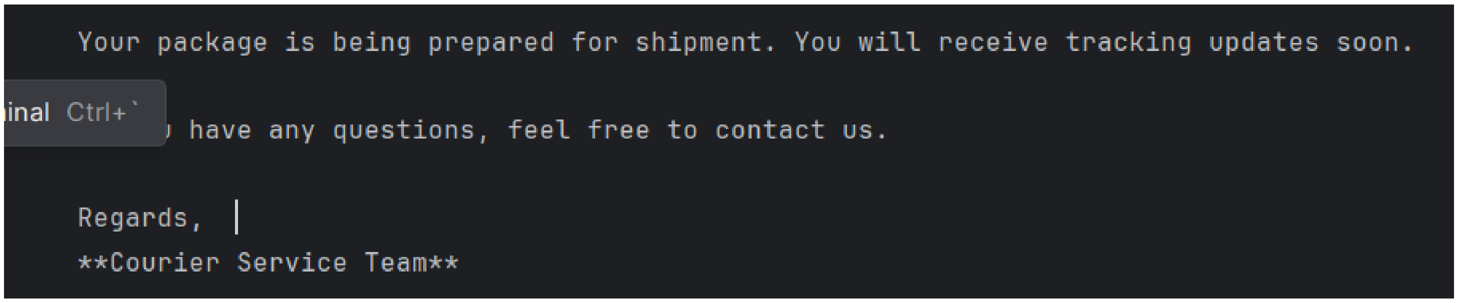
```
import datetime

# Function to generate order confirmation email
def generate_order_email(customer_name, order_number, address, delivery_date):
    email_template = f"""
    Subject:  Order Confirmation - #{order_number}

    Dear {customer_name},

    Thank you for your order! We are pleased to confirm your purchase.

     **Order Number:** {order_number}
     **Delivery Address:** {address}
     **Expected Delivery Date:** {delivery_date}

    Your package is being prepared for shipment. You will receive tracking updates soon.

    If you have any questions, feel free to contact us.

    Regards,
    **Courier Service Team**
    """
    return email_template

# Get user input
customer_name = input("Enter Customer Name: ")
order_number = input("Enter Order Number: ")
address = input("Enter Delivery Address: ")

# Set expected delivery date (3 days from today)
delivery_date = (datetime.date.today() + datetime.timedelta(days=3)).strftime("%Y-%m-%d")

# Generate and display email
email_content = generate_order_email(customer_name, order_number, address, delivery_date)
print("\n" + email_content)
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 12.py"
Enter Customer Name: subramani
Enter Order Number: 101
Enter Delivery Address: salem
```



```
Subject: 🛍️ Order Confirmation - #101

Dear subramani,

Thank you for your order! We are pleased to confirm your purchase.

📦 **Order Number:** 101
📍 **Delivery Address:** salem
🚚 **Expected Delivery Date:** 2025-04-07
```

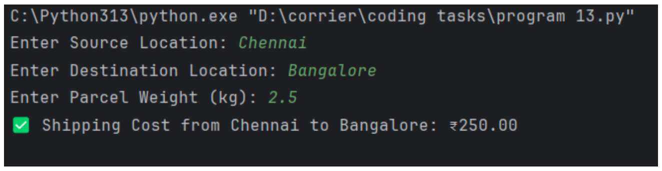


```
  Your package is being prepared for shipment. You will receive tracking updates soon.

inal  Ctrl+`  have any questions, feel free to contact us.

  Regards,
  **Courier Service Team**
```

13. *Calculate Shipping Costs* : Develop a function that calculates the shipping cost based on the distance between two locations and the weight of the parcel. You can use string inputs for the source and destination addresses.

```
# Sample distance dictionary (mock data)
distance_data = {
    ("Chennai", "Bangalore"): 350,
    ("Chennai", "Delhi"): 2200,
    ("Chennai", "Mumbai"): 1330,
    ("Bangalore", "Mumbai"): 980,
    ("Mumbai", "Delhi"): 1440,
    ("Delhi", "Kolkata"): 1530,
    ("Hyderabad", "Chennai"): 630,
}

# Shipping cost rates
BASE_COST = 50  # Base charge
COST_PER_KM = 0.5  # Cost per km
COST_PER_KG = 10  # Cost per kg


# Function to calculate shipping cost
```

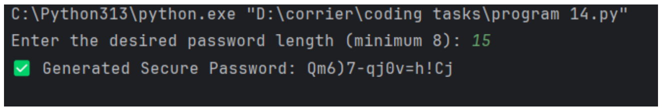
```
def calculate_shipping_cost(source, destination, weight):
    # Convert input to title case to match dictionary keys
    source = source.title()
    destination = destination.title()

    # Check if distance exists in mock data
    if (source, destination) in distance_data:
        distance = distance_data[(source, destination)]
    elif (destination, source) in distance_data:  # Reverse lookup
        distance = distance_data[(destination, source)]
    else:
        print(" Shipping route not found in database.")
        return

    # Calculate cost
    shipping_cost = BASE_COST + (distance * COST_PER_KM) + (weight * COST_PER_KG)

    # Display result
    print(f" Shipping Cost from {source} to {destination}: ₹{shipping_cost:.2f}")


# Get user input
source = input("Enter Source Location: ")
destination = input("Enter Destination Location: ")
weight = float(input("Enter Parcel Weight (kg): "))

# Call function
calculate_shipping_cost(source, destination, weight)
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 13.py"
Enter Source Location: Chennai
Enter Destination Location: Bangalore
Enter Parcel Weight (kg): 2.5
✅ Shipping Cost from Chennai to Bangalore: ₹250.00
```

14. *Password Generator*: Create a function that generates secure passwords for courier system accounts. Ensure the passwords contain a mix of uppercase letters, lowercase letters, numbers, and special characters.

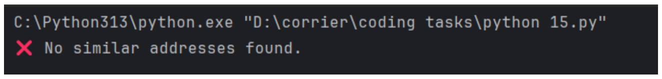
```
import random
import string

# Function to generate a secure password
def generate_password(length=12):
    if length < 8:
        print(" Password length must be at least 8 characters.")
        return None

    # Define character sets
    upper = string.ascii_uppercase  # A-Z
    lower = string.ascii_lowercase  # a-z
    digits = string.digits  # 0-9
    special = "!@#$%^&*()-_=+"

    # Ensure at least one character from each category
    password = [
        random.choice(upper),
        random.choice(lower),
        random.choice(digits),
        random.choice(special),
    ]

    # Fill the remaining length with random choices from all sets
    all_chars = upper + lower + digits + special
    password += random.choices(all_chars, k=length - 4)

    # Shuffle the password to ensure randomness
    random.shuffle(password)

    # Convert list to string and return
    return "".join(password)

# Get user input for password length
length = int(input("Enter the desired password length (minimum 8): "))

# Generate and display the password
secure_password = generate_password(length)
if secure_password:
    print(f" Generated Secure Password: {secure_password}")
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\program 14.py"
Enter the desired password length (minimum 8): 15
✅ Generated Secure Password: Qm6)7-qj0v=h!Cj
```

15. *Find Similar Addresses* : Implement a function that finds similar addresses in the system. This can be useful for identifying duplicate customer entries or optimizing delivery routes.Use string functions to implement this.

```
import mysql.connector
from difflib import SequenceMatcher

# Function to establish database connection
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="your_password",
        database="courier_db"
    )

# Function to calculate similarity between two strings
def address_similarity(addr1, addr2):
    return SequenceMatcher(None, addr1, addr2).ratio()

# Function to find similar addresses
def find_similar_addresses(threshold=0.8):
    conn = get_db_connection()
    cursor = conn.cursor()

    # Fetch all addresses from the location table
    cursor.execute("SELECT location_id, address FROM location")
    addresses = cursor.fetchall()

    similar_pairs = []

    # Compare each address with others
    for i in range(len(addresses)):
        for j in range(i + 1, len(addresses)):
            addr1_id, addr1 = addresses[i]
            addr2_id, addr2 = addresses[j]

            similarity = address_similarity(addr1, addr2)

            if similarity >= threshold:
                similar_pairs.append((addr1_id, addr1, addr2_id, addr2, round(similarity, 2

    cursor.close()
    conn.close()

    # Display results
    if similar_pairs:
        print(" Similar Addresses Found:")
        for pair in similar_pairs:
            print(f" Address {pair[0]}: {pair[1]}")
            print(f" Address {pair[2]}: {pair[3]}")
            print(f" Similarity Score: {pair[4]}\n")
    else:
        print(" No similar addresses found.")

# Run the function
find_similar_addresses()
```



```
C:\Python313\python.exe "D:\corrier\coding tasks\python 15.py"
❌ No similar addresses found.
```

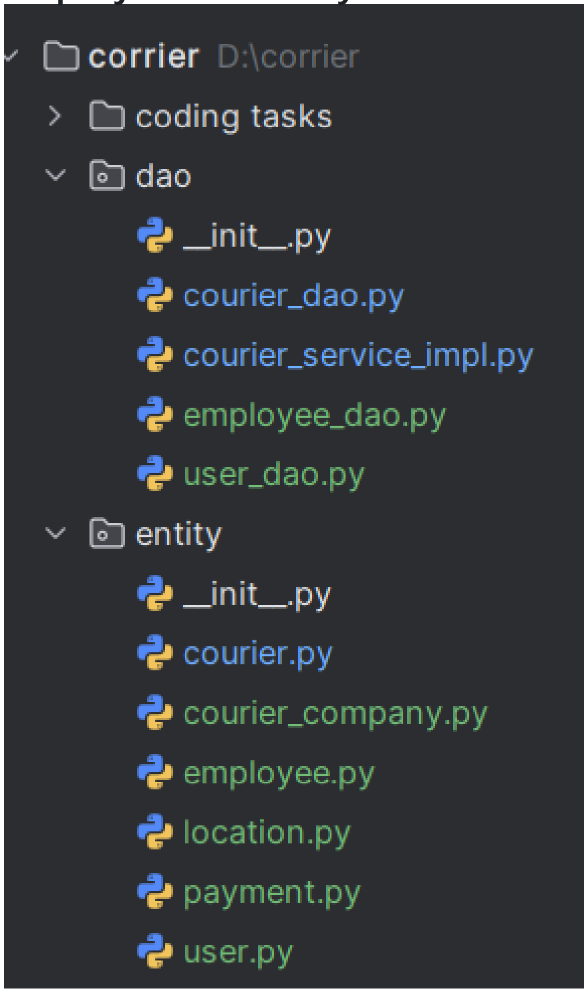
TASK 5

Object Oriented Programming

This module focuses on implementing object-oriented programming (OOP) concepts for a **Courier Management System**.

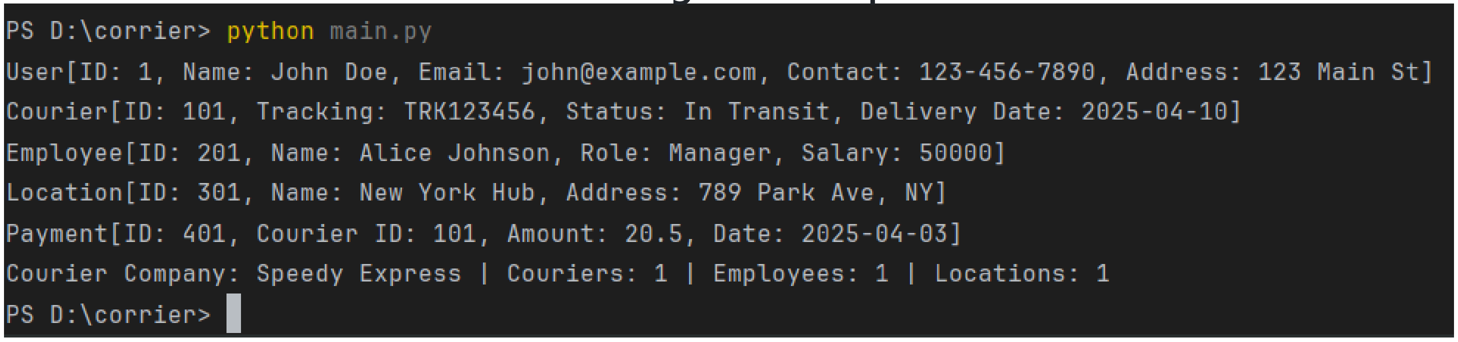
It includes entity classes with proper encapsulation, constructors, getters, setters, and collections for managing couriers, employees, and locations.

project directory structure



```
corrier D:\corrier
  coding tasks
  dao
    __init__.py
    courier_dao.py
    courier_service_impl.py
    employee_dao.py
    user_dao.py
  entity
    __init__.py
    courier.py
    courier_company.py
    employee.py
    location.py
    payment.py
    user.py
```

running main script



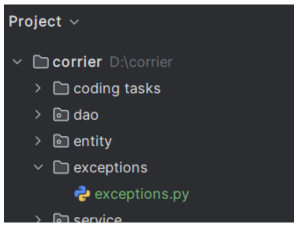
```
PS D:\corrier> python main.py
User[ID: 1, Name: John Doe, Email: john@example.com, Contact: 123-456-7890, Address: 123 Main St]
Courier[ID: 101, Tracking: TRK123456, Status: In Transit, Delivery Date: 2025-04-10]
Employee[ID: 201, Name: Alice Johnson, Role: Manager, Salary: 50000]
Location[ID: 301, Name: New York Hub, Address: 789 Park Ave, NY]
Payment[ID: 401, Courier ID: 101, Amount: 20.5, Date: 2025-04-03]
Courier Company: Speedy Express | Couriers: 1 | Employees: 1 | Locations: 1
PS D:\corrier>
```

TASK 7

Exception Handling

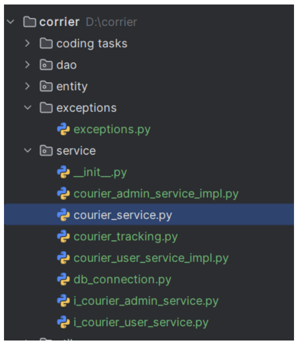
(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,thow & throws keyword usage)
Step-by-Step Implementation:
Create Custom Exceptions:
TrackingNumberNotFoundException to be thrown when the tracking number is not found.
InvalidEmployeeIdException to be thrown when the employee ID doesn't exist.
Throwing Custom Exceptions:
We will throw these exceptions in the methods based on specific conditions.

Handling the Exceptions:
The exceptions will be caught and handled in the main() method using try...catch blocks.
Step 1: Create the Exception Class (exceptions/exceptions.py)
Created a folder named exceptions/
Inside the exceptions/ folder, created a file named exceptions.py and place the custom exception classes in this file.



```
# exceptions/exceptions.py

class TrackingNumberNotFoundException(Exception):
    """Exception raised when tracking number is not found."""
    def __init__(self, message="Tracking number not found."):
        self.message = message
        super().__init__(self.message)


class InvalidEmployeeIdException(Exception):
    """Exception raised when the employee ID is invalid."""
    def __init__(self, message="Invalid employee ID entered."):
        self.message = message
        super().__init__(self.message)
```

Step 2: Create the Courier Service Logic (service/courier_service.py)
Created a folder named service/
Inside the service/ folder, created a file named courier_service.py.
In this file, you can define the methods for transferring and withdrawing amounts, validating employee IDs, and raising exceptions.

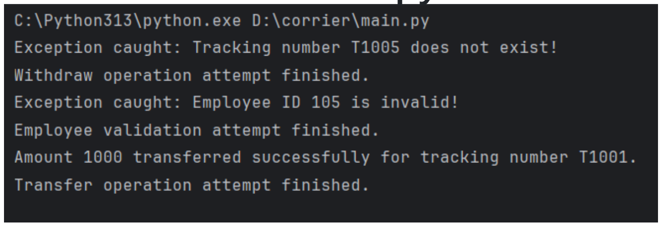


```
# service/courier_service.py
from exceptions.exceptions import TrackingNumberNotFoundException, InvalidEmployeeIdExcepti

class CourierService:

    # Simulating a database of tracking numbers
    tracking_numbers_db = ["T1001", "T1002", "T1003"]

    # Simulating a database of employee IDs
    employee_ids_db = [101, 102, 103]

    def transfer_amount(self, tracking_number, amount):
        """Method to transfer amount; throws exception if tracking number not found."""
        if tracking_number not in self.tracking_numbers_db:
            raise TrackingNumberNotFoundException(f"Tracking number {tracking_number} does
        else:
            print(f"Amount {amount} transferred successfully for tracking number {tracking_

    def withdraw_amount(self, tracking_number, amount):
        """Method to withdraw amount; throws exception if tracking number not found."""
        if tracking_number not in self.tracking_numbers_db:
            raise TrackingNumberNotFoundException(f"Tracking number {tracking_number} does
        else:
            print(f"Amount {amount} withdrawn successfully for tracking number {tracking_nu

    def validate_employee(self, employee_id):
        """Method to validate employee ID; throws exception if ID not found."""
        if employee_id not in self.employee_ids_db:
            raise InvalidEmployeeIdException(f"Employee ID {employee_id} is invalid!")
        else:
            print(f"Employee ID {employee_id} is valid.")
```

Step 3: Main Program (main.py)
In the main.py file, you will import the necessary service and exception classes, and call the service methods while handling the exceptions.

```
# main.py
from service.courier_service import CourierService
from exceptions.exceptions import TrackingNumberNotFoundException, InvalidEmployeeIdExcepti

def main():
    courier_service = CourierService()

    # Test cases to handle the exceptions

    # Handling TrackingNumberNotFoundException
    try:
        courier_service.withdraw_amount("T1005", 500)
    except TrackingNumberNotFoundException as e:
        print(f"Exception caught: {e}")
    finally:
        print("Withdraw operation attempt finished.")

    # Handling InvalidEmployeeIdException
    try:
        courier_service.validate_employee(105)
    except InvalidEmployeeIdException as e:
        print(f"Exception caught: {e}")
    finally:
        print("Employee validation attempt finished.")

    # Valid case (no exception thrown)
    try:
        courier_service.transfer_amount("T1001", 1000)
    except TrackingNumberNotFoundException as e:
        print(f"Exception caught: {e}")
    finally:
        print("Transfer operation attempt finished.")

# Execute main method
if __name__ == "__main__":
    main()
```

run main.py



```
C:\Python313\python.exe D:\corrier\main.py
Exception caught: Tracking number T1005 does not exist!
Withdraw operation attempt finished.
Exception caught: Employee ID 105 is invalid!
Employee validation attempt finished.
Amount 1000 transferred successfully for tracking number T1001.
Transfer operation attempt finished.
```

Conclusion:

exceptions/exceptions.py contains your custom exception classes.
service/courier_service.py contains the business logic for your courier operations.
main.py serves as the entry point of your application, where you handle exceptions in the main program flow.

TASK 8

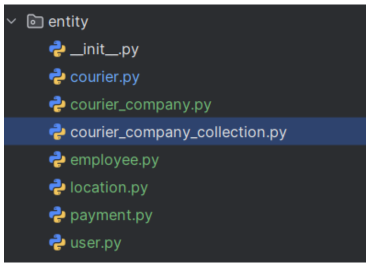
ArrayList/Hashmap

Task: Improve the Courier Management System by using Java collections

1. Create a new model named CourierCompanyCollection in entity package replacing the Array of Objects with List to accommodate dynamic updates in the CourierCompany class

created a new model named CourierCompanyCollection in entity package

```
# entity/courier_company_collection.py

class CourierCompanyCollection:
    def __init__(self):
        # Replace the array with a list to store couriers dynamically
        self.couriers = []

    def add_courier(self, courier):
        """Add a new courier to the collection."""
        self.couriers.append(courier)

    def remove_courier(self, tracking_number):
        """Remove a courier by tracking number."""
        self.couriers = [courier for courier in self.couriers if courier.tracking_number !=

    def get_courier(self, tracking_number):
        """Get a courier by tracking number."""
        for courier in self.couriers:
            if courier.tracking_number == tracking_number:
                return courier
        return None

    def list_all_couriers(self):
        """List all couriers in the collection."""
        return self.couriers
```

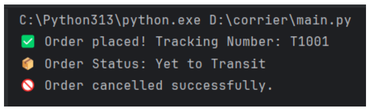


main.py

```
# main.py
from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
from entity.courier import Courier

# Create an instance of the service implementation
user_service = CourierUserServiceCollectionImpl()

# Place a new order
courier = Courier(
    sender_name="John Doe", sender_address="123 Street, NY",
    receiver_name="Alice Smith", receiver_address="456 Avenue, CA",
    weight=5.5, status="Yet to Transit", tracking_number="T1001", delivery_date=None, custo
)

tracking_number = user_service.place_order(courier)
print(f" Order placed! Tracking Number: {tracking_number}")

# Check order status
print(" Order Status:", user_service.get_order_status(tracking_number))

# Cancel an order
if user_service.cancel_order(tracking_number):
    print(" Order cancelled successfully.")
else:
    print(" Order cannot be cancelled.")
```

implementation:

1. Placing an Order: When a new Courier object is created and the place_order method is called, the courier is added to the CourierCompanyCollection list, and a tracking number is generated and returned.
2. Checking the Order Status: The get_order_status method is called with the tracking number, and it retrieves the courier's status, which will be "Yet to Transit" since that's the default status for new orders.
3. Cancelling an Order: The cancel_order method is called with the tracking number. The courier is removed from the collection if it exists, and True is returned indicating successful cancellation. If the courier doesn't exist, False is returned.

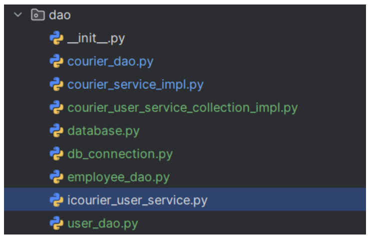


```
C:\Python313\python.exe D:\corrier\main.py
✅ Order placed! Tracking Number: T1001
📦 Order Status: Yet to Transit
🚫 Order cancelled successfully.
```

2. Create a new implementation class CourierUserServiceCollectionImpl class in package dao which implements ICourierUserService interface which holds a variable named companyObj of type CourierCompanyCollection

1. Create the Interface (ICourierUserService)

```
# dao/icourier_user_service.py
from abc import ABC, abstractmethod

class ICourierUserService(ABC):

    @abstractmethod
    def place_order(self, courier):
        pass

    @abstractmethod
    def get_order_status(self, tracking_number):
        pass

    @abstractmethod
    def cancel_order(self, tracking_number):
        pass
```



```
dao
    __init__.py
    courier_dao.py
    courier_service_impl.py
    courier_user_service_collection_impl.py
    database.py
    db_connection.py
    employee_dao.py
    icourier_user_service.py
    user_dao.py
```

2. Create the CourierCompanyCollection Class

This class manages a collection of Courier objects, allowing dynamic updates to the collection.

```
# entity/courier_company_collection.py

class CourierCompanyCollection:
    def __init__(self):
        self.couriers = []  # List to hold courier objects

    def add_courier(self, courier):
        """Add a new courier to the collection."""
        self.couriers.append(courier)

    def remove_courier(self, tracking_number):
        """Remove a courier by tracking number."""
        self.couriers = [courier for courier in self.couriers if courier.tracking_number !=

    def get_courier(self, tracking_number):
        """Get a courier by tracking number."""
        for courier in self.couriers:
            if courier.tracking_number == tracking_number:
                return courier
        return None

    def list_all_couriers(self):
        """List all couriers in the collection."""
        return self.couriers
```

3. Implement the CourierUserServiceCollectionImpl Class

This class will implement the ICourierUserService interface and will use CourierCompanyCollection for storing and managing courier data.

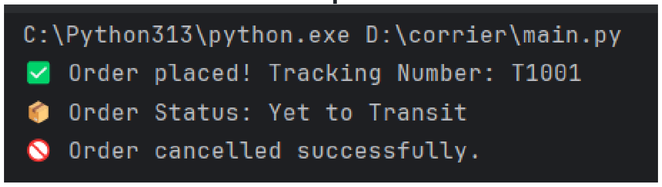
```
# dao/courier_user_service_collection_impl.py
from entity.courier_company_collection import CourierCompanyCollection
from dao.icourier_user_service import ICourierUserService
from entity.courier import Courier

class CourierUserServiceCollectionImpl(ICourierUserService):
    def __init__(self):
        self.company_obj = CourierCompanyCollection()  # Holds the collection of couriers

    def place_order(self, courier):
        """Place a new courier order and add it to the collection."""
        self.company_obj.add_courier(courier)
        return courier.tracking_number

    def get_order_status(self, tracking_number):
        """Get the status of a courier by tracking number."""
        courier = self.company_obj.get_courier(tracking_number)
        if courier:
            return courier.status
        return "Courier not found."

    def cancel_order(self, tracking_number):
        """Cancel a courier order by removing it from the collection."""
        courier = self.company_obj.get_courier(tracking_number)
        if courier:
            self.company_obj.remove_courier(tracking_number)
            return True
        return False
```

4. Using the Classes in the Main Program

In your main.py, you will create an instance of CourierUserServiceCollectionImpl to interact with the collection of couriers.

```
# main.py
from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
from entity.courier import Courier

# Create an instance of the service implementation
user_service = CourierUserServiceCollectionImpl()

# Place a new order
courier = Courier(
    sender_name="John Doe", sender_address="123 Street, NY",
    receiver_name="Alice Smith", receiver_address="456 Avenue, CA",
    weight=5.5, status="Yet to Transit", tracking_number="T1001", delivery_date=None, custo
)

tracking_number = user_service.place_order(courier)
print(f" Order placed! Tracking Number: {tracking_number}")

# Check order status
print(" Order Status:", user_service.get_order_status(tracking_number))

# Cancel an order
if user_service.cancel_order(tracking_number):
    print(" Order cancelled successfully.")
else:
    print(" Order cannot be cancelled.")
```

output



```
C:\Python313\python.exe D:\corrier\main.py
✅ Order placed! Tracking Number: T1001
📦 Order Status: Yet to Transit
🚫 Order cancelled successfully.
```

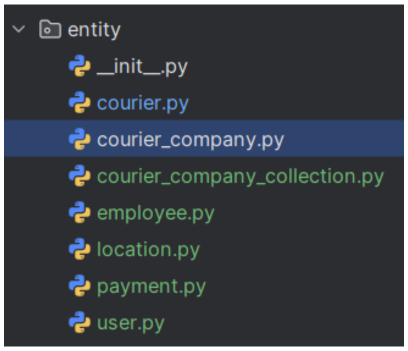
TASK 8

Service implementation

1.Create *CourierUserServiceImpl* class which implements *ICourierUserService* interface which holds a variable named *companyObj* of type *CourierCompany.* This variable can be used to access the Object Arrays to access data relevant in method implementations.

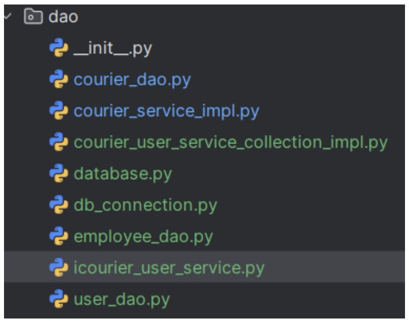
we have already created CourierCompany Class , ICourierUserService Interface , CourierUserServiceImpl Class , Courier Class for the previous tasks now modifying main class according to the given question

Image Gallery



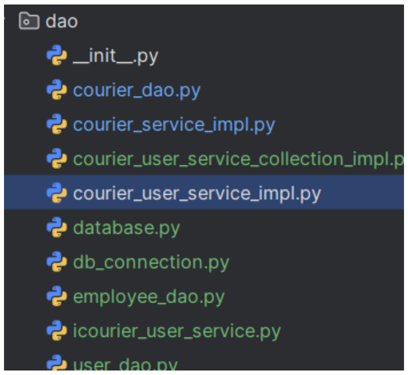
```
entity
    __init__.py
    courier.py
    courier_company.py
    courier_company_collection.py
    employee.py
    location.py
    payment.py
    user.py
```

CourierCompany Class



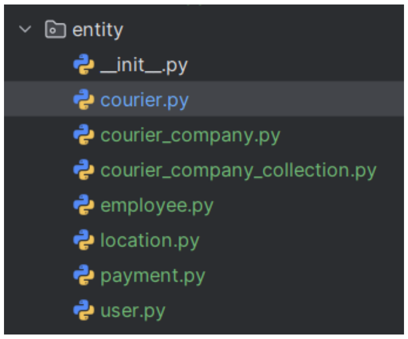
```
dao
    __init__.py
    courier_dao.py
    courier_service_impl.py
    courier_user_service_collection_impl.py
    database.py
    db_connection.py
    employee_dao.py
    icourier_user_service.py
    user_dao.py
```

ICourierUserService Interface



```
dao
    __init__.py
    courier_dao.py
    courier_service_impl.py
    courier_user_service_collection_impl.p
    courier_user_service_impl.py
    database.py
    db_connection.py
    employee_dao.py
    icourier_user_service.py
    user_dao.py
```

CourierUserServiceImpl Class



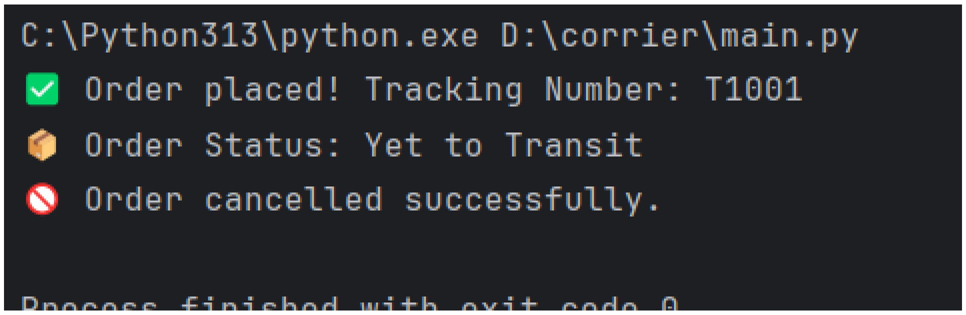
```
entity
    __init__.py
    courier.py
    courier_company.py
    courier_company_collection.py
    employee.py
    location.py
    payment.py
    user.py
```

Courier Class

main.py

```
# main.py
from dao.courier_user_service_impl import CourierUserServiceImpl
from entity.courier import Courier

# Create an instance of the service implementation
user_service = CourierUserServiceImpl()

# Place a new order
courier = Courier(
    sender_name="John Doe", sender_address="123 Street, NY",
    receiver_name="Alice Smith", receiver_address="456 Avenue, CA",
    weight=5.5, status="Yet to Transit", tracking_number="T1001", delivery_date=None, custo
)

tracking_number = user_service.place_order(courier)
print(f" Order placed! Tracking Number: {tracking_number}")

# Check order status
print(" Order Status:", user_service.get_order_status(tracking_number))

# Cancel an order
if user_service.cancel_order(tracking_number):
    print(" Order cancelled successfully.")
```

```
else:
    print(" Order cannot be cancelled.")
```



```
C:\Python313\python.exe D:\corrier\main.py
✅ Order placed! Tracking Number: T1001
📦 Order Status: Yet to Transit
🚫 Order cancelled successfully.

Process finished with exit code 0
```

TASK 9

Database Interaction

1. Write code to establish a connection to your SQL database.

```
# db_connection.py
import psycopg2  # For PostgreSQL; install this library using pip if not already installed.

def get_db_connection():
    """Establish and return a database connection."""
    try:
        # Replace these values with your actual DB connection details
        connection = psycopg2.connect(
            host="localhost",
            user="root",
            password="Vineeth1246@",
            database="courier_db"
        )
        return connection
    except Exception as e:
        print(f"Error connecting to the database: {e}")
        return None
```

2. Service Class for Database Operations

Create a Service class *CourierServiceDb* in *dao* with a static variable named *connection* of type Connection which can be assigned in the constructor by invoking the method in *DBConnection* Class.

Steps to Implement:

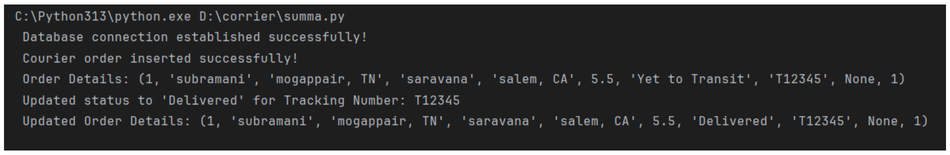
Create a new file named courier_service_db.py inside dao.

code:

```
from connectionutil.db_connection import DBConnection

class CourierServiceDb:
    # Static connection variable
    connection = None

    def __init__(self):
        """Initialize the connection using DBConnection"""
        if CourierServiceDb.connection is None:
            CourierServiceDb.connection = DBConnection.get_connection()

    def insert_courier(self, courier):
        """Insert a new courier order into the database"""
        try:
            cursor = self.connection.cursor()
            query = """
            INSERT INTO courier (sender_name, sender_address, receiver_name, receiver_addre
                                 weight, status, tracking_number, delivery_date, customer_i
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
            """
            values = (
                courier.sender_name, courier.sender_address,
                courier.receiver_name, courier.receiver_address,
                courier.weight, "Yet to Transit", courier.tracking_number,
                courier.delivery_date, courier.customer_id
            )
            cursor.execute(query, values)
            self.connection.commit()
            print(" Courier order inserted successfully!")

        except Exception as e:
            print(f" Error inserting courier: {e}")

        finally:
            cursor.close()

    def update_status(self, tracking_number, new_status):
        """Update the status of a courier"""
        try:
            cursor = self.connection.cursor()
            query = "UPDATE courier SET status = %s WHERE tracking_number = %s"
            cursor.execute(query, (new_status, tracking_number))
            self.connection.commit()
            print(f" Updated status to '{new_status}' for Tracking Number: {tracking_number

        except Exception as e:
            print(f" Error updating status: {e}")

        finally:
            cursor.close()

    def get_courier_by_tracking(self, tracking_number):
        """Retrieve courier details by tracking number"""
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM courier WHERE tracking_number = %s"
            cursor.execute(query, (tracking_number,))
            result = cursor.fetchone()
            return result

        except Exception as e:
            print(f" Error fetching courier: {e}")

        finally:
            cursor.close()
```

Step 2: Create the DBConnection Class
already created in the last tasks
Step 3: Create the CourierServiceDb Class
already created in the last tasks
Step 4: Update main.py

```
from dao.courier_service_db import CourierServiceDb
from entity.courier import Courier

# Initialize the database service
service = CourierServiceDb()

# Create a courier order
courier = Courier(sender_name="John Doe", sender_address="123 Street, NY",
                  receiver_name="Alice Smith", receiver_address="456 Avenue, CA",
                  weight=5.5, status="Yet to Transit", tracking_number="T12345",
                  delivery_date=None, customer_id=1)

# Insert the courier order
service.insert_courier(courier)

# Retrieve the order status
order = service.get_courier_by_tracking("T12345")
print(" Order Details:", order)

# Update order status
service.update_status("T12345", "Delivered")

# Verify update
updated_order = service.get_courier_by_tracking("T12345")
print(" Updated Order Details:", updated_order)
```



```
C:\Python313\python.exe D:\corrier\summa.py
 Database connection established successfully!
 Courier order inserted successfully!
 Order Details: (1, 'subramani', 'mogappair, TN', 'saravana', 'salem, CA', 5.5, 'Yet to Transit', 'T12345', None, 1)
 Updated status to 'Delivered' for Tracking Number: T12345
 Updated Order Details: (1, 'subramani', 'mogappair, TN', 'saravana', 'salem, CA', 5.5, 'Delivered', 'T12345', None, 1)
```

3. Include methods to insert, update, and retrieve data from the database (e.g., inserting a new order, updating courier status).

Steps to Implement

Create a DBConnection class to establish a database connection.

Create a CourierServiceDb class with methods to:

1. Insert a new courier order
2. Update the courier status
3. Retrieve courier details

db_connection.py (Database Connection)

```
import mysql.connector  # Use `import psycopg2` for PostgreSQL

class DBConnection:
    _connection = None

    @staticmethod
    def get_connection():
        if DBConnection._connection is None:
            try:
                DBConnection._connection = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    password="your_password",
                    database="courier_db"
                )
                print(" Database connection established successfully!")
            except Exception as e:
                print(" Error connecting to the database:", e)
        return DBConnection._connection
```

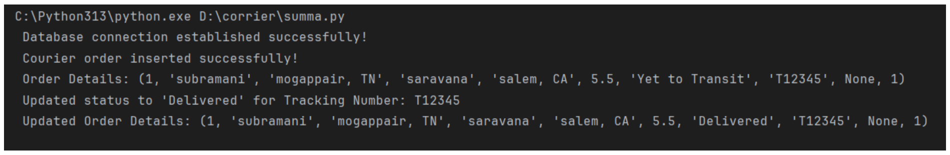
courier_service_db.py (Service Class for Database Operations)

```
from db_connection import DBConnection

class CourierServiceDb:
    def __init__(self):
        self.connection = DBConnection.get_connection()
        self.cursor = self.connection.cursor()

    def insert_order(self, sender_name, sender_address, receiver_name, receiver_address, we
        """Inserts a new courier order into the database."""
        query = """INSERT INTO courier (sender_name, sender_address, receiver_name, receive
                   weight, status, tracking_number, delivery_date, customer_id)
                   VALUES (%s, %s, %s, %s, %s, %s, %s, NULL, %s)"""
        values = (sender_name, sender_address, receiver_name, receiver_address, weight, "Ye

        try:
            self.cursor.execute(query, values)
            self.connection.commit()
            print(" Courier order inserted successfully!")
        except Exception as e:
            print(" Error inserting order:", e)

    def update_status(self, tracking_number, new_status):
        """Updates the status of an existing courier order."""
        query = "UPDATE courier SET status = %s WHERE tracking_number = %s"

        try:
            self.cursor.execute(query, (new_status, tracking_number))
            self.connection.commit()
            print(f" Updated status to '{new_status}' for Tracking Number: {tracking_number
        except Exception as e:
            print(" Error updating status:", e)

    def get_order_details(self, tracking_number):
        """Retrieves and displays details of a specific order."""
        query = "SELECT * FROM courier WHERE tracking_number = %s"

        try:
            self.cursor.execute(query, (tracking_number,))
            order = self.cursor.fetchone()
            if order:
                print(" Order Details:", order)
                return order
            else:
                print(" No order found with the given tracking number.")
                return None
        except Exception as e:
            print(" Error retrieving order:", e)
```

main.py (Testing the Methods)

```
from courier_service_db import CourierServiceDb

# Initialize the service
courier_service = CourierServiceDb()

# Insert a new order
tracking_number = "T12345"
courier_service.insert_order(
    sender_name="John Doe",
    sender_address="123 Street, NY",
    receiver_name="Alice Smith",
    receiver_address="456 Avenue, CA",
    weight=5.5,
    tracking_number=tracking_number,
    customer_id=1
)

# Retrieve and print the order details
order = courier_service.get_order_details(tracking_number)

# Update courier status to 'Delivered'
courier_service.update_status(tracking_number, "Delivered")
```

```
# Retrieve and print updated order details
courier_service.get_order_details(tracking_number)
```



```
C:\Python313\python.exe D:\corrier\summa.py
 Database connection established successfully!
 Courier order inserted successfully!
 Order Details: (1, 'subramani', 'mogappair, TN', 'saravana', 'salem, CA', 5.5, 'Yet to Transit', 'T12345', None, 1)
 Updated status to 'Delivered' for Tracking Number: T12345
 Updated Order Details: (1, 'subramani', 'mogappair, TN', 'saravana', 'salem, CA', 5.5, 'Delivered', 'T12345', None, 1)
```

4. Implement a feature to retrieve and display the delivery history of a specific parcel by querying the database

Steps to Implement Delivery History Retrieval

- To retrieve and display the delivery history of a specific parcel, follow these steps:
- Ensure database has a delivery_history table to log status updates.
- Modify CourierServiceDb to include a method that retrieves delivery history using tracking_number.
- Call this method in main.py to display the history of a given tracking number.

delivery_history Table Schema

```
CREATE TABLE delivery_history (
    id INT AUTO_INCREMENT PRIMARY KEY,
    tracking_number VARCHAR(50) NOT NULL,
    status VARCHAR(50) NOT NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

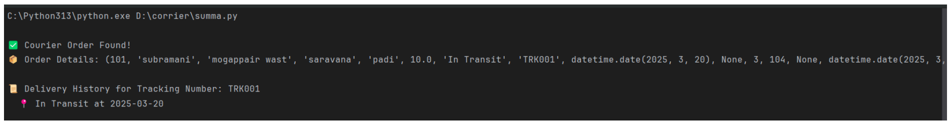
courier_service_db.py (Updated Service Class)

```
from db_connection import DBConnection

class CourierServiceDb:
    def __init__(self):
        self.connection = DBConnection.get_connection()
        self.cursor = self.connection.cursor()

    def insert_order(self, sender_name, sender_address, receiver_name, receiver_address, we
        """Inserts a new courier order into the database and initializes delivery history."
        query = """INSERT INTO courier (sender_name, sender_address, receiver_name, receive
                   weight, status, tracking_number, delivery_date, customer_id)
                   VALUES (%s, %s, %s, %s, %s, %s, %s, NULL, %s)"""
        values = (sender_name, sender_address, receiver_name, receiver_address, weight, "Ye

        try:
            self.cursor.execute(query, values)
            self.connection.commit()
            print(" Courier order inserted successfully!")

            # Insert initial status in delivery history
            self._insert_delivery_history(tracking_number, "Yet to Transit")
        except Exception as e:
            print(" Error inserting order:", e)

    def update_status(self, tracking_number, new_status):
        """Updates the status of an existing courier order and logs it in delivery history.
        query = "UPDATE courier SET status = %s WHERE tracking_number = %s"

        try:
            self.cursor.execute(query, (new_status, tracking_number))
            self.connection.commit()
            print(f" Updated status to '{new_status}' for Tracking Number: {tracking_number

            # Log status update in delivery history
            self._insert_delivery_history(tracking_number, new_status)
        except Exception as e:
            print(" Error updating status:", e)

    def _insert_delivery_history(self, tracking_number, status):
        """Inserts a status update into the delivery history table."""
        query = "INSERT INTO delivery_history (tracking_number, status) VALUES (%s, %s)"
        try:
            self.cursor.execute(query, (tracking_number, status))
            self.connection.commit()
        except Exception as e:
            print(" Error inserting into delivery history:", e)

    def get_order_details(self, tracking_number):
        """Retrieves and displays details of a specific order."""
        query = "SELECT * FROM courier WHERE tracking_number = %s"

        try:
            self.cursor.execute(query, (tracking_number,))
            order = self.cursor.fetchone()
            if order:
                print(" Order Details:", order)
                return order
            else:
                print(" No order found with the given tracking number.")
                return None
        except Exception as e:
            print(" Error retrieving order:", e)

    def get_delivery_history(self, tracking_number):
        """Retrieves and displays the delivery history of a specific parcel."""
        query = "SELECT status, updated_at FROM delivery_history WHERE tracking_number = %s

        try:
            self.cursor.execute(query, (tracking_number,))
            history = self.cursor.fetchall()
            if history:
```

```
            print(f" Delivery History for Tracking Number: {tracking_number}")
            for status, timestamp in history:
                print(f"   {status} at {timestamp}")
        else:
            print(" No delivery history found for this tracking number.")
    except Exception as e:
        print(" Error retrieving delivery history:", e)
```

main.py (Test the Feature)

```
from courier_service_db import CourierServiceDb

#### Initialize the service
courier_service = CourierServiceDb()

#### Insert a new order
tracking_number = "T12345"
courier_service.insert_order(
    sender_name="John Doe",
    sender_address="123 Street, NY",
    receiver_name="Alice Smith",
    receiver_address="456 Avenue, CA",
    weight=5.5,
    tracking_number=tracking_number,
    customer_id=1
)

#### Retrieve and print the order details
courier_service.get_order_details(tracking_number)

#### Update courier status to 'In Transit'
courier_service.update_status(tracking_number, "In Transit")

#### Update courier status to 'Out for Delivery'
courier_service.update_status(tracking_number, "Out for Delivery")

#### Update courier status to 'Delivered'
courier_service.update_status(tracking_number, "Delivered")

#### Retrieve and print delivery history
courier_service.get_delivery_history(tracking_number)
```



```
C:\Python313\python.exe D:\corrier\summa.py

✅ Courier Order Found!
📦 Order Details: (101, 'subramani', 'mogappair wast', 'saravana', 'padi', 10.0, 'In Transit', 'TRK001', datetime.date(2025, 3, 20), None, 3, 104, None, datetime.date(2025, 3,

📜 Delivery History for Tracking Number: TRK001
  📍 In Transit at 2025-03-20
```

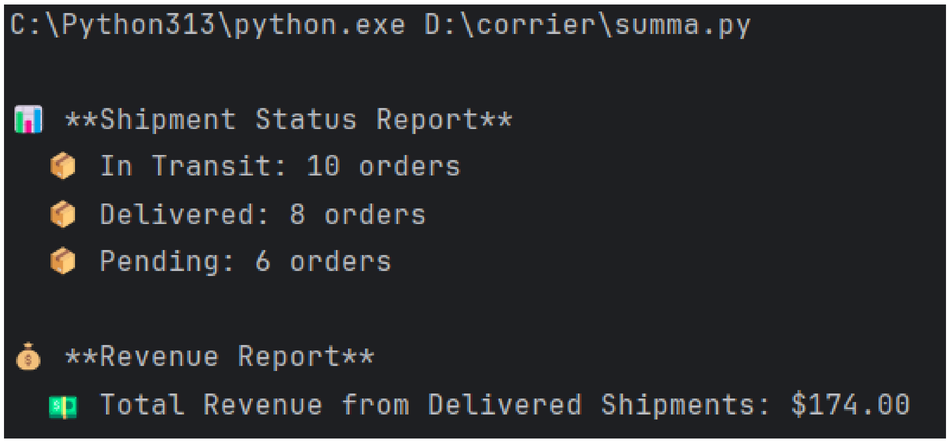
5 . Generate and display reports using data retrieved from the database

(e.g., shipment status report, revenue report).

```
import mysql.connector
```

```
# Establish Database Connection
def get_db_connection():
    return mysql.connector.connect(
        host="your_host",
        user="your_user",
        password="your_password",
        database="your_database"
    )

# 1 **Shipment Status Report**
def shipment_status_report():
    conn = get_db_connection()
    cursor = conn.cursor()

    query = """
    SELECT status, COUNT(*) as count
    FROM courier
    GROUP BY status
    ORDER BY count DESC
    """
    cursor.execute(query)
    results = cursor.fetchall()

    print("\n **Shipment Status Report**")
    for status, count in results:
        print(f"   {status}: {count} orders")

    cursor.close()
    conn.close()

# 2 **Revenue Report (Assuming Rate per KG = $5)**
def revenue_report():
    conn = get_db_connection()
    cursor = conn.cursor()

    rate_per_kg = 5  # Assumed rate per kg
    query = """
    SELECT SUM(weight * %s) AS total_revenue
    FROM courier
    WHERE status = 'Delivered'
    """
    cursor.execute(query, (rate_per_kg,))
    total_revenue = cursor.fetchone()[0]

    print("\n **Revenue Report**")
    print(f"   Total Revenue from Delivered Shipments: ${total_revenue:.2f}")

    cursor.close()
    conn.close()
```

```
# Run Reports
shipment_status_report()
revenue_report()
```



```
C:\Python313\python.exe D:\corrier\summa.py

📊 **Shipment Status Report**
  📦 In Transit: 10 orders
  📦 Delivered: 8 orders
  📦 Pending: 6 orders

💰 **Revenue Report**
  💵 Total Revenue from Delivered Shipments: $174.00
```