

# COURIER MANAGEMENT SYSTEM ASSIGNMENT

## CODING PART

### Task 1: Control Flow Statements

1: 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.

```
from db_connection import get_db_connection
def check_order_status(order_number):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return

    try:
        cursor = connection.cursor()
        query = "SELECT status FROM courier WHERE CourierID = %s"
        cursor.execute(query, (order_number,))
        result = cursor.fetchone()

        if result:
            print(f"The status of order {order_number} is: {result[0]}")
        else:
            print("Order not found.")

    except Exception as err:
        print(f"Error fetching order status: {err}")

    finally:
        cursor.close()
order_number = input("Enter the order number: ")
check_order_status(order_number)
```

The screenshot shows the PyCharm IDE interface. The project structure on the left includes a PythonProject folder with db\_connection.py and task1.py. The task1.py file contains Python code for checking order status. The run output window at the bottom shows the execution of task1.py, inputting '1' as the order number, and receiving the output 'The status of order 1 is: Delivered'. The system tray at the bottom indicates it's 83°F and partly cloudy.

```

def check_order_status(order_number):
    usage
    result = cursor.fetchone()

    if result:
        print(f"The status of order {order_number} is: {result[0]}")
    else:
        print("Order not found.")

    except Exception as err:
        print(f"Error fetching order status: {err}")

    finally:
        cursor.close()
order_number = input("Enter the order number: ")
check_order_status(order_number)

```

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(parcel_id):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT weight FROM courier WHERE CourierID = %s"
        cursor.execute(query, (int(parcel_id),))
        result = cursor.fetchone()
        if result:
            weight = float(result[0]) # Convert DB result to float
            if weight <= 2:
                category = "Light"
            elif 2 < weight <= 5:
                category = "Medium"
            else:
                category = "Heavy"
            print(f"Parcel ID: {parcel_id} | Weight: {weight}kg -> Category: {category}")
        else:
            print("Courier not found.")
    except Exception as err:
        print(f"Error fetching the weight: {err}")
    finally:
        if cursor:

```

```

        cursor.close()
if connection:
    connection.close()
parcel_id = input("Enter the courier ID: ").strip()
categorize_parcel(parcel_id)

```

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

```

from db_connection import get_db_connection
def categorize_parcel(parcel_id):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT weight FROM courier WHERE CourierID = %s"
        cursor.execute(query, (int(parcel_id),))
        result = cursor.fetchone()
        if result:
            weight = float(result[0]) # Convert DB result to float
            if weight <= 2:
                category = "Light"
            elif 2 < weight <= 5:
                category = "Medium"
            else:
                category = "Heavy"
            print(f"Parcel ID: {parcel_id} | Weight: {weight}kg -> Category: {category}")
    except Exception as err:
        print(f"Error fetching the weight: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

```

```

        print("Courier not found.")
    except Exception as err:
        print(f"Error fetching the weight: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
parcel_id = input("Enter the courier ID: ").strip()
categorize_parcel(parcel_id)

```

The screenshot shows the PyCharm IDE interface. The top navigation bar includes 'CourierManagementSystem' and 'Version control'. The left sidebar shows the project structure with files like db\_connection.py, task1.py, task2.py, and task3.py. The main code editor window displays the provided Python code. Below the editor is a terminal window showing the execution of task3.py, which prompts for a username and password, logs in successfully, and prints a welcome message. The bottom status bar indicates the file is saved, the encoding is UTF-8, and the Python version is 3.13.

```

def categorize_parcel(parcel_id):
    else:
        print("Courier not found.")

    except Exception as err:
        print(f"Error fetching the weight: {err}")

    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

parcel_id = input("Enter the courier ID: ").strip()
categorize_parcel(parcel_id)

```

#### 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.

```

from db_connection import get_db_connection
def assign_courier(shipment_id):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        cursor.execute("SELECT destination, weight FROM shipments WHERE ShipmentID = %s",
(shipment_id,))
        shipment = cursor.fetchone()
        if not shipment:
            print(f"Shipment {shipment_id} not found.")
            return
        destination, weight = shipment
        query = """SELECT CourierID, SenderName, Weight

```

```

    FROM courier
    WHERE ReceiverAddress = %s AND Weight >= %
    ORDER BY Weight ASC"""
cursor.execute(query, (destination, weight))
couriers = cursor.fetchall()
if not couriers:
    print(f"No available couriers for shipment {shipment_id}.")
    return
assigned_courier_id = couriers[0][0]
assigned_courier_name = couriers[0][1]
cursor.execute("UPDATE shipments SET AssignedCourierID = %s WHERE ShipmentID = %s",
               (assigned_courier_id, shipment_id))
connection.commit()
print(f"Shipment {shipment_id} assigned to {assigned_courier_name} (CourierID: {assigned_courier_id}).")
except Exception as err:
    print(f"Error assigning courier: {err}")
finally:
    if cursor:
        cursor.close()
    if connection:
        connection.close()
shipment_id = input("Enter Shipment ID: ").strip()
assign_courier(shipment_id)

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** CourierManagementSystem
- Files:** db\_connection.py, task1.py, task2.py, task3.py, task4.py (active), task4.py (terminal).
- Code in task4.py:**

```

def assign_courier(shipment_id):
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM couriers WHERE ReceiverAddress = %s AND Weight >= %s ORDER BY Weight ASC", (destination, weight))
    couriers = cursor.fetchall()
    if not couriers:
        print(f"No available couriers for shipment {shipment_id}.")
        return
    assigned_courier_id = couriers[0][0]
    assigned_courier_name = couriers[0][1]
    cursor.execute("UPDATE shipments SET AssignedCourierID = %s WHERE ShipmentID = %s", (assigned_courier_id, shipment_id))
    connection.commit()
    print(f"Shipment {shipment_id} assigned to {assigned_courier_name} (CourierID: {assigned_courier_id}).")
    except Exception as err:
        print(f"Error assigning courier: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
    shipment_id = input("Enter Shipment ID: ").strip()
    assign_courier(shipment_id)

```
- Terminal Output:**

```

C:\Users\Nithyasree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyasree\PycharmProjects\PythonProject\task4.py
Enter Shipment ID: 1
No available couriers for shipment 1.
Process finished with exit code 0

```
- System Status:** 82°F Partly cloudy, Search bar, Taskbar with various icons.

## Task 2: Loops and Iteration

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

```

from db_connection import get_db_connection
def display_orders(customer_id):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT ServicesID, ServiceName FROM courierservices WHERE ServicesID = %s"
        cursor.execute(query, (customer_id,))
        orders = cursor.fetchall()
        if not orders:
            print(f"No orders found for Customer ID {customer_id}.")
            return
        print(f"Orders for Customer ID {customer_id}:")
        for order in orders:
            print(f"Order ID: {order[0]}, Details: {order[1]}")
    except Exception as err:
        print(f"Error fetching orders: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
customer_id = input("Enter Customer ID: ").strip()
display_orders(customer_id)

```

The screenshot shows the PyCharm IDE interface with the CourierManagementSystem project open. The task5.py file is the active editor. The code implements a function to display orders for a given customer ID. When run, it prompts the user for a customer ID, lists the available orders, and then prints the details of the first order. The terminal window shows the execution path and the resulting output.

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

```
import time
from db_connection import get_db_connection
def track_courier(courier_id):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT CurrentLocation, Destination FROM couriertracking WHERE CourierID = %s"
        cursor.execute(query, (courier_id,))
        result = cursor.fetchone()
        if not result:
            print(f"No tracking data found for Courier ID {courier_id}.")
            return
        current_location, destination = result
        print(f"Tracking Courier ID {courier_id} from {current_location} to {destination}...\n")
        while current_location != destination:
            time.sleep(2)
            cursor.execute("SELECT CurrentLocation FROM couriertracking WHERE CourierID = %s",
                           (courier_id,))
            current_location = cursor.fetchone()[0]
            print(f"Current Location: {current_location}")
            print(f"Courier ID {courier_id} has reached its destination: {destination}!")
        except Exception as err:
            print(f"Error tracking courier: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
courier_id = input("Enter Courier ID: ").strip()
track_courier(courier_id)
```

```

def track_courier(courier_id):
    usage
    time.sleep(2)
    cursor.execute("SELECT CurrentLocation FROM couriertracking WHERE CourierID = %s", (courier_id,))
    current_location = cursor.fetchone()[0]
    print(f"Current location: {current_location}")

    print(f"Courier ID {courier_id} has reached its destination: {destination}!")

except Exception as err:
    print(f"Error tracking courier: {err}")

finally:
    if cursor:
        cursor.close()
    if connection:
        connection.close()
courier_id = input("Enter Courier ID: ").strip()
track_courier(courier_id)

```

## Task 3: Arrays and Data Structures

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

```

from db_connection import get_db_connection
def get_tracking_history(parcel_id):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT LocationHistory FROM parceltracking WHERE ParcelID = %s"
        cursor.execute(query, (parcel_id,))
        result = cursor.fetchone()
        if not result:
            print(f"No tracking history found for Parcel ID {parcel_id}.")
            return
        tracking_history = result[0].split(", ")
        print(f"Tracking History for Parcel ID {parcel_id}:")
        for index, location in enumerate(tracking_history, start=1):
            print(f"{index}. {location}")
    except Exception as err:
        print(f"Error fetching tracking history: {err}")
    finally:
        if cursor:
            cursor.close()

```

```

if connection:
    connection.close()
parcel_id = input("Enter Parcel ID: ").strip()
get_tracking_history(parcel_id)

```

```

def get_tracking_history(parcel_id):
    try:
        connection = get_db_connection()
        cursor = connection.cursor()
        cursor.execute(f"SELECT TrackingHistory FROM parcel WHERE ParcelID = {parcel_id}")
        result = cursor.fetchone()
        tracking_history = result[0].split(", ")
        print(f"Tracking History for Parcel ID {parcel_id}:")
        for index, location in enumerate(tracking_history, start=1):
            print(f"{index}. {location}")
    except Exception as err:
        print(f"Error fetching tracking history: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
parcel_id = input("Enter Parcel ID: ").strip()
get_tracking_history(parcel_id)

```

C:\Users\Nithyas\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyas\PycharmProjects\PythonProject\task7.py

Enter Parcel ID: 1

Tracking History for Parcel ID 1:

1. Warehouse
2. Mylapore
3. City Center

Process finished with exit code 0

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

```

from db_connection import get_db_connection
distance_matrix = {
    ("Warehouse", "City Center"): 5,
    ("Warehouse", "Mylapore"): 3,
    ("Hub A", "City Center"): 4
}
def calculate_distance(location1, location2):
    """Returns the distance between two locations based on predefined distances."""
    if location1 == location2:
        return 0
    return distance_matrix.get((location1, location2), float('inf'))
def find_nearest_courier(order_location):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT CourierID, SenderName, SenderAddress FROM courier WHERE Availability = 1"
        cursor.execute(query)
        couriers = cursor.fetchall()
        if not couriers:

```

```

        print("No available couriers at the moment.")
        return
    nearest_courier = None
    min_distance = float('inf')
    for courier in couriers:
        courier_id, courier_name, courier_location = courier
        distance = calculate_distance(courier_location, order_location)
        if distance < min_distance:
            min_distance = distance
            nearest_courier = (courier_id, courier_name, courier_location)
    if nearest_courier:
        print(f"Nearest available courier: {nearest_courier[1]} (ID: {nearest_courier[0]}) at {nearest_courier[2]}")
    except Exception as err:
        print(f"Error finding courier: {err}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
order_location = input("Enter Order Location: ").strip()
find_nearest_courier(order_location)

```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PythonProject [CourierManagementSystem]' with files like db\_connection.py, task1.py, task2.py, task4.py, task5.py, task6.py, task7.py, and task8.py. The main editor window shows the Python code for finding the nearest courier. The bottom 'Run' tab shows the terminal output of the program running successfully with the message 'No available couriers at the moment.' The status bar at the bottom right indicates the current time as 04-04-2025.

## 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.

```
from db_connection import get_db_connection
def track_parcel(tracking_number):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        query = "SELECT Status FROM parcels WHERE TrackingNumber = %s"
        cursor.execute(query, (tracking_number,))
        result = cursor.fetchone()
        if result:
            print(f"Tracking Number: {tracking_number} | Status: {result[0]}")
        else:
            print("Invalid Tracking Number. Please check again.")
    except Exception as err:
        print(f"Error fetching parcel status: {err}")
    finally:
        cursor.close()
        connection.close()
tracking_number = input("Enter your parcel tracking number: ").strip()
track_parcel(tracking_number)
```

The screenshot shows the PyCharm IDE interface. The project structure on the left includes a PythonProject folder containing a .venv library root and several task files (task1.py to task9.py). The current file is task9.py, which contains the provided Python code for tracking a parcel. The run tab at the bottom shows the command: C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\task9.py. The run output window displays the interaction: Enter your parcel tracking number: 1, followed by the response: Tracking Number: 1 | Status: Delivered. The status bar at the bottom right indicates the date as 04-04-2025.

10. Customer Data Validation: Write a function which takes 2 parameters, data-denotes the data and detail-denotes if it is name address 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
import mysql.connector
from mysql.connector import Error
def get_db_connection():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            database='couriermanagementsystem',
            user='root',
            password='Deepa@25'
        )
        if connection.is_connected():
            return connection
    except Error as err:
        print(f"Error: {err}")
        return None
def validate_user_data(data, detail):
    connection = get_db_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        if detail == "name":
            if not re.fullmatch(r"[A-Z][a-zA-Z\s]*$", data):
                return "Invalid Name: Only letters allowed, must start with a capital letter."
            cursor.execute("SELECT * FROM user WHERE Name = %s", (data,))
        elif detail == "address":
            if not re.fullmatch(r"[A-Za-z0-9\s,-]+\$", data):
                return "Invalid Address: Special characters are not allowed."
            cursor.execute("SELECT * FROM user WHERE Address = %s", (data,))
        elif detail == "phone":
            if not re.fullmatch(r"\d{3}-\d{3}-\d{4}\$)", data):
                return "Invalid Phone Number: Format should be ###-###-####."
            cursor.execute("SELECT * FROM user WHERE Phone = %s", (data,))
        else:
            return "Invalid Detail Type"
        if cursor.fetchone():
            return f"Valid {detail.capitalize()} (Exists in DB)"
        else:
            return f"Valid {detail.capitalize()} (Not found in DB)"
    except Exception as err:
        return f"Error validating {detail}: {err}"
    finally:
        cursor.close()
        connection.close()
detail_type = input("Enter detail type (name, address, phone): ").strip().lower()
data_input = input(f"Enter {detail_type}: ").strip()

```

```
result = validate_user_data(data_input, detail_type)
print(result)
```

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists files like task2.py, task4.py, task5.py, task6.py, task7.py, task8.py, task9.py, and task10.py. The current file, task10.py, is open in the editor. The code contains a function validate\_user\_data that interacts with a database to validate user input. The run output window below shows the execution of task10.py, where the user enters 'name' as the detail type, and the program prints 'Valid Name (Exists in DB)'.

```
def validate_user_data(data, detail):
    usage
    else:
        return f"Valid {detail.capitalize()} (Not found in DB)"

    except Exception as err:
        return f"Error validating {detail}: {err}"

    finally:
        cursor.close()
        connection.close()

detail_type = input("Enter detail type (name, address, phone): ").strip().lower()
data_input = input(f"Enter {detail_type}: ").strip()
result = validate_user_data(data_input, detail_type)
print(result)
```

**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
from db_connection import get_db_connection
def format_address(user_id, street, city, state, zip_code):
    formatted_street = street.title()
    formatted_city = city.title()
    formatted_state = state.upper() # States are usually in uppercase (e.g., NY, CA)
    if not re.fullmatch(r"\d{5}(-\d{4})?", zip_code):
        return "Invalid ZIP Code format. Use XXXXX or XXXXX-XXXX."
    formatted_address = f"{formatted_street}, {formatted_city}, {formatted_state} - {zip_code}"
    connection = get_db_connection()
    if connection is None:
        return "Database connection failed."
    try:
        cursor = connection.cursor()
        query = "UPDATE user SET Address = %s WHERE UserID = %s"
        cursor.execute(query, (formatted_address, user_id))
        connection.commit()
        print("Address updated successfully in the database.")
    except Exception as err:
        print(f"Error updating address: {err}")
    finally:
        cursor.close()
        connection.close()
```

```

    return formatted_address
user_id = input("Enter user ID: ").strip()
street = input("Enter street: ").strip()
city = input("Enter city: ").strip()
state = input("Enter state (abbreviation): ").strip()
zip_code = input("Enter ZIP code: ").strip()
formatted_address = format_address(user_id, street, city, state, zip_code)
print(f"\nFormatted Address:\n{formatted_address}")

```

**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
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
customer_name = input("Enter Customer Name: ")
order_number = input("Enter Order Number: ")

```

```

address = input("Enter Delivery Address: ")
delivery_date = (datetime.date.today() + datetime.timedelta(days=3)).strftime("%Y-%m-%d")
email_content = generate_order_email(customer_name, order_number, address, delivery_date)
print("\n" + email_content)

```

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists files like task5.py, task6.py, task7.py, task8.py, task9.py, task10.py, task11.py, and task12.py. The main editor window displays Python code for generating an order email. Below the editor is the run output terminal, which shows the user entering their details and the resulting confirmation email. The bottom status bar indicates the Python version is 3.13.

```

def generate_order_email(customer_name, order_number, address, delivery_date):
    """+
    Regards,
    *Courier Service Team**
    """
    return email_template

customer_name = input("Enter Customer Name: ")
order_number = input("Enter Order Number: ")
address = input("Enter Delivery Address: ")
delivery_date = (datetime.date.today() + datetime.timedelta(days=3)).strftime("%Y-%m-%d")

email_content = generate_order_email(customer_name, order_number, address, delivery_date)
print("\n" + email_content)

```

**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.

```

distance_data = {
    ("Chennai", "Bangalore"): 350,
    ("Chennai", "Delhi"): 2200,
    ("Chennai", "Mumbai"): 1330,
    ("Bangalore", "Mumbai"): 980,
    ("Mumbai", "Delhi"): 1440,
    ("Delhi", "Kolkata"): 1530,
    ("Hyderabad", "Chennai"): 630,
}
BASE_COST = 50 # Base charge
COST_PER_KM = 0.5 # Cost per km
COST_PER_KG = 10 # Cost per kg
def calculate_shipping_cost(source, destination, weight):
    # Convert input to title case to match dictionary keys
    source = source.title()
    destination = destination.title()
    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
shipping_cost = BASE_COST + (distance * COST_PER_KM) + (weight * COST_PER_KG)
print(f"Shipping Cost from {source} to {destination}: ₹{shipping_cost:.2f}")
source = input("Enter Source Location: ")
destination = input("Enter Destination Location: ")
weight = float(input("Enter Parcel Weight (kg): "))
calculate_shipping_cost(source, destination, weight)

```

The screenshot shows the PyCharm interface with the 'CourierManagementSystem' project open. The 'task13.py' file is the active editor, displaying the shipping cost calculation code. The run output window shows the execution of the program with the specified inputs and the resulting shipping cost.

**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
from db_connection import get_db_connection
def generate_password(length=12):
    """Generate a secure password with a mix of uppercase, lowercase, numbers, and special characters."""
    if length < 8:
        print("Password length should be at least 8 characters for security.")
        return None

    characters = (
        string.ascii_uppercase + string.ascii_lowercase + string.digits + string.punctuation
    )
    password = ''.join(random.choices(characters, k=length))
    return password

def store_password(username, password):

```

```

"""Store the generated password in the database."""
connection = get_db_connection()
if connection is None:
    print("Failed to connect to the database.")
    return
try:
    cursor = connection.cursor()
    query = "UPDATE user SET password = %s WHERE Name = %s"
    cursor.execute(query, (password, username))
    connection.commit()
    print("Password successfully updated in the database.")
except Exception as err:
    print(f"Error storing password: {err}")
finally:
    cursor.close()
    connection.close()
username = input("Enter username for the account: ").strip()
password = generate_password()
if password:
    print(f"Generated Password: {password}")
    store_password(username, password)

```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'CourierManagementSystem' with files like db\_connection.py, task1.py through task14.py, and task11.py to task13.py. The main editor window shows the code for task14.py. The terminal window at the bottom shows the execution of the script. It prompts for a username ('Enter username for the account: Nithya'), generates a password ('Generated Password: H!Y7Tm;J\|P\*'), and then prints 'Password successfully updated in the database.' The status bar at the bottom right indicates the current date and time (04-04-2025) and system information.

**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.

```

from db_connection import get_db_connection
def find_similar_addresses(input_address):
    """Find similar addresses in the database using string matching techniques."""
    connection = get_db_connection()

```

```

if connection is None:
    print("Failed to connect to the database.")
    return

try:
    cursor = connection.cursor()
    # Query to find similar addresses using SQL functions
    query = """
SELECT Address FROM user
WHERE Address LIKE %s
OR SOUNDEX(Address) = SOUNDEX(%s)
"""

    cursor.execute(query, (f"%{input_address}%", input_address))
    results = cursor.fetchall()
    if results:
        print("Similar Addresses Found:")
        for row in results:
            print(row[0])
    else:
        print("No similar addresses found.")
except Exception as err:
    print(f"Error finding similar addresses: {err}")
finally:
    cursor.close()
    connection.close()

input_address = input("Enter an address to search for similar ones: ").strip()
find_similar_addresses(input_address)

```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files like task8.py through task14.py, task15.py, and a connection file. The main editor window contains the Python code for task15.py. The bottom terminal window shows the execution of the script, where the user enters 'Saidapet' and the program outputs 'Similar Addresses Found:' followed by 'Nithya, Saidapet, TN - 600015'. The status bar at the bottom right indicates the date and time as 04-04-2025 and 02:22.

## Task 5: Object Oriented Programming

1. Create the following model/entity classes within package entities with variables declared private, constructors(default and parametrized,getters, setters and toString()) 1. User Class:  
Variables: userID , userName , email , password , contactNumber , address

class User:

```
def __init__(self, user_id=None, user_name=None, email=None, password=None,
contact_number=None, address=None):
    self.__user_id = user_id
    self.__user_name = user_name
    self.__email = email
    self.__password = password
    self.__contact_number = contact_number
    self.__address = address

def get_user_id(self):
    return self.__user_id

def get_user_name(self):
    return self.__user_name

def get_email(self):
    return self.__email

def get_password(self):
    return self.__password

def get_contact_number(self):
    return self.__contact_number

def get_address(self):
    return self.__address

# Setters

def set_user_id(self, user_id):
    self.__user_id = user_id

def set_user_name(self, user_name):
    self.__user_name = user_name

def set_email(self, email):
    self.__email = email
```

```

def set_password(self, password):
    self.__password = password

def set_contact_number(self, contact_number):
    self.__contact_number = contact_number

def set_address(self, address):
    self.__address = address

# toString equivalent (for displaying object details)

def __str__(self):
    return f"User(user_id={self.__user_id}, user_name='{self.__user_name}', email='{self.__email}', "
\\
    f"contact_number='{self.__contact_number}', address='{self.__address}')"

```

### "IMPORTING USER CLASS"

```

from entities.user import User

# Creating a User object

user1 = User(1, "Nithya", "nithu@gmailexample.com", "hi@123", "9876543210", "123 Main Street")

# Display user details

print(user1)

print("User Email:", user1.get_email()) # Get email

user1.set_email("newemail@example.com") # Update email

print("Updated User Email:", user1.get_email())

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "PythonProject [CourierManagementSystem]" located at "C:\Users\Nithya". It contains a "entities" folder which includes "user.py" and "db\_connection.py". Other files like "main.py", "task1.py", etc., are also listed.
- Main.py Content:**

```

from entities.user import User

# Creating a User object
user1 = User(1, "Nithya", "nithu@gmailexample.com", "hi@123", "9876543210", "123 Main Street")

# Display user details
print(user1)

# Accessing and modifying attributes using getters and setters
print("User Email:", user1.get_email()) # Get email
user1.set_email("newemail@example.com") # Update email
print("Updated User Email:", user1.get_email())

```
- Run Tab:** Shows the command run: "C:\Users\Nithya\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithya\PycharmProjects\PythonProject\main.py". The output window shows the results of the program execution.
- Output Window:**

```

User(user_id=1, user_name='Nithya', email='nithu@gmailexample.com', contact_number='9876543210', address='123 Main Street')
User Email: nithu@gmailexample.com
Updated User Email: newemail@example.com

```
- Bottom Status Bar:** Shows the current time as 4:53, date as 04-04-2025, and system status including battery level and network connection.

2. Courier Class Variables: courierID , senderName , senderAddress , receiverName , receiverAddress , weight , status, trackingNumber , deliveryDate ,userId

```
class Courier:  
    tracking_counter = 1000  
  
    def __init__(self, courier_id=None, sender_name=None, sender_address=None,  
     receiver_name=None,  
     receiver_address=None, weight=None, status="Yet to Transit", delivery_date=None,  
     user_id=None):  
        self.__courier_id = courier_id  
        self.__sender_name = sender_name  
        self.__sender_address = sender_address  
        self.__receiver_name = receiver_name  
        self.__receiver_address = receiver_address  
        self.__weight = weight  
        self.__status = status  
        self.__tracking_number = f"TRK-{Courier.tracking_counter}"  
  
    Courier.tracking_counter += 1 # Increment for the next courier  
    self.__delivery_date = delivery_date  
    self.__user_id = user_id  
  
    def get_courier_id(self):  
        return self.__courier_id  
  
    def get_sender_name(self):  
        return self.__sender_name  
  
    def get_sender_address(self):  
        return self.__sender_address  
  
    def get_receiver_name(self):  
        return self.__receiver_name  
  
    def get_receiver_address(self):  
        return self.__receiver_address  
  
    def get_weight(self):  
        return self.__weight  
  
    def get_status(self):
```

```

    return self.__status

def get_tracking_number(self):
    return self.__tracking_number

def get_delivery_date(self):
    return self.__delivery_date

def get_user_id(self):
    return self.__user_id

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

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

def __str__(self):
    return f"Courier(courier_id={self.__courier_id}, sender='{self.__sender_name}', receiver='{self.__receiver_name}', "
           f"status='{self.__status}', tracking_number='{self.__tracking_number}', delivery_date='{self.__delivery_date}')"

```

#### "IMPORTING COURIER CLASS"

```

from entities.courier import Courier
courier1 = Courier(1, "Nithya", "123 Main St", "Ezhil", "J road", 2.5, "In Transit", "2025-04-10", 101)
print(courier1)
print("Tracking Number:", courier1.get_tracking_number())
courier1.set_status("Delivered")
print("Updated Status:", courier1.get_status())

```

```

from entities.courier import Courier
# Creating a new Courier object
courier1 = Courier(courier_id=1, sender_name="Nithya", sender_address="123 Main St", receiver_name="Ezhil", receiver_address="456 Elm St")
# Display courier details
print(courier1)
# Get tracking number
print("Tracking Number:", courier1.get_tracking_number())
# Update status
courier1.set_status("Delivered")
print("Updated Status:", courier1.get_status())

```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\test\_courier.py

Tracking Number: TRK-1000  
Updated Status: Delivered

Process finished with exit code 0

### 3. Employee Class: Variables employeeID , employeeName , email , contactNumber , role String, salary

class Employee:

```

def __init__(self, employeeID, employeeName, email, contactNumber, role, salary):
    self.__employeeID = employeeID
    self.__employeeName = employeeName
    self.__email = email
    self.__contactNumber = contactNumber
    self.__role = role
    self.__salary = salary

def get_employeeID(self):
    return self.__employeeID

def get_employeeName(self):
    return self.__employeeName

def get_email(self):
    return self.__email

def get_contactNumber(self):
    return self.__contactNumber

def get_role(self):
    return self.__role

```

```

def get_salary(self):
    return self.__salary

def set_employeeName(self, employeeName):
    self.__employeeName = employeeName

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

def set_contactNumber(self, contactNumber):
    self.__contactNumber = contactNumber

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

def set_salary(self, salary):
    self.__salary = salary

def __str__(self):
    return (f"Employee ID: {self.__employeeID}, Name: {self.__employeeName}, Email: {self.__email}, "
           f"Contact: {self.__contactNumber}, Role: {self.__role}, Salary: {self.__salary}")

```

#### **"IMPORTING EMPLOYEE CLASS"**

```

from entities.employee import Employee

employee1 = Employee(1, "John Doe", "john.doe@example.com", "9876543210", "Manager",
75000)

print(employee1)

employee1.set_role("Senior Manager")

print("Updated Role:", employee1.get_role())

employee1.set_salary(85000)

print("Updated Salary:", employee1.get_salary())

```

```

from entities.employee import Employee
employee1 = Employee( employeeId=1, employeeName="John Doe", email="john.doe@example.com", contactNumber="9876543210")
# Display employee details
print(employee1)
# Update role
employee1.set_role("Senior Manager")
print("Updated Role:", employee1.get_role())
# Update salary
employee1.set_salary(85000)
print("Updated Salary:", employee1.get_salary())

```

#### 4. Location Class Variables LocationID , LocationName , Address

class Location:

```

def __init__(self, locationID, locationName, address):
    self.__locationID = locationID
    self.__locationName = locationName
    self.__address = address

def get_locationID(self):
    return self.__locationID

def get_locationName(self):
    return self.__locationName

def get_address(self):
    return self.__address

def set_locationName(self, locationName):
    self.__locationName = locationName

def set_address(self, address):
    self.__address = address

def __str__(self):
    return f"Location ID: {self.__locationID}, Name: {self.__locationName}, Address: {self.__address}"

```

#### "IMPORTING LOCATION CLASS"

```

from entities.location import Location
location1 = Location(101, "Temple", "PK street, Saidapet")
print(location1)
# Update location name
location1.set_locationName("Central Warehouse")
print("Updated Location Name:", location1.get_locationName())
location1.set_address("789 GST Road, Urban Square")
print("Updated Address:", location1.get_address())

```

```

from entities.location import Location
location1 = Location(101, "Temple", "PK street, Saidapet")

# Display location details
print(location1)

# Update location name
location1.set_locationName("Central Warehouse")
print("Updated Location Name:", location1.get_locationName())

# Update address
location1.set_address("789 GST Road, Urban Square")
print("Updated Address:", location1.get_address())

```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\test\_location.py

Location ID: 101, Name: Temple, Address: PK street, Saidapet  
 Updated Location Name: Central Warehouse  
 Updated Address: 789 GST Road, Urban Square

Process finished with exit code 0

**5. CourierCompany Class Variables** `companyName` , `courierDetails` -collection of Courier Objects, `employeeDetails` collection of Employee Objects, `locationDetails` - collection of Location Objects.

```

from entities.courier import Courier
from entities.employee import Employee
from entities.location import Location

class CourierCompany:

    def __init__(self, companyName):
        self.__companyName = companyName
        self.__courierDetails = [] # List to store Courier objects
        self.__employeeDetails = [] # List to store Employee objects
        self.__locationDetails = [] # List to store Location objects

    def get_companyName(self):
        return self.__companyName

    def get_courierDetails(self):

```

```

        return self.__courierDetails

    def get_employeeDetails(self):
        return self.__employeeDetails

    def get_locationDetails(self):
        return self.__locationDetails

    def set_companyName(self, companyName):
        self.__companyName = companyName

    def add_courier(self, courier):
        if isinstance(courier, Courier):
            self.__courierDetails.append(courier)

    def add_employee(self, employee):
        if isinstance(employee, Employee):
            self.__employeeDetails.append(employee)

    def add_location(self, location):
        if isinstance(location, Location):
            self.__locationDetails.append(location)

    def __str__(self):
        return f"Company Name: {self.__companyName}, " \
               f"Couriers: {len(self.__courierDetails)}, " \
               f"Employees: {len(self.__employeeDetails)}, " \
               f"Locations: {len(self.__locationDetails)}"

```

### **"IMPORTING COURIERCOMPANY CLASS"**

```

from entities.courier import Courier
from entities.employee import Employee
from entities.location import Location

class CourierCompany:

    def __init__(self, companyName):
        self.__companyName = companyName
        self.__courierDetails = [] # List to store Courier objects
        self.__employeeDetails = [] # List to store Employee objects
        self.__locationDetails = [] # List to store Location objects

```

```
def get_companyName(self):
    return self.__companyName

def get_courierDetails(self):
    return self.__courierDetails

def get_employeeDetails(self):
    return self.__employeeDetails

def get_locationDetails(self):
    return self.__locationDetails

def set_companyName(self, companyName):
    self.__companyName = companyName

def add_courier(self, courier):
    if isinstance(courier, Courier):
        self.__courierDetails.append(courier)

def add_employee(self, employee):
    if isinstance(employee, Employee):
        self.__employeeDetails.append(employee)

def add_location(self, location):
    if isinstance(location, Location):
        self.__locationDetails.append(location)

def __str__(self):
    return f"Company Name: {self.__companyName}, " \
           f"Couriers: {len(self.__courierDetails)}, " \
           f"Employees: {len(self.__employeeDetails)}, " \
           f"Locations: {len(self.__locationDetails)}"
```

```
from entities.courier_company import CourierCompany
from entities.courier import Courier
from entities.employee import Employee
from entities.location import Location

# Creating a CourierCompany object
company = CourierCompany("SpeedX Couriers")

# Adding Courier objects
courier1 = Courier(courier_id=1, sender_name="Alice", sender_address="123 Main St", receiver_name="Bob", receiver_address="456 Elm St")
company.add_courier(courier1)

# Adding Employee objects
employee1 = Employee(employee_id=201, employee_name="John Doe", email="john@example.com", contact_number="9876543210", role="Delivery")
company.add_employee(employee1)

# Adding Location objects
location1 = Location(location_id=101, location_name="Downtown Hub", address="456 Market Street, City Center")
company.add_location(location1)
```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\test\_courier\_company.py  
Company Name: SpeedX Couriers, Couriers: 1, Employees: 1, Locations: 1  
Process finished with exit code 0

## 6. Payment Class: Variables PaymentID long, CourierID long, Amount double, PaymentDate Date

```
import datetime
```

```
class Payment:
```

```
    def __init__(self, paymentID, courierID, amount, paymentDate=None):
        self.__paymentID = paymentID
        self.__courierID = courierID
        self.__amount = amount
        self.__paymentDate = paymentDate if paymentDate else datetime.date.today()
```

```
    def get_paymentID(self):
```

```
        return self.__paymentID
```

```
    def get_courierID(self):
```

```
        return self.__courierID
```

```
    def get_amount(self):
```

```
        return self.__amount
```

```
    def get_paymentDate(self):
```

```
        return self.__paymentDate
```

```
    def set_paymentID(self, paymentID):
```

```
        self.__paymentID = paymentID
```

```
    def set_courierID(self, courierID):
```

```

    self.__courierID = courierID

def set_amount(self, amount):
    self.__amount = amount

def set_paymentDate(self, paymentDate):
    self.__paymentDate = paymentDate

def __str__(self):
    return f"Payment ID: {self.__paymentID}, Courier ID: {self.__courierID}, Amount: {self.__amount}, Date: {self.__paymentDate}"

```

### **“IMPORTING PAYMENT CLASS”**

```

from entities.payment import Payment

import datetime

payment1 = Payment(501, 1, 150.75)

print(payment1)

payment1.set_amount(175.50)

payment1.set_paymentDate(datetime.date(2025, 4, 5))

print("Updated Payment:", payment1)

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project View:** Shows files like task6.py, task8.py, task9.py, task10.py, task11.py, task12.py, task13.py, task14.py, task15.py, test\_courier.py, test\_courier\_company.py, test\_employee.py, test\_location.py, test\_payment.py, and test\_user.py.
- Code Editor:** The current file is test\_payment.py, containing Python code to create a Payment object, update its amount and date, and print the updated details.
- Run Tab:** Shows the run configuration for task6 and test\_payment.
- Output Tab:** Displays the terminal output of the executed code.
- Bottom Status Bar:** Shows system information including temperature (82°F), weather (Partly cloudy), and system status (ENG IN).

```

1 from entities.payment import Payment
2 import datetime
3
4 # Creating a Payment object
5 payment1 = Payment(paymentID: 501, courierID: 1, amount: 150.75)
6
7 # Display payment details
8 print(payment1)
9
10 # Updating payment details
11 payment1.set_amount(175.50)
12 payment1.set_paymentDate(datetime.date(year: 2025, month: 4, day: 5))
13 print("Updated Payment:", payment1)
14

```

```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\test_payment.py
Payment ID: 501, Courier ID: 1, Amount: 150.75, Date: 2025-04-04
Updated Payment: Payment ID: 501, Courier ID: 1, Amount: 175.5, Date: 2025-04-05
Process finished with exit code 0

```

**Task 6: Service Provider Interface /Abstract class Create 2 Interface /Abstract class ICourierUserService and ICourierAdminService interface**

```
ICourierUserService {  
    // Customer-related functions  
  
    placeOrder()  
  
    getOrderStatus();  
  
    cancelOrder()  
  
    getAssignedOrder();  
  
    from abc import ABC, abstractmethod  
  
    class ICourierUserService(ABC):  
  
        @abstractmethod  
        def place_order(self, courier_obj):  
            """  
            Place a new courier order.  
            :param courier_obj: Courier object created using user inputs  
            :return: Unique tracking number for the order  
            """  
            pass  
  
        @abstractmethod  
        def get_order_status(self, tracking_number):  
            """  
            Get the status of a courier order.  
            :param tracking_number: Tracking number of the order  
            :return: Status string (e.g., 'In Transit', 'Delivered')  
            """  
            pass  
        @abstractmethod  
        def cancel_order(self, tracking_number):  
            """  
            Cancel a courier order.  
            :param tracking_number: Tracking number to cancel  
            :return: True if cancellation successful, else False  
            """  
            pass  
        @abstractmethod  
        def get_assigned_order(self, courier_staff_id):  
            """  
            Get orders assigned to a courier staff member.  
            :param courier_staff_id: Staff ID  
            :return: List of courier orders  
            """  
            pass
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "CourierManagementSystem". It contains a "PythonProject" folder with a ".venv" library root. Under "dao", there are files: I\_courier\_admin\_service.py, I\_courier\_user\_service.py, and exception.py. Under "entities", there are files: courier.py, courier\_company.py, employee.py, location.py, payment.py, and user.py. There are also main.py, task1.py, task2.py, and task3.py files.
- Code Editor:** The file "I\_courier\_user\_service.py" is open. The code defines an abstract base class "ICourierUserService(ABC)". It includes two abstract methods: "cancel\_order" which takes a tracking number and returns a boolean indicating cancellation success, and "get\_assigned\_order" which takes a staff ID and returns a list of orders assigned to that staff member.
- Run Output:** The terminal window shows the command "python C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\dao\I\_courier\_user\_service.py" was run, and the process finished with exit code 0.
- System Tray:** The system tray shows icons for Anna Salai Construction, battery level (1516), and network status (ENG IN).

// Admin functions

ICourierAdminService

int addCourierStaff(Employee obj);

from abc import ABC, abstractmethod

```
class ICourierAdminService(ABC):
    @abstractmethod
    def add_courier_staff(self, employee_obj):
        """
        Add a new courier staff member.
        :param employee_obj: Employee object
        :return: ID of newly added staff
        """
    pass
```

```

from abc import ABC, abstractmethod

class ICourierAdminService(ABC):
    @abstractmethod
    def add_courier_staff(self, employee_obj):
        """
        Add a new courier staff member.
        :param employee_obj: Employee object
        :return: ID of newly added staff
        """
        pass

```

## Task 7: Exception Handling (Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,throw & throws keyword usage)

1. TrackingNumberNotFoundException :throw this exception when user try to withdraw amount or transfer amount to another acco

```

from db_connection import get_db_connection
from exception.tracking_number_not_found_exception import TrackingNumberNotFoundException
def find_tracking_number(tracking_number):
    conn = get_db_connection()
    if not conn:
        print("Failed to connect to DB.")
        return
    cursor = conn.cursor()
    cursor.execute("SELECT trackingNumber FROM Courier WHERE trackingNumber = %s",
    (tracking_number,))
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    if not result:
        raise TrackingNumberNotFoundException(tracking_number)
if __name__ == "__main__":
    try:
        user_tracking_input = input("Enter Tracking Number to Check Status: ")
        find_tracking_number(user_tracking_input)
        print("Tracking number is valid and found in system.")
    except TrackingNumberNotFoundException as e:
        print("Error:", e)

```

```

finally:
    print("Tracking number check complete.")

```

The screenshot shows the PyCharm IDE interface with the project 'CourierManagementSystem' open. The current file is 'check\_tracking\_number.py'. The code contains logic to handle tracking number input and exceptions. The terminal window shows the output of running the script. It prompts for a tracking number, receives '1' as input, and prints an error message: 'Error: Tracking number '1' not found.' followed by 'Tracking number check complete.' The status bar at the bottom indicates the script was run with exit code 0.

```

def find_tracking_number(tracking_number):
    usage
    conn.close()
    if not result:
        raise TrackingNumberNotFoundException(tracking_number)
if __name__ == "__main__":
    try:
        user_tracking_input = input("Enter Tracking Number to Check Status: ")
        find_tracking_number(user_tracking_input)
        print("Tracking number is valid and found in system.")
    except TrackingNumberNotFoundException as e:
        print("Error: ", e)
finally:
    print("Tracking number check complete.")

```

THE ABOVE IMAGE SHOWS THE EXCEPTION THROWN FOR INVALID INPUT OF THE TRACKING NUMBER

This screenshot is identical to the previous one, showing the PyCharm IDE and terminal output. However, the tracking number entered is now 'TR1', which is valid. The terminal output shows 'Tracking number is valid and found in system.' and 'Tracking number check complete.' The status bar indicates the exit code is 0.

```

def find_tracking_number(tracking_number):
    usage
    conn.close()
    if not result:
        raise TrackingNumberNotFoundException(tracking_number)
if __name__ == "__main__":
    try:
        user_tracking_input = input("Enter Tracking Number to Check Status: ")
        find_tracking_number(user_tracking_input)
        print("Tracking number is valid and found in system.")
    except TrackingNumberNotFoundException as e:
        print("Error: ", e)
finally:
    print("Tracking number check complete.")

```

THE ABOVE IMAGE SHOWS THE OUTPUT GIVEN FOR VALID INPUT OF THE TRACKING NUMBER

2. InvalidEmployeeIdException throw this exception when id entered for the employee not existing in the system

```

from db_connection import get_db_connection
from exception.invalid_employee_id_exception import InvalidEmployeeIdException

```

```
def find_employee_by_id(emp_id):
    conn = get_db_connection()

    if not conn:
        print("Failed to connect to DB.")

        return

    cursor = conn.cursor()

    cursor.execute("SELECT employeeID FROM Employee WHERE employeeID = %s", (emp_id,))

    result = cursor.fetchone()

    cursor.close()

    conn.close()

    if not result:
        raise InvalidEmployeeIdException(emp_id)

if __name__ == "__main__":
    try:
        emp_id_input = int(input("Enter Employee ID to Validate: "))

        find_employee_by_id(emp_id_input)

        print(" Employee ID is valid and found in system.")

    except InvalidEmployeeIdException as e:
        print("Error:", e)

    except ValueError:
        print("Error: Please enter a valid numeric employee ID.")

    finally:
        print("Employee ID check complete.")
```

```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\check_employee_id.py
Enter Employee ID to Validate: 6
Error: Invalid Employee ID: 6
Employee ID check complete.

Process finished with exit code 0

```

THE ABOVE IMAGE SHOWS THE EXCEPTION THROWN FOR INVALID INPUT OF THE EMPLOYEE ID

```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\check_employee_id.py
Enter Employee ID to Validate: 1
Employee ID is valid and found in system.
Employee ID check complete.

Process finished with exit code 0

```

THE ABOVE IMAGE SHOWS OUTPUT GIVEN FOR VALID INPUT OF THE EMPLOYEE ID

## Task 8: 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

from entities.courier import Courier

from entities.employee import Employee

from entities.location import Location

```

class CourierCompanyCollection:

    def __init__(self, company_name):
        self.company_name = company_name
        self.courier_details = [] # List of Courier objects
        self.employee_details = [] # List of Employee objects
        self.location_details = [] # List of Location objects

    def add_courier(self, courier: Courier):
        self.courier_details.append(courier)

    def add_employee(self, employee: Employee):
        self.employee_details.append(employee)

    def add_location(self, location: Location):
        self.location_details.append(location)

    def get_all_couriers(self):
        return self.courier_details

    def get_all_employees(self):
        return self.employee_details

    def get_all_locations(self):
        return self.location_details

    def __str__(self):
        return f"Company Name: {self.company_name}\n" \
               f"Total Couriers: {len(self.courier_details)}\n" \
               f"Total Employees: {len(self.employee_details)}\n" \
               f"Total Locations: {len(self.location_details)}"

```

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

```

from dao.i_courier_user_service import ICourierUserService
from entities.courier_company_collection import CourierCompanyCollection

class CourierUserServiceCollectionImpl(ICourierUserService):

    def __init__(self, company_obj: CourierCompanyCollection):
        self.company_obj = company_obj

```

```

        self.tracking_number_seed = 1000 # Static starting point

    def place_order(self, courier_obj):
        self.tracking_number_seed += 1
        courier_obj.set_tracking_number(f"TRK{self.tracking_number_seed}")
        self.company_obj.add_courier(courier_obj)
        return courier_obj.get_tracking_number()

    def get_order_status(self, tracking_number):
        for courier in self.company_obj.get_all_couriers():
            if courier.get_tracking_number() == tracking_number:
                return courier.get_status()
        return "Tracking number not found"

    def cancel_order(self, tracking_number):
        for courier in self.company_obj.get_all_couriers():
            if courier.get_tracking_number() == tracking_number:
                self.company_obj.courier_details.remove(courier)
        return True
        return False

    def get_assigned_order(self, courier_staff_id):
        assigned_orders = [
            c for c in self.company_obj.get_all_couriers()
            if c.get_sender_contact() == courier_staff_id # Adjust as per actual logic
        ]
        return assigned_orders

```

**By themselves, these two classes (CourierCompanyCollection and CourierUserServiceCollectionImpl) won't give any output unless we call them explicitly, like in a test or main module.**

**Calling them in main.py:**

```

from entities.courier import Courier
from entities.courier_company_collection import CourierCompanyCollection
from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
# Initialize tracking number from DB before creating any courier
Courier.initialize_tracking_counter()
company = CourierCompanyCollection("SpeedXpress")
service = CourierUserServiceCollectionImpl(company)

```

```

# Create a courier and place the order
courier1 = Courier(1, "Alice", "123 Main St", "Bob", "456 Elm St", 2.5, "In Transit", "2025-04-10", 101)
tracking = service.place_order(courier1)
print("Tracking Number:", tracking)

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "CourierManagementSystem". It contains several packages and modules:
  - exception:** Contains `invalid\_employee\_id.exception.py` and `tracking\_number\_not\_found\_exception.py`.
  - util:** Contains `db\_conn\_util.py`, `db\_property\_util.py`, `check\_employee\_id.py`, `check\_tracking\_number.py`, and `db\_connection.py`.
  - entities.courier:** Contains `courier.py`.
  - entities.courier\_company:** Contains `courier\_company.py`.
  - dao.courier\_user\_service:** Contains `courier\_user\_service.py`.
  - main:** Contains `main.py`.
  - task1-8:** Contains `task1.py` through `task8.py`.
- Code Editor:** The `main.py` file is open, showing Python code to create a courier and place an order. The output of the run command is visible in the terminal below the editor.
- Terminal Output:**

```
C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\main.py
Courier Inserted into DB
Tracking Number: TRK-1001
Process finished with exit code 0
```
- System Tray:** Shows the date and time as 04-04-2025, 17:02.

## Task 9: 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.

### CourierUserServiceImpl (in dao package)

```

from dao.i_courier_user_service import ICourierUserService

from entities.courier_company import CourierCompany

class CourierUserServiceCollectionImpl(ICourierUserService):

    def __init__(self, company_obj: CourierCompany):
        self.company_obj = company_obj

    def place_order(self, courier):
        self.company_obj.courier_details.append(courier)
        return courier.get_tracking_number()

    def get_order_status(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.get_tracking_number() == tracking_number:
                return courier.get_status()
        return None

```

```

def cancel_order(self, tracking_number):
    for courier in self.company_obj.courier_details:
        if courier.get_tracking_number() == tracking_number:
            self.company_obj.courier_details.remove(courier)
    return True

return False

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "CourierManagementSystem". It contains a "PythonProject" folder with ".venv" and "connectionutil" subfolders, and "dao" and "entities" packages.
- CourierUserServiceImpl.py Content:**

```

class CourierUserServiceImpl(ICourierUserService):
    def get_order_status(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.get_tracking_number() == tracking_number:
                return courier.get_status()
        return None

    def cancel_order(self, tracking_number):
        for courier in self.company_obj.courier_details:
            if courier.get_tracking_number() == tracking_number:
                self.company_obj.courier_details.remove(courier)
        return True

    return False

```
- Run Tab:** Shows the command run: C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:/Users/Nithyashree/PycharmProjects/PythonProject/dao/CourierUserServiceImpl.py
- Bottom Status Bar:** Displays system information like temperature (86°F), battery level, network, and date/time (04-04-2025).

2.Create CourierAdminService Impl class which inherits from CourierUserServiceImpl and implements ICourierAdminService interface.

#### CourierAdminServiceImpl (in dao package)

```

from dao.courier_user_service_impl import CourierUserServiceImpl
from dao.i_courier_admin_service import ICourierAdminService

class CourierAdminServiceImpl(CourierUserServiceImpl, ICourierAdminService):

    def __init__(self, company_obj):
        super().__init__(company_obj)

    def view_all_couriers(self):
        return self.company_obj.courier_details

```

The screenshot shows the PyCharm IDE interface. The project tree on the left displays a PythonProject named 'CourierManagementSystem' with several packages and files under 'dao' and 'entities'. The code editor window on the right shows the file 'CourierAdminServiceImpl.py' with the following content:

```
1 from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
2 from dao.i_courier_admin_service import ICourierAdminService
3
4 class CourierAdminServiceImpl(CourierUserServiceCollectionImpl, ICourierAdminService):
5     def __init__(self, company_obj):
6         super().__init__(company_obj)
7
8     def view_all_couriers(self):
9         return self.company_obj.courier_details
```

The status bar at the bottom indicates the file path 'C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject\dao\CourierAdminServiceImpl.py', encoding 'UTF-8', and Python version '3.13 (PythonProject)'.

3. Create CourierAdminServiceCollectionImpl class which inherits from CourierUserServiceCollectionImpl and implements ICourierAdminService interface.

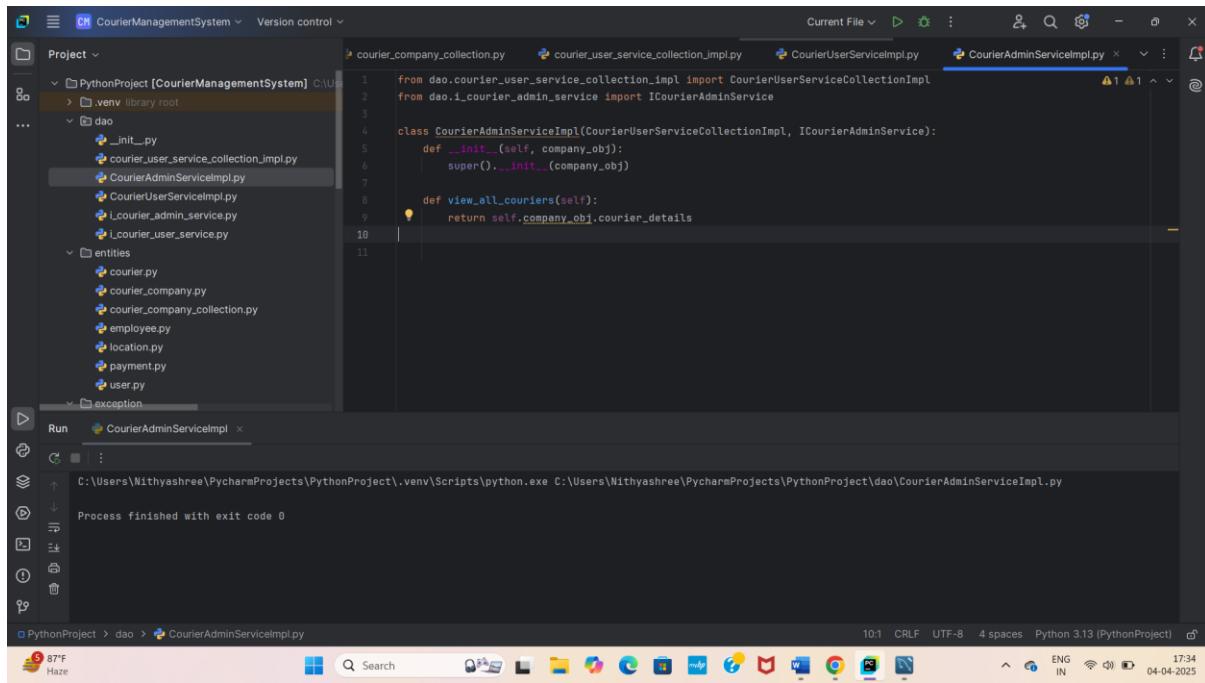
#### CourierAdminServiceCollectionImpl (in dao package)

```
from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
from dao.i_courier_admin_service import ICourierAdminService

class CourierAdminServiceCollectionImpl(CourierUserServiceCollectionImpl,
ICourierAdminService):

    def __init__(self, company_obj):
        super().__init__(company_obj)

    def view_all_couriers(self):
        return self.company_obj.courier_details
```



```
from dao.courier_user_service_collection_impl import CourierUserServiceCollectionImpl
from dao.i_courier_admin_service import ICourierAdminService

class CourierAdminServiceImpl(CourierUserServiceCollectionImpl, ICourierAdminService):
    def __init__(self, company_obj):
        super().__init__(company_obj)

    def view_all_couriers(self):
        return self.company_obj.courier_details
```

## Task 10: Database Interaction

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

```
import mysql.connector
import configparser
class DBConnection:
    connection = None
    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            config = configparser.ConfigParser()
            config.read('db.properties')
            DBConnection.connection = mysql.connector.connect(
                host=config['mysql']['host'],
                user=config['mysql']['user'],
                password=config['mysql']['password'],
                database=config['mysql']['database']
            )
        return DBConnection.connection
```

The screenshot shows the PyCharm IDE interface with the 'CourierManagementSystem' project open. The left sidebar displays the project structure, including a 'connectionutil' folder containing 'db.properties' and 'DBConnection.py', and a 'dao' folder containing several Python files like '\_init\_.py', 'courier\_user\_service\_collection\_implementation.py', 'CourierAdminServiceImpl.py', 'CourierServiceDb.py', 'CourierUserServiceImpl.py', 'I\_courier\_admin\_service.py', 'I\_courier\_user\_service.py', 'entities/courier.py', 'entities/courier\_company.py', 'entities/courier\_company\_collection.py', and 'entities/employee.py'. The main editor window shows the 'DBConnection.py' file with the following code:

```

4 class DBConnection: 5 usages
5     @staticmethod 1 usage
6         def get_connection():
7             if DBConnection.connection is None:
8                 config = configparser.ConfigParser()
9                 config.read('connectionutil/db.properties')
10
11                 DBConnection.connection = mysql.connector.connect(
12                     host=config['mysql']['host'],
13                     user=config['mysql']['user'],
14                     password=config['mysql']['password'],
15                     database=config['mysql']['database']
16                 )
17
18             return DBConnection.connection
19
20

```

The 'Run' tab at the bottom shows the command run: C:\Users\Wityashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Wityashree\PycharmProjects\PythonProject\connectionutil\DBConnection.py, and the output: Process finished with exit code 0.

2. 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.

```

from connectionutil.DBConnection import DBConnection
class CourierServiceDb:
    connection = DBConnection.get_connection()
    @staticmethod
    def insert_courier(courier):
        try:
            cursor = CourierServiceDb.connection.cursor()
            sql = "INSERT INTO courier (SenderId, SenderAddress, ReceiverName,
ReceiverAddress, Weight, Status, TrackingNumber, DeliveryDate, UserID) VALUES (%s, %s, %s,
%s, %s, %s, %s, %s, %s)"
            values = (
                courier.get_sender_name(),
                courier.get_sender_address(),
                courier.get_receiver_name(),
                courier.get_receiver_address(),
                courier.get_weight(),
                courier.get_status(),
                courier.get_tracking_number(),
                courier.get_delivery_date(),
                courier.get_user_id()
            )
            cursor.execute(sql, values)
            CourierServiceDb.connection.commit()
            print("Courier inserted successfully.")
        except Exception as e:

```

```

        print("Error inserting courier into DB:", e)
    # Add update_courier_status, get_courier_by_tracking, etc.

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "CourierManagementSystem". It contains a "connectionutil" folder with "db.properties" and "DBConnection.py", and a "dao" folder containing "CourierServiceDb.py", "CourierUserServiceImpl.py", "LCourierAdminService.py", and "LCourierUserService.py". There are also "entities" and "library root" folders.
- CourierServiceDb.py Content:**

```

3  class CourierServiceDb: 2 usages
7      def insert_courier(courier):
19          cursor.execute(sql, values)
20          CourierServiceDb.connection.commit()
21          print("Courier inserted successfully.")
22      except Exception as e:
23          print("Error inserting courier into DB:", e)
24
25      # Add update_courier_status, get_courier_by_tracking, etc.
26
27
28
29
30

```
- Run Tab:** Shows the command run: `C:\Users\Wityashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Wityashree\PycharmProjects\PythonProject\connectionutil\DBConnection.py` and the output: "Process finished with exit code 0".
- Bottom Status Bar:** Shows the system status: 30:1 CRLF, UTF-8, 4 spaces, Python 3.13 (PythonProject), ENG IN, 17:56, 04-04-2025.

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

```
from connectionutil.DBConnection import DBConnection
```

```

def display_orders(user_id):
    """
    Fetch and display all orders for a given user ID.
    """

    connection = DBConnection.get_connection()
    if connection is None:
        print("Failed to connect to the database.")
        return
    try:
        cursor = connection.cursor()
        # Assuming Order table has OrderID, DeliveryAddress, and Status columns
        query = "SELECT OrderID, DeliveryAddress FROM Orders WHERE UserID = %s"
        cursor.execute(query, (user_id,))
        orders = cursor.fetchall()
        if not orders:
            print(f" No orders found for User ID {user_id}.")
            return
        print(f"\n Orders for User ID {user_id}:")
        for order in orders:
            print(f" Order ID: {order[0]}, Destination: {order[1]}")
    except Exception as err:

```

```

print(f" Error fetching orders: {err}")

finally:
    if 'cursor' in locals():
        cursor.close()
    if 'connection' in locals():
        connection.close()
# Example Usage
user_id = input("Enter User ID: ").strip()
display_orders(user_id)

```

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'PythonProject [CourierManagementSystem]'. The 'Run' tool window at the bottom contains a terminal session. The terminal output shows:

```

C:\Users\Nithyashree\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:/Users/Nithyashree/PycharmProjects/PythonProject/dao/CourierServiceDb.py
Enter User ID: 1
Loading config from: C:/Users/Nithyashree/PycharmProjects/PythonProject/connectionutil/db.properties

Orders for User ID 1:
+ Order ID: 1, Destination: 123 Main St, Saidapet, TN 10001

Process finished with exit code 0

```

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

```
from connectionutil.DBConnection import DBConnection
```

```
def get_parcel_history(tracking_number):
```

```
    """
```

Retrieve and display the delivery history of a specific parcel using tracking number.

```
    """
```

```
    connection = DBConnection.get_connection()
```

```
    if connection is None:
```

```
        print(" Failed to connect to the database.")
```

```
    return
```

```
try:
```

```
cursor = connection.cursor()

query = """
    SELECT TrackingNumber, Status, DeliveryDate
    FROM courier
    WHERE TrackingNumber = %s
"""

cursor.execute(query, (tracking_number,))

history = cursor.fetchall()

if not history:
    print(f" No delivery history found for Tracking Number {tracking_number}.")
    return

print(f"\n Delivery History for Tracking Number {tracking_number}:")
for record in history:
    print(f" Status: {record[1]}, Delivery Date: {record[2]}")

except Exception as err:
    print(f" Error retrieving parcel history: {err}")

finally:
    if 'cursor' in locals():
        cursor.close()
    if 'connection' in locals():
        connection.close()

# Example usage

tracking_number = input("Enter Tracking Number: ").strip()
get_parcel_history(tracking_number)
```

```

def get_parcel_history(tracking_number):
    usage
    except Exception as err:
        print(f" Error retrieving parcel history: {err}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'connection' in locals():
            connection.close()
    # Example usage
    tracking_number = input("Enter Tracking Number: ").strip()
    get_parcel_history(tracking_number)

```

5. Generate and display reports using data retrieved from the database (e.g., shipment status report, revenue report).

```
from connectionutil.DBConnection import DBConnection
```

```
def shipment_status_report():
```

```
    """
```

Generate a shipment status report showing counts of each status.

```
    """
```

```
connection = DBConnection.get_connection()
```

```
if connection is None:
```

```
    print("Failed to connect to the database.")
```

```
    return
```

```
try:
```

```
    cursor = connection.cursor()
```

```
    query = """
```

```
        SELECT Status, COUNT(*) AS Count
```

```
        FROM courier
```

```
        GROUP BY Status
```

```
    """
```

```
    cursor.execute(query)
```

```
    results = cursor.fetchall()
```

```

print("\n Shipment Status Report:")

for row in results:

    print(f" {row[0]}: {row[1]} shipments")

except Exception as err:

    print(f" Error generating shipment status report: {err}")

finally:

    if 'cursor' in locals():

        cursor.close()

    if 'connection' in locals():

        connection.close()

shipment_status_report()

```

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "CourierManagementSystem". It contains several modules like db.properties, connectionutil\db.properties, CourierServiceDb.py, get\_parcel\_history.py, and shipment\_status\_report.py.
- Current File:** The file "shipment\_status\_report.py" is open. The code defines a function `shipment\_status\_report` which prints a usage message, handles exceptions, and closes database connections.
- Run Output:** The bottom pane shows the terminal output of the run command "shipment\_status\_report". It starts by loading configuration from "C:\Users\Withyashree\PycharmProjects\PythonProject\connectionutil\db.properties". Then it prints a summary of the shipment status report:

```

Shipment Status Report:
Delivered: 2 shipments
Pending: 2 shipments
In Transit: 2 shipments

```

- System Status:** The taskbar at the bottom shows various system icons, including weather (86°F Haze), search, file explorer, and browser.