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)
```

```
"C:\Users\Sugandan\PycharmProjects\p
Enter the order status: Delivered
The order has been delivered.
```

2. 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.")
```

```
"C:\Users\Sugandan\PycharmProjects\pythonPro
Enter the parcel weight in kilograms: 10
The parcel is categorized as Heavy.
```

3. 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.")
```

```
"C:\Users\Sugandan\PycharmProjects\
Enter username: employee
Enter password: employee123
Employee login successful.

Process finished with exit code 0
```

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.

```
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}")
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware foundation\Scripts\python.exe"
Assigned 18 kg shipment to Courier A.
Assigned 8 kg shipment to Courier B.
No suitable courier found for 22 kg shipment to Location Z.
Courier A has the following shipments: [{'weight': 18, 'destination': 'Location X'}]
Courier B has the following shipments: [{'weight': 8, 'destination': 'Location Y'}]
Courier C has the following shipments: []
```

Task 2: Loops and Iteration

5. 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)
```

```
"C:\Users\Sugandan\PycharmProjects\pythonP
Enter customer name: CustomerA
Orders for CustomerA:
Order ID: 1, Status: Processing
Order ID: 3, Status: Cancelled
Order ID: 5, Status: Delivered
```

6. 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.")
```

```
"C:\Users\Sugandan\PycharmProjects\pythonP
Enter customer name: CustomerA
Orders for CustomerA:
Order ID: 1, Status: Processing
Order ID: 3, Status: Cancelled
Order ID: 5, Status: Delivered
```

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.

```
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']}")
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware fo
Tracking history for Parcel ABC123:
2024-03-13 12:21:36 - Location: Warehouse
2024-03-13 12:21:37 - Location: In Transit
2024-03-13 12:21:38 - Location: Local Distribution Center
2024-03-13 12:21:39 - Location: Delivered

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.

```
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.")
```

```
"C:\Users\Sugandan\PycharmProjects\pyth
The nearest available courier for the r

Process finished with exit code 0
```

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.

```
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)
```

```
"C:\Users\Sugandan\PycharmProjects\pythonPro
Enter the parcel tracking number: ABC123
Parcel ABC123 is currently in transit.

Process finished with exit code 0
```

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

```
import re

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.")
```

```
"C:\Users\Sugandan\PycharmProjects\pyth
Invalid name.
Invalid address.
Phone number is valid.

Process finished with exit code 0
```

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.

```
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)
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexa
Enter street address: Rambo St
Enter city: Austin
Enter state: Texas
Enter zip code: 000241
Formatted Address: Rambo St, Austin, TEXAS 000241

Process finished with exit code 0
```

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_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)
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware foundation\Scripts\python.exe" C:\Users\Sug
Enter customer's name: Sugandan
Enter order number: 2012
Enter delivery address: 302,2 nd cross st,pondichery 605110
Order Confirmation Email:

    Subject: Order Confirmation - Order #2012

    Dear Sugandan,

    Thank you for placing an order with us! Your order #2012 has been confirmed.

    Order Details:
    - Order Number: 2012
    - Delivery Address: 302,2 nd cross st,pondichery 605110
    - Expected Delivery Date: 2024-03-15

    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
```

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.

```
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}")
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware foundation
Enter source address: 123 Main Street, CityA, StateA, 12345
Enter destination address: 456 Broadway, CityB, StateB, 67890
Enter parcel weight in kilograms: 2.5
The estimated shipping cost is $10.50

Process finished with exit code 0
```

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

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)
```

```
"C:\Users\Sugandan\PycharmProjects\pytho
Generated Password: \V].1&DviRsp

Process finished with exit code 0
```

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.

```
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)
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware foundati
Enter a address:123 Main Street, CityA, StateA, 12345
Similar addresses to '123 Main Street, CityA, StateA, 12345':
123 Main Street, CityA, StateA, 12345

Process finished with exit code 0
```

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

5. 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 them 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 CMS.dao
import CMS.exception

class CourierServiceDb:
    connection = CMS.dao.connect_to_sql_server()

    def __init__(self):
        self.connection = CMS.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.")
```

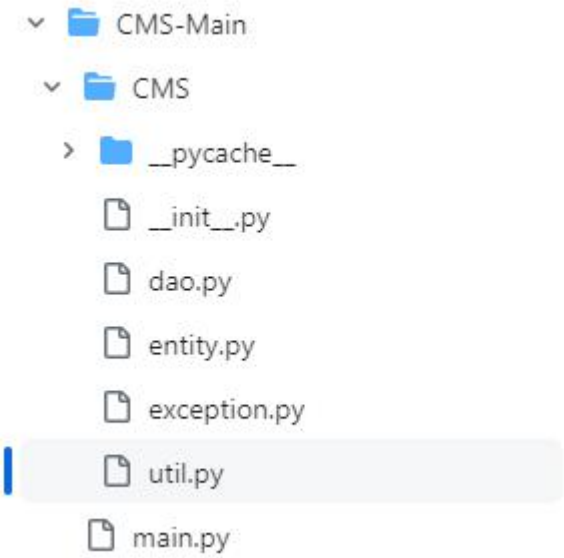
```
        except Exception as ex:
            print(f"Error cancelling order: {ex}")
        finally:
            cursor.close()

    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()
```

File Structure



MAIN.py

```
import CMS




def main():
    connection = CMS.dao.connect_to_sql_server()

    courier_service = CMS.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 7.")
```

```
    CMS.dao.close_connection(connection)


 if __name__ == "__main__":
    main()
```

Util.py

```
import CMS.dao
import CMS.exception

class CourierServiceDb:
    connection = CMS.dao.connect_to_sql_server()

    def __init__(self):
        self.connection = CMS.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
```

```
Connected Successfully

Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 1
Enter your courier ID: 225
Enter sender's name: francis
Enter sender's address: fgftyftf,gfgyfty
Enter receiver's name: john
Enter receiver's address: ftyftrffuyfufkikkkk
Enter weight: 35
Enter tracking number: 321
Enter delivery date (YYYY-MM-DD): 2024-06-06
Enter location ID: 2
Enter employee ID: 3
Enter service ID: 2
Order inserted successfully.
```

```
Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 2
Enter tracking number: 321
Delivery history retrieved successfully.
Order status for tracking number 321: [(225, 'francis', 'fgftyftf,gfgyfty', 'john', 'ftyftrffuyfufkikkkk', Decimal('35.00'), 'Processing', '321',
 '2024-06-06', 2, 3, 2)]
```

```
Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 3
Enter tracking number: 321
Order cancelled successfully.
```

```
Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 4
Enter employee ID: 2
Couriers retrieved successfully.
Couriers handled by Employee ID: 2
(1, 'Rajesh Kumar', '12 Gandhi Nagar, Chennai', 'Ananya Singh', '78 Vindhya Nagar, Coimbatore', Decimal('2.50'), 'In Transit', 'TN123456',
 '2024-03-01', 1, 2, 1)
(2, 'Priya Sharma', '34 Kaveri Street, Bangalore', 'Amit Patel', '56 Krishna Lane, Hyderabad', Decimal('1.80'), 'Delivered', 'TN789012', '2024-03-02',
  1, 2, 2)
(5, 'Sara Khan', '90 Yamuna Road, Delhi', 'David Lee', '67 Forest Lane, Pune', Decimal('2.00'), 'In Transit', 'TN234567', '2024-03-04', 1, 2, 2)
(7, 'Emma Wilson', '45 Lake Avenue, Kolkata', 'Michael Johnson', '23 Park Street, Mumbai', Decimal('3.50'), 'In Transit', 'TN456789', '2024-03-06', 3,
  2, 2)
(8, 'David Lee', '67 Forest Lane, Pune', 'Sara Khan', '90 Yamuna Road, Delhi', Decimal('4.50'), 'In Transit', 'TN567890', '2024-03-07', 3, 2, 1)
```

```
Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 5
Enter staff ID:12
Enter staff name: Rahul
Enter staff email: rahul@123.com
Enter staff contact number: 9699456446
Enter staff role: Manager
Enter staff salary: 35000
Courier staff added successfully.
```

```
Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 6
Shipment status report generated successfully.
Shipment status report:
('TN123456', 'In Transit')
('TN789012', 'Delivered')
('TN345678', 'In Transit')
('TN345679', 'In Transit')
('TN234567', 'In Transit')
('TN345688', 'Delivered')
('TN456789', 'In Transit')
('TN567890', 'In Transit')
('TN678901', 'Delivered')
('123654', 'Cancelled')
('321', 'Cancelled')
Revenue report generated successfully.
Total Revenue: 44601.50
```

```
Courier Service Menu:
1. Place an order
2. Get order status
3. Cancel an order
4. Get assigned orders
5. Add courier staff (Admin)
6. Generate report
7. Exit
Enter your choice: 7
Exiting program...
Connection closed.

Process finished with exit code 0
```