**Assignment-Courier Management System**

**Student Name:Sugandan E**

**Coding**

**Task 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.

def check\_delivery\_status(order\_status):

if order\_status == "Delivered":

print("The order has been delivered.")

elif order\_status == "Processing":

print("The order is still being processed.")

elif order\_status == "Cancelled":

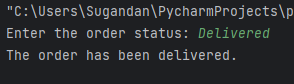
print("The order has been cancelled.")

else:

print("Invalid order status. Please check the status again.")

order\_status\_input = input("Enter the order status: ")

check\_delivery\_status(order\_status\_input)



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

def categorize\_parcel(weight):

categories = {

"Light": lambda x: x < 5,

"Medium": lambda x: 5 <= x < 10,

"Heavy": lambda x: x >= 10

}

for category, condition in categories.items():

if condition(weight):

print(f"The parcel is categorized as {category}.")

break

else:

print("Invalid weight. Please check the weight again.")

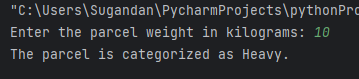
try:

parcel\_weight = float(input("Enter the parcel weight in kilograms: "))

categorize\_parcel(parcel\_weight)

except ValueError:

print("Invalid input. Please enter a valid numerical weight.")



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

employee\_username = "employee"

employee\_password = "employee123"

customer\_username = "customer"

customer\_password = "customer123"

username\_input = input("Enter username: ")

password\_input = input("Enter password: ")

if username\_input == employee\_username and password\_input == employee\_password:

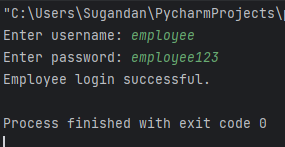
print("Employee login successful.")

elif username\_input == customer\_username and password\_input == customer\_password:

print("Customer login successful.")

else:

print("Invalid username or password. Please try again.")



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

class Courier:

def \_\_init\_\_(self, name, proximity, load\_capacity):

self.name = name

self.proximity = proximity

self.load\_capacity = load\_capacity

self.shipments = []

def assign\_shipment(self, shipment):

self.shipments.append(shipment)

print(f"Assigned {shipment['weight']} kg shipment to {self.name}.")

courier1 = Courier("Courier A", 10, 20)

courier2 = Courier("Courier B", 5, 15)

courier3 = Courier("Courier C", 12, 25)

shipments = [

{"weight": 18, "destination": "Location X"},

{"weight": 8, "destination": "Location Y"},

{"weight": 22, "destination": "Location Z"}

]

for shipment in shipments:

suitable\_couriers = [courier for courier in [courier1, courier2, courier3] if

courier.proximity <= 10 and courier.load\_capacity >= shipment["weight"]]

if suitable\_couriers:

closest\_courier = min(suitable\_couriers, key=lambda x: x.proximity)

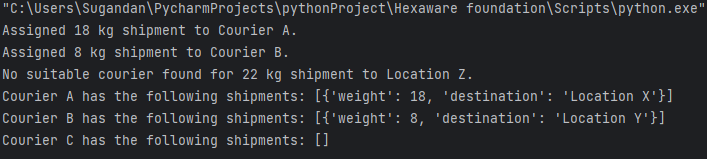
closest\_courier.assign\_shipment(shipment)

else:

print(f"No suitable courier found for {shipment['weight']} kg shipment to {shipment['destination']}.")

for courier in [courier1, courier2, courier3]:

print(f"{courier.name} has the following shipments: {courier.shipments}")



**Task 2: Loops and Iteration**

1. Write a Python program that uses a for loop to display all the orders for a specific customer.

class Order:

def \_\_init\_\_(self, order\_id, customer\_name, status):

self.order\_id = order\_id

self.customer\_name = customer\_name

self.status = status

orders = [

Order(1, "CustomerA", "Processing"),

Order(2, "CustomerB", "Delivered"),

Order(3, "CustomerA", "Cancelled"),

Order(4, "CustomerC", "Processing"),

Order(5, "CustomerA", "Delivered"),

]

def display\_orders\_for\_customer(customer\_name):

customer\_orders = [order for order in orders if order.customer\_name == customer\_name]

if customer\_orders:

print(f"Orders for {customer\_name}:")

for order in customer\_orders:

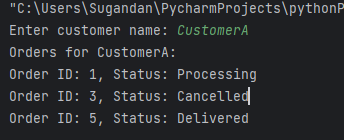
print(f"Order ID: {order.order\_id}, Status: {order.status}")

else:

print(f"No orders found for {customer\_name}.")

customer\_name\_input = input("Enter customer name: ")

display\_orders\_for\_customer(customer\_name\_input)



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

import time

class Courier:

def \_\_init\_\_(self, name, current\_location, destination):

self.name = name

self.current\_location = current\_location

self.destination = destination

def update\_location(self):

if self.current\_location < self.destination:

self.current\_location += 1

print(f"{self.name}'s current location: {self.current\_location}")

else:

print(f"{self.name} has reached the destination.")

courier\_name = "Courier A"

starting\_location = 0

destination\_location = 10

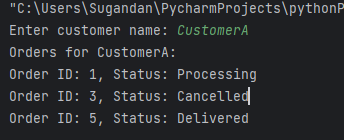
courier = Courier(courier\_name, starting\_location, destination\_location)

while courier.current\_location < courier.destination:

courier.update\_location()

time.sleep(1)

print("Tracking completed.")



**Task 3: Arrays and Data Structures**

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

class Parcel:

def \_\_init\_\_(self, tracking\_id):

self.tracking\_id = tracking\_id

self.tracking\_history = []

def update\_location(self, location):

timestamp = time.strftime("%Y-%m-%d %H:%M:%S")

update\_entry = {"timestamp": timestamp, "location": location}

self.tracking\_history.append(update\_entry)

import time

parcel\_tracking\_id = "ABC123"

parcel = Parcel(parcel\_tracking\_id)

locations = ["Warehouse", "In Transit", "Local Distribution Center", "Delivered"]

for location in locations:

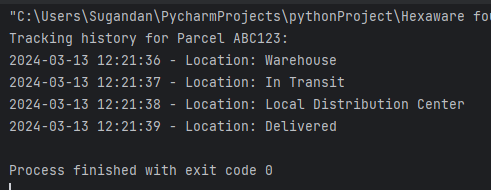
parcel.update\_location(location)

time.sleep(1)

print(f"Tracking history for Parcel {parcel.tracking\_id}:")

for entry in parcel.tracking\_history:

print(f"{entry['timestamp']} - Location: {entry['location']}")



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

import math

class Courier:

def \_\_init\_\_(self, name, current\_location, availability):

self.name = name

self.current\_location = current\_location

self.availability = availability

def find\_nearest\_courier(new\_order\_location, couriers):

available\_couriers = [courier for courier in couriers if courier.availability]

if not available\_couriers:

print("No available couriers.")

return None

nearest\_courier = min(available\_couriers, key=lambda x: abs(x.current\_location - new\_order\_location))

return nearest\_courier

couriers = [

Courier("Courier A", 5, True),

Courier("Courier B", 8, True),

Courier("Courier C", 12, False),

Courier("Courier D", 3, True),

]

new\_order\_location = 7

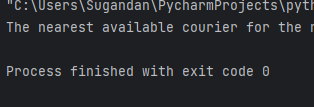
nearest\_courier = find\_nearest\_courier(new\_order\_location, couriers)

if nearest\_courier:

print(f"The nearest available courier for the new order is {nearest\_courier.name}.")

else:

print("No available couriers.")



**Task 4: Strings,2d Arrays, user defined functions,Hashmap**

1. 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.

class ParcelTracker:

def \_\_init\_\_(self):

self.tracking\_data = [

["ABC123", "In Transit"],

["XYZ456", "Out for Delivery"],

["123DEF", "Processing"],

["789GHI", "Delivered"],

]

def get\_tracking\_status(self, tracking\_number):

for item in self.tracking\_data:

if item[0] == tracking\_number:

return item[1]

return "Tracking number not found."

def simulate\_tracking\_process(self, tracking\_number):

status = self.get\_tracking\_status(tracking\_number)

if status == "In Transit":

print(f"Parcel {tracking\_number} is currently in transit.")

elif status == "Out for Delivery":

print(f"Parcel {tracking\_number} is out for delivery.")

elif status == "Processing":

print(f"Parcel {tracking\_number} is still processing.")

elif status == "Delivered":

print(f"Parcel {tracking\_number} has been delivered.")

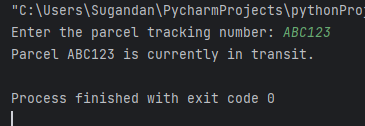
else:

print(f"Invalid tracking number: {tracking\_number}")

parcel\_tracker = ParcelTracker()

user\_tracking\_number = input("Enter the parcel tracking number: ")

parcel\_tracker.simulate\_tracking\_process(user\_tracking\_number)



1. 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

def validate\_customer\_information(data, detail):

if detail == "name":

if data.isalpha() and data.istitle():

return True

else:

return False

elif detail == "address":

if data.isalnum() or data.replace(" ", "").isalpha():

return True

else:

return False

elif detail == "phone\_number":

phone\_number\_pattern = re.compile(r'^\d{3}-\d{3}-\d{4}$')

if phone\_number\_pattern.match(data):

return True

else:

return False

else:

return False

customer\_name = "John Doe"

customer\_address = "123 Main Street"

customer\_phone\_number = "555-123-4567"

if validate\_customer\_information(customer\_name, "name"):

print("Name is valid.")

else:

print("Invalid name.")

if validate\_customer\_information(customer\_address, "address"):

print("Address is valid.")

else:

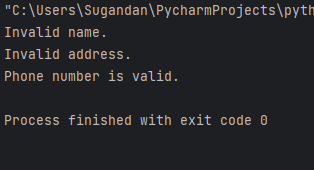
print("Invalid address.")

if validate\_customer\_information(customer\_phone\_number, "phone\_number"):

print("Phone number is valid.")

else:

print("Invalid phone number.")



1. 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.

def format\_address(street, city, state, zip\_code):

formatted\_street = ' '.join(word.capitalize() for word in street.split())

formatted\_city = city.capitalize()

formatted\_state = state.upper()

formatted\_zip\_code = zip\_code[:5] + '-' + zip\_code[5:] if len(zip\_code) == 9 else zip\_code

formatted\_address = f"{formatted\_street}, {formatted\_city}, {formatted\_state} {formatted\_zip\_code}"

return formatted\_address

street\_input = input("Enter street address: ")

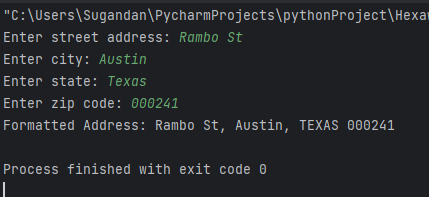
city\_input = input("Enter city: ")

state\_input = input("Enter state: ")

zip\_code\_input = input("Enter zip code: ")

formatted\_address = format\_address(street\_input, city\_input, state\_input, zip\_code\_input)

print("Formatted Address:", formatted\_address)



1. 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\_confirmation\_email(customer\_name, order\_number, delivery\_address):

expected\_delivery\_date = datetime.date.today() + datetime.timedelta(days=2)

email\_content = f"""

Subject: Order Confirmation - Order

Dear {customer\_name},

Thank you for placing an order with us! Your order #{order\_number} has been confirmed.

Order Details:

- Order Number: {order\_number}

- Delivery Address: {delivery\_address}

- Expected Delivery Date: {expected\_delivery\_date}

If you have any questions or concerns, please feel free to contact our customer support.

Thank you for choosing our service!

Best regards,

Your Company Name

"""

return email\_content

customer\_name\_input = input("Enter customer's name: ")

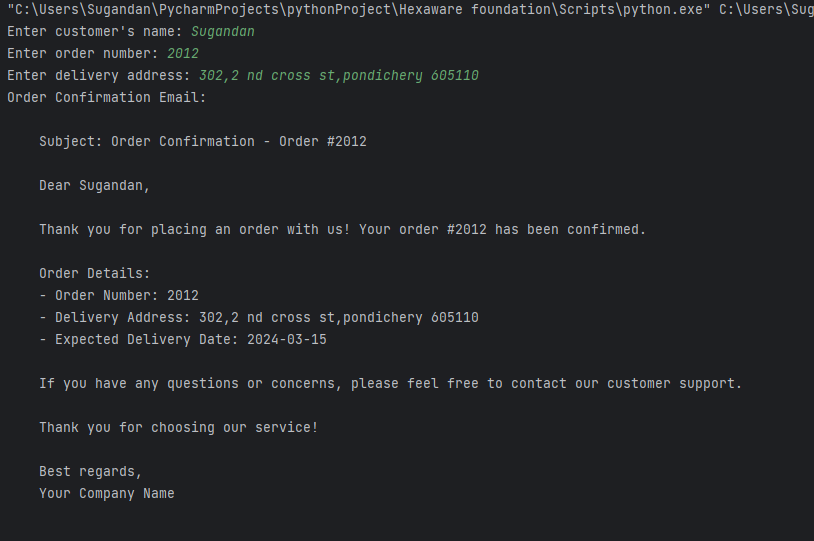
order\_number\_input = input("Enter order number: ")

delivery\_address\_input = input("Enter delivery address: ")

order\_confirmation\_email = generate\_order\_confirmation\_email(customer\_name\_input, order\_number\_input, delivery\_address\_input)

print("Order Confirmation Email:")

print(order\_confirmation\_email)



1. 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.

def calculate\_shipping\_cost(source\_address, destination\_address, parcel\_weight):

BASE\_COST = 5

DISTANCE\_COST\_FACTOR = 0.1

WEIGHT\_COST\_FACTOR = 0.2

distance\_km = 50

distance\_cost = distance\_km \* DISTANCE\_COST\_FACTOR

weight\_cost = parcel\_weight \* WEIGHT\_COST\_FACTOR

total\_cost = BASE\_COST + distance\_cost + weight\_cost

return total\_cost

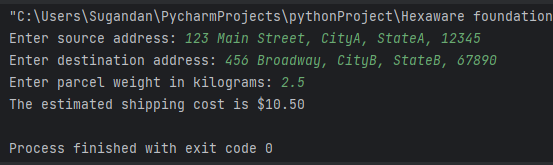
source\_address\_input = input("Enter source address: ")

destination\_address\_input = input("Enter destination address: ")

parcel\_weight\_input = float(input("Enter parcel weight in kilograms: "))

shipping\_cost = calculate\_shipping\_cost(source\_address\_input, destination\_address\_input, parcel\_weight\_input)

print(f"The estimated shipping cost is ${shipping\_cost:.2f}")



1. 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

def generate\_secure\_password(length=12):

uppercase\_letters = string.ascii\_uppercase

lowercase\_letters = string.ascii\_lowercase

digits = string.digits

special\_characters = string.punctuation

all\_characters = uppercase\_letters + lowercase\_letters + digits + special\_characters

password = random.choice(uppercase\_letters) + random.choice(lowercase\_letters) + random.choice(digits) + random.choice(special\_characters)

for \_ in range(length - 4):

password += random.choice(all\_characters)

password\_list = list(password)

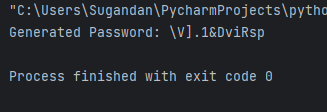
random.shuffle(password\_list)

password = ''.join(password\_list)

return password

generated\_password = generate\_secure\_password()

print("Generated Password:", generated\_password)



1. 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.

def find\_similar\_addresses(address, addresses, similarity\_threshold=0.8):

similar\_addresses = []

for other\_address in addresses:

similarity\_score = calculate\_similarity(address, other\_address)

if similarity\_score >= similarity\_threshold:

similar\_addresses.append(other\_address)

return similar\_addresses

def calculate\_similarity(address1, address2):

address1 = address1.lower()

address2 = address2.lower()

intersection = set(address1.split()) & set(address2.split())

union = set(address1.split()) | set(address2.split())

similarity\_score = len(intersection) / len(union)

return similarity\_score

all\_addresses = [

"123 Main Street, CityA, StateA, 12345",

"124 Main St, CityA, StateA, 12345",

"456 Broadway, CityB, StateB, 67890",

"789 Elm St, CityC, StateC, 98765",

]

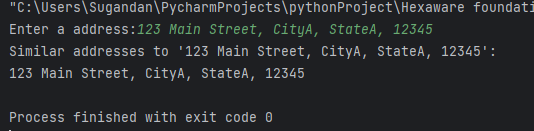
input\_address = input(“Enter a address:”)

similar\_addresses = find\_similar\_addresses(input\_address, all\_addresses)

print(f"Similar addresses to '{input\_address}':")

for similar\_address in similar\_addresses:

print(similar\_address)



**Task 5: Object Oriented Programming**

**Scope : Entity classes/Models/POJO, Abstraction/Encapsulation**

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

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

4.Location Class Variables LocationID , LocationName , Address

1. CourierCompany Class Variables companyName , courierDetails -collection of Courier Objects, employeeDetailscollection of Employee Objects, locationDetails - collection of Location Objects. 6. Payment Class: Variables PaymentID long, CourierID long, Amount double, PaymentDate Date

class User:

def \_\_init\_\_(self, userID, userName, email, password, contactNumber, address):

self.\_\_userID = userID

self.\_\_userName = userName

self.\_\_email = email

self.\_\_password = password

self.\_\_contactNumber = contactNumber

self.\_\_address = address

def get\_userID(self):

return self.\_\_userID

def set\_userID(self, userID):

self.\_\_userID = userID

def get\_userName(self):

return self.\_\_userName

def set\_userName(self, userName):

self.\_\_userName = userName

def get\_email(self):

return self.\_\_email

def set\_email(self, email):

self.\_\_email = email

def get\_password(self):

return self.\_\_password

def set\_password(self, password):

self.\_\_password = password

def get\_contactNumber(self):

return self.\_\_contactNumber

def set\_contactNumber(self, contactNumber):

self.\_\_contactNumber = contactNumber

def get\_address(self):

return self.\_\_address

def set\_address(self, address):

self.\_\_address = address

def \_\_str\_\_(self):

return f"UserID: {self.\_\_userID}, UserName: {self.\_\_userName}, Email: {self.\_\_email}, Password: {self.\_\_password}, ContactNumber: {self.\_\_contactNumber}, Address: {self.\_\_address}"

class Courier:

def \_\_init\_\_(self, courierID, senderName, senderAddress, receiverName, receiverAddress, weight, status,

trackingNumber, deliveryDate, userId):

self.\_\_courierID = courierID

self.\_\_senderName = senderName

self.\_\_senderAddress = senderAddress

self.\_\_receiverName = receiverName

self.\_\_receiverAddress = receiverAddress

self.\_\_weight = weight

self.\_\_status = status

self.\_\_trackingNumber = trackingNumber

self.\_\_deliveryDate = deliveryDate

self.\_\_userId = userId

def get\_courierID(self):

return self.\_\_courierID

def set\_courierID(self, courierID):

self.\_\_courierID = courierID

def get\_senderName(self):

return self.\_\_senderName

def set\_senderName(self, senderName):

self.\_\_senderName = senderName

def get\_senderAddress(self):

return self.\_\_senderAddress

def set\_senderAddress(self, senderAddress):

self.\_\_senderAddress = senderAddress

def get\_receiverName(self):

return self.\_\_receiverName

def set\_receiverName(self, receiverName):

self.\_\_receiverName = receiverName

def get\_receiverAddress(self):

return self.\_\_receiverAddress

def set\_receiverAddress(self, receiverAddress):

self.\_\_receiverAddress = receiverAddress

def get\_weight(self):

return self.\_\_weight

def set\_weight(self, weight):

self.\_\_weight = weight

def get\_status(self):

return self.\_\_status

def set\_status(self, status):

self.\_\_status = status

def get\_trackingNumber(self):

return self.\_\_trackingNumber

def set\_trackingNumber(self, trackingNumber):

self.\_\_trackingNumber = trackingNumber

def get\_deliveryDate(self):

return self.\_\_deliveryDate

def set\_deliveryDate(self, deliveryDate):

self.\_\_deliveryDate = deliveryDate

def get\_userId(self):

return self.\_\_userId

def set\_userId(self, userId):

self.\_\_userId = userId

def \_\_str\_\_(self):

return f"CourierID: {self.\_\_courierID}, SenderName: {self.\_\_senderName}, SenderAddress: {self.\_\_senderAddress}, ReceiverName: {self.\_\_receiverName}, ReceiverAddress: {self.\_\_receiverAddress}, Weight: {self.\_\_weight}, Status: {self.\_\_status}, TrackingNumber: {self.\_\_trackingNumber}, DeliveryDate: {self.\_\_deliveryDate}, UserID: {self.\_\_userId}"

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 set\_employeeID(self, employeeID):

self.\_\_employeeID = employeeID

def get\_employeeName(self):

return self.\_\_employeeName

def set\_employeeName(self, employeeName):

self.\_\_employeeName = employeeName

def get\_email(self):

return self.\_\_email

def set\_email(self, email):

self.\_\_email = email

def get\_contactNumber(self):

return self.\_\_contactNumber

def set\_contactNumber(self, contactNumber):

self.\_\_contactNumber = contactNumber

def get\_role(self):

return self.\_\_role

def set\_role(self, role):

self.\_\_role = role

def get\_salary(self):

return self.\_\_salary

def set\_salary(self, salary):

self.\_\_salary = salary

def \_\_str\_\_(self):

return f"EmployeeID: {self.\_\_employeeID}, EmployeeName: {self.\_\_employeeName}, Email: {self.\_\_email}, ContactNumber: {self.\_\_contactNumber}, Role: {self.\_\_role}, Salary: {self.\_\_salary}"

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 set\_LocationID(self, LocationID):

self.\_\_LocationID = LocationID

def get\_LocationName(self):

return self.\_\_LocationName

def set\_LocationName(self, LocationName):

self.\_\_LocationName = LocationName

def get\_Address(self):

return self.\_\_Address

def set\_Address(self, Address):

self.\_\_Address = Address

def \_\_str\_\_(self):

return f"LocationID: {self.\_\_LocationID}, LocationName: {self.\_\_LocationName}, Address: {self.\_\_Address}"

class CourierCompany:

def \_\_init\_\_(self, companyName):

self.\_\_companyName = companyName

self.\_\_courierDetails = []

self.\_\_employeeDetails = []

self.\_\_locationDetails = []

def get\_companyName(self):

return self.\_\_companyName

def set\_companyName(self, companyName):

self.\_\_companyName = companyName

def add\_courier(self, courier):

self.\_\_courierDetails.append(courier)

def remove\_courier(self, courier):

self.\_\_courierDetails.remove(courier)

def add\_employee(self, employee):

self.\_\_employeeDetails.append(employee)

def remove\_employee(self, employee):

self.\_\_employeeDetails.remove(employee)

def add\_location(self, location):

self.\_\_locationDetails.append(location)

def remove\_location(self, location):

self.\_\_locationDetails.remove(location)

def \_\_str\_\_(self):

return f"CompanyName: {self.\_\_companyName}, CourierDetails: {self.\_\_courierDetails}, EmployeeDetails: {self.\_\_employeeDetails}, LocationDetails: {self.\_\_locationDetails}"

class Payment:

def \_\_init\_\_(self, PaymentID, CourierID, LocationID, Amount, PaymentDate, EmployeeID):

self.\_\_PaymentID = PaymentID

self.\_\_CourierID = CourierID

self.\_\_LocationID = LocationID

self.\_\_Amount = Amount

self.\_\_PaymentDate = PaymentDate

self.\_\_EmployeeID = EmployeeID

def get\_PaymentID(self):

return self.\_\_PaymentID

def set\_PaymentID(self, PaymentID):

self.\_\_PaymentID = PaymentID

def get\_CourierID(self):

return self.\_\_CourierID

def set\_CourierID(self, CourierID):

self.\_\_CourierID = CourierID

def get\_LocationID(self):

return self.\_\_LocationID

def set\_LocationID(self, LocationID):

self.\_\_LocationID = LocationID

def get\_Amount(self):

return self.\_\_Amount

def set\_Amount(self, Amount):

self.\_\_Amount = Amount

def get\_PaymentDate(self):

return self.\_\_PaymentDate

def set\_PaymentDate(self, PaymentDate):

self.\_\_PaymentDate = PaymentDate

def get\_EmployeeID(self):

return self.\_\_EmployeeID

def set\_EmployeeID(self, EmployeeID):

self.\_\_EmployeeID = EmployeeID

def \_\_str\_\_(self):

return f"PaymentID: {self.\_\_PaymentID}, CourierID: {self.\_\_CourierID}, LocationID: {self.\_\_LocationID}, Amount: {self.\_\_Amount}, PaymentDate: {self.\_\_PaymentDate}, EmployeeID: {self.\_\_EmployeeID}"e}"

**Task 6: Service Provider Interface /Abstract class**

**Create 2 Interface /Abstract class ICourierUserService and ICourierAdminService interface ICourierUserService { // Customer-related functions**

**placeOrder()**

/\*\* Place a new courier order. \* @param courierObj Courier object created using values entered by users \* @return The unique tracking number for the courier order . Use a static variable to generate unique tracking number. Initialize the static variable in Courier class with some random value. Increment the static variable each time in the constructor to generate next values.

**getOrderStatus();**

/\*\*Get the status of a courier order. \*@param trackingNumber The tracking number of the courier order. \* @return The status of the courier order (e.g., yetToTransit, In Transit, Delivered). \*/

**cancelOrder()**

/\*\* Cancel a courier order. \* @param trackingNumber The tracking number of the courier order to be canceled. \* @return True if the order was successfully canceled, false otherwise.\*/

**getAssignedOrder();** /\*\* Get a list of orders assigned to a specific courier staff member \* @param courierStaffId The ID of the courier staff member. \* @return A list of courier orders assigned to the staff member.\*/

// Admin functions

**ICourierAdminService**

**int addCourierStaff(Employee obj);**

/\*\* Add a new courier staff member to the system. \* @param name The name of the courier staff member. \* @param contactNumber The contact number of the courier staff member. \* @return The ID of the newly added courier staff member. \*/

from abc import ABC, abstractmethod

class ICourierUserService(ABC):

@abstractmethod

def placeOrder(self, courierObj):

pass

@abstractmethod

def getOrderStatus(self, trackingNumber):

pass

@abstractmethod

def cancelOrder(self, trackingNumber):

pass

@abstractmethod

def getAssignedOrder(self, courierStaffId):

pass

class ICourierAdminService(ABC):

@abstractmethod

def addCourierStaff(self, name, contactNumber):

Pass

**Task 7: Exception Handling**

**(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,thow & throws keyword usage)**

**Define the following custom exceptions and throw t**hem in methods whenever needed . Handle all the excpetionsin main method,

1. **TrackingNumberNotFoundException :**throw this exception when user try to withdraw amount or transfer amount to another acco
2. **InvalidEmployeeIdException** throw this exception when id entered for the employee not existing in the system

from abc import ABC, abstractmethod

class TrackingNumberNotFoundException(Exception):

pass

class InvalidEmployeeIdException(Exception):

pass

class ICourierUserService(ABC):

@abstractmethod

def placeOrder(self, courierObj):

pass

@abstractmethod

def getOrderStatus(self, trackingNumber):

pass

@abstractmethod

def cancelOrder(self, trackingNumber):

pass

@abstractmethod

def getAssignedOrder(self, courierStaffId):

pass

class ICourierAdminService(ABC):

@abstractmethod

def addCourierStaff(self, name, contactNumber):

pass

def getCouriersByEmployee(self, employee\_id):

try:

cursor = self.connection.cursor()

sql\_query = """SELECT \*

FROM Couriers

WHERE EmployeeID = ?"""

cursor.execute(sql\_query, (employee\_id,))

couriers = cursor.fetchall()

print("Couriers retrieved successfully.")

return couriers

except exception.InvalidEmployeeIdException as ex:

print(f"Error retrieving assigned orders: {ex}")

except Exception as ex:

print(f"Error retrieving assigned orders: {ex}")

finally:

cursor.close()

def updateCourierStatus(self, trackingNumber, newStatus):

try:

cursor = self.connection.cursor()

sql\_query = """UPDATE Couriers

SET Status = ?

WHERE TrackingNumber = ?"""

cursor.execute(sql\_query, (newStatus, trackingNumber))

self.connection.commit()

print("Order cancelled successfully.")

except exception.TrackingNumberNotFoundException as ex:

print(f"Error cancelling order: {ex}")

except Exception as ex:

print(f"Error cancelling order: {ex}")

finally:

cursor.close()

def addCourierStaff(self, empID, name, email, contact\_number, role, salary):

try:

cursor = self.connection.cursor()

sql\_query = """INSERT INTO Employees (EmployeeID,Name, Email, ContactNumber, Role, Salary)

VALUES (?,?, ?, ?, ?, ?)"""

cursor.execute(sql\_query, (empID, name, email, contact\_number, role, salary))

self.connection.commit()

print("Courier staff added successfully.")

except Exception as ex:

print(f"Error adding courier staff: {ex}")

finally:

cursor.close()

def insertOrder(self, courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight, status,

tracking\_number, delivery\_date, location\_id, employee\_id, service\_id):

try:

cursor = self.connection.cursor()

sql\_query = """INSERT INTO Couriers (CourierID, SenderName, SenderAddress, ReceiverName, ReceiverAddress, Weight, Status, TrackingNumber, DeliveryDate, LocationID, EmployeeID, ServiceID)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"""

cursor.execute(sql\_query,

(courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight, status,

tracking\_number, delivery\_date, location\_id, employee\_id, service\_id))

self.connection.commit()

print("Order inserted successfully.")

except Exception as ex:

print(f"Error inserting order: {ex}")

finally:

cursor.close()

**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.

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

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

**class ICourierUserService(ABC):**

**@abstractmethod**

**def placeOrder(self, courierObj):**

**pass**

**@abstractmethod**

**def getOrderStatus(self, trackingNumber):**

**pass**

**@abstractmethod**

**def cancelOrder(self, trackingNumber):**

**pass**

**@abstractmethod**

**def getAssignedOrder(self, courierStaffId):**

**pass**

**class ICourierAdminService(ICourierUserService):**

**@abstractmethod**

**def addCourierStaff(self, name, contactNumber):**

**pass**

**class CourierUserServiceImpl(ICourierUserService):**

**def \_\_init\_\_(self, company\_name):**

**self.companyObj = CourierCompany(company\_name)**

**def placeOrder(self, courierObj):**

**self.companyObj.add\_courier(courierObj)**

**print("Order placed successfully.")**

**def getOrderStatus(self, trackingNumber):**

**for courier in self.companyObj.courier\_details:**

**if courier.trackingNumber == trackingNumber:**

**return courier.status**

**return "Order not found."**

**def cancelOrder(self, trackingNumber):**

**for courier in self.companyObj.courier\_details:**

**if courier.trackingNumber == trackingNumber:**

**self.companyObj.remove\_courier(courier)**

**print("Order canceled successfully.")**

**return**

**print("Order not found.")**

**def getAssignedOrder(self, courierStaffId):**

**assigned\_orders = []**

**for courier in self.companyObj.courier\_details:**

**if courier.employeeId == courierStaffId:**

**assigned\_orders.append(courier)**

**return assigned\_orders**

**class CourierAdminServiceImpl(CourierUserServiceImpl, ICourierAdminService):**

**def addCourierStaff(self, name, contactNumber):**

**new\_employee\_id = len(self.companyObj.employee\_details) + 1**

**new\_employee = Employee(new\_employee\_id, name, None, contactNumber, None, None)**

**self.companyObj.add\_employee(new\_employee)**

**print("Courier staff added successfully.")**

**class CourierAdminServiceCollectionImpl(CourierUserServiceImpl, ICourierAdminService):**

**def \_\_init\_\_(self, company\_name):**

**super().\_\_init\_\_(company\_name)**

**def addCourierStaff(self, name, contactNumber):**

**new\_employee\_id = len(self.companyObj.employee\_details) + 1**

**new\_employee = Employee(new\_employee\_id, name, None, contactNumber, None, None)**

**self.companyObj.add\_employee(new\_employee)**

**print("Courier staff added successfully.")**

**Task 9: Database Interaction Connect your application to the SQL database for the Courier Management System**

1. Write code to establish a connection to your SQL database. Create a class DBConnection in a package connectionutil with a static variable connection of Type Connection and a static method getConnection() which returns connection. Connection properties supplied in the connection string should be read from a property file.
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.
3. Include methods to insert, update, and retrieve data from the database (e.g., inserting a new order, updating courier status).
4. Implement a feature to retrieve and display the delivery history of a specific parcel by querying the database. 1. Generate and display reports using data retrieved from the database (e.g., shipment status report, revenue report).

**import dao**

**import exception**

**import entity**

**class CourierServiceDb:**

**connection = dao.connect\_to\_sql\_server()**

**def \_\_init\_\_(self):**

**self.connection = dao.connect\_to\_sql\_server()**

**def cancelOrder(self, tracking\_number):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """UPDATE Couriers**

**SET Status = 'Cancelled'**

**WHERE TrackingNumber = ?"""**

**cursor.execute(sql\_query, (tracking\_number,))**

**self.connection.commit()**

**print("Order cancelled successfully.")**

**except Exception as ex:**

**print(f"Error cancelling order: {ex}")**

**finally:**

**cursor.close()**

**def getCouriersByEmployee(self, employee\_id):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT \***

**FROM Couriers**

**WHERE EmployeeID = ?"""**

**cursor.execute(sql\_query, (employee\_id,))**

**couriers = cursor.fetchall()**

**print("Couriers retrieved successfully.")**

**return couriers**

**except exception.InvalidEmployeeIdException as ex:**

**print(f"Error retrieving assigned orders: {ex}")**

**except Exception as ex:**

**print(f"Error retrieving assigned orders: {ex}")**

**finally:**

**cursor.close()**

**def addCourierStaff(self, empID, name, email, contact\_number, role, salary):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """INSERT INTO Employees (EmployeeID,Name, Email, ContactNumber, Role, Salary)**

**VALUES (?,?, ?, ?, ?, ?)"""**

**cursor.execute(sql\_query, (empID, name, email, contact\_number, role, salary))**

**self.connection.commit()**

**print("Courier staff added successfully.")**

**except Exception as ex:**

**print(f"Error adding courier staff: {ex}")**

**finally:**

**cursor.close()**

**def insertOrder(self, courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight, status,**

**tracking\_number, delivery\_date, location\_id, employee\_id, service\_id):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """INSERT INTO Couriers (CourierID, SenderName, SenderAddress, ReceiverName, ReceiverAddress, Weight, Status, TrackingNumber, DeliveryDate, LocationID, EmployeeID, ServiceID)**

**VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"""**

**cursor.execute(sql\_query,**

**(courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight, status,**

**tracking\_number, delivery\_date, location\_id, employee\_id, service\_id))**

**self.connection.commit()**

**print("Order inserted successfully.")**

**except Exception as ex:**

**print(f"Error inserting order: {ex}")**

**finally:**

**cursor.close()**

**def updateCourierStatus(self, trackingNumber, newStatus):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """UPDATE Couriers**

**SET Status = ?**

**WHERE TrackingNumber = ?"""**

**cursor.execute(sql\_query, (newStatus, trackingNumber))**

**self.connection.commit()**

**print("Order cancelled successfully.")**

**except exception.TrackingNumberNotFoundException as ex:**

**print(f"Error cancelling order: {ex}")**

**except Exception as ex:**

**print(f"Error cancelling order: {ex}")**

**finally:**

**cursor.close()**

**def retrieveDeliveryHistory(self, trackingNumber):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT \***

**FROM Couriers**

**WHERE TrackingNumber = ?"""**

**cursor.execute(sql\_query, (trackingNumber,))**

**delivery\_history = cursor.fetchall()**

**print("Delivery history retrieved successfully.")**

**return delivery\_history**

**except Exception as ex:**

**print(f"Error retrieving delivery history: {ex}")**

**finally:**

**cursor.close()**

**def generateShipmentStatusReport(self):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT TrackingNumber, Status**

**FROM Couriers"""**

**cursor.execute(sql\_query)**

**shipment\_status\_report = cursor.fetchall()**

**print("Shipment status report generated successfully.")**

**return shipment\_status\_report**

**except Exception as ex:**

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

**finally:**

**cursor.close()**

**def generateRevenueReport(self):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT SUM(Amount) as TotalRevenue**

**FROM Payments"""**

**cursor.execute(sql\_query)**

**total\_revenue = cursor.fetchone()[0]**

**print("Revenue report generated successfully.")**

**return total\_revenue**

**except Exception as ex:**

**print(f"Error generating revenue report: {ex}")**

**finally:**

**cursor.close()**

**MAIN.py**

**import dao**

**import entity**

**import exception**

**import util**

**from abc import ABC, abstractmethod**

**def main():**

**connection = dao.connect\_to\_sql\_server()**

**courier\_service = util.CourierServiceDb()**

**while True:**

**print("\nCourier Service Menu:")**

**print("1. Place an order")**

**print("2. Get order status")**

**print("3. Cancel an order")**

**print("4. Get assigned orders")**

**print("5. Add courier staff (Admin)")**

**print("6. Generate report")**

**print("7. Exit")**

**choice = input("Enter your choice: ")**

**if choice == '1':**

**courierID = int(input("Enter your courier ID: "))**

**sender\_name = input("Enter sender's name: ")**

**sender\_address = input("Enter sender's address: ")**

**receiver\_name = input("Enter receiver's name: ")**

**receiver\_address = input("Enter receiver's address: ")**

**weight = float(input("Enter weight: "))**

**tracking\_number = input("Enter tracking number: ")**

**delivery\_date = input("Enter delivery date (YYYY-MM-DD): ")**

**location\_id = int(input("Enter location ID: "))**

**employee\_id = int(input("Enter employee ID: "))**

**service\_id = int(input("Enter service ID: "))**

**courier\_service.insertOrder(courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight,**

**"Processing", tracking\_number, delivery\_date, location\_id, employee\_id,**

**service\_id)**

**elif choice == '2':**

**tracking\_number = input("Enter tracking number: ")**

**status = courier\_service.retrieveDeliveryHistory(tracking\_number)**

**print(f"Order status for tracking number {tracking\_number}: {status}")**

**elif choice == '3':**

**tracking\_number = input("Enter tracking number: ")**

**courier\_service.cancelOrder(tracking\_number)**

**elif choice == '4':**

**employee\_id = input("Enter employee ID: ")**

**couriers = courier\_service.getCouriersByEmployee(employee\_id)**

**print("Couriers handled by Employee ID:", employee\_id)**

**for courier in couriers:**

**print(courier)**

**elif choice == '5':**

**empID = int(input("Enter staff ID:"))**

**name = input("Enter staff name: ")**

**email = input("Enter staff email: ")**

**contact\_number = input("Enter staff contact number: ")**

**role = input("Enter staff role: ")**

**salary = float(input("Enter staff salary: "))**

**courier\_service.addCourierStaff(empID, name, email, contact\_number, role, salary)**

**elif choice == '6':**

**shipment\_status\_report = courier\_service.generateShipmentStatusReport()**

**print("Shipment status report:")**

**for row in shipment\_status\_report:**

**print(row)**

**total\_revenue = courier\_service.generateRevenueReport()**

**print("Total Revenue:", total\_revenue)**

**elif choice == '7':**

**print("Exiting program...")**

**break**

**else:**

**print("Invalid choice. Please enter a number between 1 and 6.")**

**dao.close\_connection(connection)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**Util.py**

**import dao**

**import exception**

**import entity**

**class CourierServiceDb:**

**connection = dao.connect\_to\_sql\_server()**

**def \_\_init\_\_(self):**

**self.connection = dao.connect\_to\_sql\_server()**

**def cancelOrder(self, tracking\_number):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """UPDATE Couriers**

**SET Status = 'Cancelled'**

**WHERE TrackingNumber = ?"""**

**cursor.execute(sql\_query, (tracking\_number,))**

**self.connection.commit()**

**print("Order cancelled successfully.")**

**except Exception as ex:**

**print(f"Error cancelling order: {ex}")**

**finally:**

**cursor.close()**

**def getCouriersByEmployee(self, employee\_id):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT \***

**FROM Couriers**

**WHERE EmployeeID = ?"""**

**cursor.execute(sql\_query, (employee\_id,))**

**couriers = cursor.fetchall()**

**print("Couriers retrieved successfully.")**

**return couriers**

**except exception.InvalidEmployeeIdException as ex:**

**print(f"Error retrieving assigned orders: {ex}")**

**except Exception as ex:**

**print(f"Error retrieving assigned orders: {ex}")**

**finally:**

**cursor.close()**

**def addCourierStaff(self, empID, name, email, contact\_number, role, salary):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """INSERT INTO Employees (EmployeeID,Name, Email, ContactNumber, Role, Salary)**

**VALUES (?,?, ?, ?, ?, ?)"""**

**cursor.execute(sql\_query, (empID, name, email, contact\_number, role, salary))**

**self.connection.commit()**

**print("Courier staff added successfully.")**

**except Exception as ex:**

**print(f"Error adding courier staff: {ex}")**

**finally:**

**cursor.close()**

**def insertOrder(self, courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight, status,**

**tracking\_number, delivery\_date, location\_id, employee\_id, service\_id):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """INSERT INTO Couriers (CourierID, SenderName, SenderAddress, ReceiverName, ReceiverAddress, Weight, Status, TrackingNumber, DeliveryDate, LocationID, EmployeeID, ServiceID)**

**VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"""**

**cursor.execute(sql\_query,**

**(courierID, sender\_name, sender\_address, receiver\_name, receiver\_address, weight, status,**

**tracking\_number, delivery\_date, location\_id, employee\_id, service\_id))**

**self.connection.commit()**

**print("Order inserted successfully.")**

**except Exception as ex:**

**print(f"Error inserting order: {ex}")**

**finally:**

**cursor.close()**

**def updateCourierStatus(self, trackingNumber, newStatus):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """UPDATE Couriers**

**SET Status = ?**

**WHERE TrackingNumber = ?"""**

**cursor.execute(sql\_query, (newStatus, trackingNumber))**

**self.connection.commit()**

**print("Order cancelled successfully.")**

**except exception.TrackingNumberNotFoundException as ex:**

**print(f"Error cancelling order: {ex}")**

**except Exception as ex:**

**print(f"Error cancelling order: {ex}")**

**finally:**

**cursor.close()**

**def retrieveDeliveryHistory(self, trackingNumber):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT \***

**FROM Couriers**

**WHERE TrackingNumber = ?"""**

**cursor.execute(sql\_query, (trackingNumber,))**

**delivery\_history = cursor.fetchall()**

**print("Delivery history retrieved successfully.")**

**return delivery\_history**

**except Exception as ex:**

**print(f"Error retrieving delivery history: {ex}")**

**finally:**

**cursor.close()**

**def generateShipmentStatusReport(self):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT TrackingNumber, Status**

**FROM Couriers"""**

**cursor.execute(sql\_query)**

**shipment\_status\_report = cursor.fetchall()**

**print("Shipment status report generated successfully.")**

**return shipment\_status\_report**

**except Exception as ex:**

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

**finally:**

**cursor.close()**

**def generateRevenueReport(self):**

**try:**

**cursor = self.connection.cursor()**

**sql\_query = """SELECT SUM(Amount) as TotalRevenue**

**FROM Payments"""**

**cursor.execute(sql\_query)**

**total\_revenue = cursor.fetchone()[0]**

**print("Revenue report generated successfully.")**

**return total\_revenue**

**except Exception as ex:**

**print(f"Error generating revenue report: {ex}")**

**finally:**

**cursor.close()**

**Entity.py**

**class User:**

**def \_\_init\_\_(self, userID, userName, email, password, contactNumber, address):**

**self.\_\_userID = userID**

**self.\_\_userName = userName**

**self.\_\_email = email**

**self.\_\_password = password**

**self.\_\_contactNumber = contactNumber**

**self.\_\_address = address**

**def get\_userID(self):**

**return self.\_\_userID**

**def set\_userID(self, userID):**

**self.\_\_userID = userID**

**def get\_userName(self):**

**return self.\_\_userName**

**def set\_userName(self, userName):**

**self.\_\_userName = userName**

**def get\_email(self):**

**return self.\_\_email**

**def set\_email(self, email):**

**self.\_\_email = email**

**def get\_password(self):**

**return self.\_\_password**

**def set\_password(self, password):**

**self.\_\_password = password**

**def get\_contactNumber(self):**

**return self.\_\_contactNumber**

**def set\_contactNumber(self, contactNumber):**

**self.\_\_contactNumber = contactNumber**

**def get\_address(self):**

**return self.\_\_address**

**def set\_address(self, address):**

**self.\_\_address = address**

**def \_\_str\_\_(self):**

**return f"UserID: {self.\_\_userID}, UserName: {self.\_\_userName}, Email: {self.\_\_email}, Password: {self.\_\_password}, ContactNumber: {self.\_\_contactNumber}, Address: {self.\_\_address}"**

**class Courier:**

**def \_\_init\_\_(self, courierID, senderName, senderAddress, receiverName, receiverAddress, weight, status,**

**trackingNumber, deliveryDate, userId):**

**self.\_\_courierID = courierID**

**self.\_\_senderName = senderName**

**self.\_\_senderAddress = senderAddress**

**self.\_\_receiverName = receiverName**

**self.\_\_receiverAddress = receiverAddress**

**self.\_\_weight = weight**

**self.\_\_status = status**

**self.\_\_trackingNumber = trackingNumber**

**self.\_\_deliveryDate = deliveryDate**

**self.\_\_userId = userId**

**def get\_courierID(self):**

**return self.\_\_courierID**

**def set\_courierID(self, courierID):**

**self.\_\_courierID = courierID**

**def get\_senderName(self):**

**return self.\_\_senderName**

**def set\_senderName(self, senderName):**

**self.\_\_senderName = senderName**

**def get\_senderAddress(self):**

**return self.\_\_senderAddress**

**def set\_senderAddress(self, senderAddress):**

**self.\_\_senderAddress = senderAddress**

**def get\_receiverName(self):**

**return self.\_\_receiverName**

**def set\_receiverName(self, receiverName):**

**self.\_\_receiverName = receiverName**

**def get\_receiverAddress(self):**

**return self.\_\_receiverAddress**

**def set\_receiverAddress(self, receiverAddress):**

**self.\_\_receiverAddress = receiverAddress**

**def get\_weight(self):**

**return self.\_\_weight**

**def set\_weight(self, weight):**

**self.\_\_weight = weight**

**def get\_status(self):**

**return self.\_\_status**

**def set\_status(self, status):**

**self.\_\_status = status**

**def get\_trackingNumber(self):**

**return self.\_\_trackingNumber**

**def set\_trackingNumber(self, trackingNumber):**

**self.\_\_trackingNumber = trackingNumber**

**def get\_deliveryDate(self):**

**return self.\_\_deliveryDate**

**def set\_deliveryDate(self, deliveryDate):**

**self.\_\_deliveryDate = deliveryDate**

**def get\_userId(self):**

**return self.\_\_userId**

**def set\_userId(self, userId):**

**self.\_\_userId = userId**

**def \_\_str\_\_(self):**

**return f"CourierID: {self.\_\_courierID}, SenderName: {self.\_\_senderName}, SenderAddress: {self.\_\_senderAddress}, ReceiverName: {self.\_\_receiverName}, ReceiverAddress: {self.\_\_receiverAddress}, Weight: {self.\_\_weight}, Status: {self.\_\_status}, TrackingNumber: {self.\_\_trackingNumber}, DeliveryDate: {self.\_\_deliveryDate}, UserID: {self.\_\_userId}"**

**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 set\_employeeID(self, employeeID):**

**self.\_\_employeeID = employeeID**

**def get\_employeeName(self):**

**return self.\_\_employeeName**

**def set\_employeeName(self, employeeName):**

**self.\_\_employeeName = employeeName**

**def get\_email(self):**

**return self.\_\_email**

**def set\_email(self, email):**

**self.\_\_email = email**

**def get\_contactNumber(self):**

**return self.\_\_contactNumber**

**def set\_contactNumber(self, contactNumber):**

**self.\_\_contactNumber = contactNumber**

**def get\_role(self):**

**return self.\_\_role**

**def set\_role(self, role):**

**self.\_\_role = role**

**def get\_salary(self):**

**return self.\_\_salary**

**def set\_salary(self, salary):**

**self.\_\_salary = salary**

**def \_\_str\_\_(self):**

**return f"EmployeeID: {self.\_\_employeeID}, EmployeeName: {self.\_\_employeeName}, Email: {self.\_\_email}, ContactNumber: {self.\_\_contactNumber}, Role: {self.\_\_role}, Salary: {self.\_\_salary}"**

**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 set\_LocationID(self, LocationID):**

**self.\_\_LocationID = LocationID**

**def get\_LocationName(self):**

**return self.\_\_LocationName**

**def set\_LocationName(self, LocationName):**

**self.\_\_LocationName = LocationName**

**def get\_Address(self):**

**return self.\_\_Address**

**def set\_Address(self, Address):**

**self.\_\_Address = Address**

**def \_\_str\_\_(self):**

**return f"LocationID: {self.\_\_LocationID}, LocationName: {self.\_\_LocationName}, Address: {self.\_\_Address}"**

**class CourierCompany:**

**def \_\_init\_\_(self, companyName):**

**self.\_\_companyName = companyName**

**self.\_\_courierDetails = []**

**self.\_\_employeeDetails = []**

**self.\_\_locationDetails = []**

**def get\_companyName(self):**

**return self.\_\_companyName**

**def set\_companyName(self, companyName):**

**self.\_\_companyName = companyName**

**def add\_courier(self, courier):**

**self.\_\_courierDetails.append(courier)**

**def remove\_courier(self, courier):**

**self.\_\_courierDetails.remove(courier)**

**def add\_employee(self, employee):**

**self.\_\_employeeDetails.append(employee)**

**def remove\_employee(self, employee):**

**self.\_\_employeeDetails.remove(employee)**

**def add\_location(self, location):**

**self.\_\_locationDetails.append(location)**

**def remove\_location(self, location):**

**self.\_\_locationDetails.remove(location)**

**def \_\_str\_\_(self):**

**return f"CompanyName: {self.\_\_companyName}, CourierDetails: {self.\_\_courierDetails}, EmployeeDetails: {self.\_\_employeeDetails}, LocationDetails: {self.\_\_locationDetails}"**

**class Payment:**

**def \_\_init\_\_(self, PaymentID, CourierID, LocationID, Amount, PaymentDate, EmployeeID):**

**self.\_\_PaymentID = PaymentID**

**self.\_\_CourierID = CourierID**

**self.\_\_LocationID = LocationID**

**self.\_\_Amount = Amount**

**self.\_\_PaymentDate = PaymentDate**

**self.\_\_EmployeeID = EmployeeID**

**def get\_PaymentID(self):**

**return self.\_\_PaymentID**

**def set\_PaymentID(self, PaymentID):**

**self.\_\_PaymentID = PaymentID**

**def get\_CourierID(self):**

**return self.\_\_CourierID**

**def set\_CourierID(self, CourierID):**

**self.\_\_CourierID = CourierID**

**def get\_LocationID(self):**

**return self.\_\_LocationID**

**def set\_LocationID(self, LocationID):**

**self.\_\_LocationID = LocationID**

**def get\_Amount(self):**

**return self.\_\_Amount**

**def set\_Amount(self, Amount):**

**self.\_\_Amount = Amount**

**def get\_PaymentDate(self):**

**return self.\_\_PaymentDate**

**def set\_PaymentDate(self, PaymentDate):**

**self.\_\_PaymentDate = PaymentDate**

**def get\_EmployeeID(self):**

**return self.\_\_EmployeeID**

**def set\_EmployeeID(self, EmployeeID):**

**self.\_\_EmployeeID = EmployeeID**

**def \_\_str\_\_(self):**

**return f"PaymentID: {self.\_\_PaymentID}, CourierID: {self.\_\_CourierID}, LocationID: {self.\_\_LocationID}, Amount: {self.\_\_Amount}, PaymentDate: {self.\_\_PaymentDate}, EmployeeID: {self.\_\_EmployeeID}"**

**Dao.py**

**import pyodbc**

**def connect\_to\_sql\_server():**

**try:**

**conn = pyodbc.connect('Driver={SQL Server};'**

**'Server=DESKTOP-A08GADU\SQLEXPRESS01;'**

**'Database=Courier;'**

**'Trusted\_Connection=yes;')**

**print("Connected Successfully")**

**return conn**

**except pyodbc.Error as ex:**

**print(f"Error: {ex}")**

**def close\_connection(conn):**

**conn.close()**

**print("Connection closed.")**

**Exception.py**

**from abc import ABC, abstractmethod**

**class TrackingNumberNotFoundException(Exception):**

**pass**

**class InvalidEmployeeIdException(Exception):**

**pass**

**class ICourierUserService(ABC):**

**@abstractmethod**

**def placeOrder(self, courierObj):**

**pass**

**@abstractmethod**

**def getOrderStatus(self, trackingNumber):**

**pass**

**@abstractmethod**

**def cancelOrder(self, trackingNumber):**

**pass**

**@abstractmethod**

**def getAssignedOrder(self, courierStaffId):**

**pass**

**class ICourierAdminService(ABC):**

**@abstractmethod**

**def addCourierStaff(self, name, contactNumber):**

**pass**

