Courier Management System

Coding Task 9

Service implementation

1.Create CourierUserServiceImpl class which implements ICourierUserService interface which holds a variable named companyObj of type CourierCompany.

This variable can be used to access the Object Arrays to access data relevant in method implementations.

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.

In the service implementation phase of the Courier Management System, I created two key service classes: CourierUserServiceImpl and CourierAdminServiceImpl. The CourierUserServiceImpl class implements the ICourierUserService interface and provides core functionalities such as user creation, login, placing a courier order, tracking order status, and order cancellation.

The class interacts with user-defined data structures like ArrayList and HashMap to store and manage user, courier, and payment details efficiently. Furthermore, the CourierAdminServiceImpl class extends CourierUserServiceImpl and implements the ICourierAdminService interface to include advanced features such as viewing all orders and payments, updating courier status, and managing employees, locations, and courier companies.

Both services utilize the CourierCompany object to access and update centralized company-related data, enabling smooth integration and handling of company-level courier operations. Through these implementations, I encapsulated business logic effectively while promoting modular and maintainable code architecture.

Task 9.1 CourierAdminServiceImpl

```
package dao;

import entity.*;

import java.util.*;

public class CourierAdminServiceImp implements ICourierAdminService {

    @Override
    public void viewAllOrders(Map<Integer, List<Courier>> courierMap) {...}
```

```
@Override
public boolean updateCourierStatus(String trackingNumber, String newStatus, Map<Integer, List<Courier>> courierMap) {...}

@Override
public void viewAllPayments(List<Payment> paymentList) {...}

@Override
public void createEmployee(Scanner scanner, List<Employee> employeeList) {...}

@Override
public void createLocation(Scanner scanner, List<Location> locationList) {...}

public void createCourierCompany(Scanner scanner, List<CourierCompany> companyList, List<Employee> employeeList,
                                 List<Location> locationList) {...}
private Employee findEmployeeById(int empID, List<Employee> employeeList) {...}
private Location findLocationById(int locID, List<Location> locationList) {...}

public void assignEmployeeLocationCompanyToCourier(Scanner scanner, Map<Integer, List<Courier>> courierMap,
                                                   List<CourierCompany> companyList) {
```

---

1. void viewAllOrders(Map<Integer, List<Courier>> courierMap)

- **Purpose**: To display all courier orders placed in the system.
- **Input**: courierMap: A map where key = user ID, value = list of couriers placed by the user.
- **Output**: No return value (void), but prints all order details to the console.

```
public class CourierAdminServiceImp implements ICourierAdminService {

    @Override
    public void viewAllOrders(Map<Integer, List<Courier>> courierMap) {
        if (courierMap.isEmpty()) {
            System.out.println("No orders available.");
            return;
        }

        System.out.println("\n--- All Courier Orders ---");
        for (List<Courier> courierList : courierMap.values()) {
            for (Courier courier : courierList) {
                System.out.println(courier);
            }
        }
    }
```

---

2. boolean updateCourierStatus(String trackingNumber, String newStatus, Map<Integer, List<Courier>> courierMap)

- **Purpose**: To update the delivery status of a courier using the tracking number.

- **Input**:
    - trackingNumber: Unique ID of the courier to be updated.
    - newStatus: New status to set (e.g., "Delivered", "In-Transit").
    - courierMap: Map containing all courier data.
- **Output**: Returns true if the update is successful, false if the courier was not found.

```
@Override
public boolean updateCourierStatus(String trackingNumber, String newStatus, Map<Integer, List<Courier>> courierMap) {
    for (List<Courier> courierList : courierMap.values()) {
        for (Courier courier : courierList) {
            if (courier.getTrackingNumber().equals(trackingNumber)) {
                courier.setStatus(newStatus);
                System.out.println("Status updated successfully for Tracking Number: " + trackingNumber);
                return true;
            }
        }
    }
    System.out.println("Tracking Number not found!");
    return false;
}
```

---

3. void viewAllPayments(List<Payment> paymentList)

- **Purpose**: To display details of all payments made by users.
- **Input**:
    - paymentList: A list containing all payment records.
- **Output**: No return value (void), but prints payment info like amount, user, courier ID, etc.

```
@Override
public void viewAllPayments(List<Payment> paymentList) {
    if (paymentList.isEmpty()) {
        System.out.println("No payments recorded.");
        return;
    }

    System.out.println("\n--- All Payments ---");
    for (Payment payment : paymentList) {
        System.out.println(payment);
    }
}
```

---

4. void createEmployee(Scanner scanner, List<Employee> employeeList)

- **Purpose**: To add a new employee into the system.

- **Input**:
    - scanner: Used to take input (like name, ID, role) from admin.
    - employeeList: The list to which the new employee is added.
- **Output**: No return value. New employee is added to the list.

```java
@Override
public void createEmployee(Scanner scanner, List<Employee> employeeList) {
    System.out.print("Enter Employee Name: ");
    String name = scanner.nextLine();

    System.out.print("Enter Email: ");
    String email = scanner.nextLine();

    System.out.print("Enter Contact Number: ");
    String contactNumber = scanner.nextLine();

    System.out.print("Enter Role: ");
    String role = scanner.nextLine();

    System.out.print("Enter Salary: ");
    double salary = scanner.nextDouble();
    scanner.nextLine();

    int employeeID = employeeList.size() + 1;

    Employee newEmployee = new Employee(employeeID, name, email, contactNumber, role, salary);
    employeeList.add(newEmployee);

    System.out.println("Employee added successfully with Employee ID: " + employeeID);
}
```

---

5. void createLocation(Scanner scanner, List<Location> locationList)

- **Purpose**: To add a new delivery location to the system.
- **Input**:
    - scanner: Used for reading input from admin (location name, pin code).
    - locationList: The list to store new locations.
- **Output**: No return value. New location is stored in the list.

```java
@Override
public void createLocation(Scanner scanner, List<Location> locationList) {
    System.out.print("Enter Location Name: ");
    String locationName = scanner.nextLine();

    System.out.print("Enter Address: ");
    String address = scanner.nextLine();

    int locationID = locationList.size() + 1;

    Location newLocation = new Location(locationID, locationName, address);
    locationList.add(newLocation);

    System.out.println("Location added successfully with Location ID: " + locationID);
}
```

---

6. void createCourierCompany(Scanner scanner, List<CourierCompany> companyList, List<Employee> employeeList, List<Location> locationList)

- **Purpose**: To create a new courier company and associate it with employees and locations.
- **Input**:
    - scanner: Used for reading input.
    - companyList: List to store new courier company objects.
    - employeeList: To assign existing employees to the new company.
    - locationList: To assign operating locations for the company.
- **Output**: No return value. Adds new courier company to the list.

```java
public void createCourierCompany(Scanner scanner, List<CourierCompany> companyList, List<Employee> employeeList,
                                 List<Location> locationList) {
    System.out.print("Enter Courier Company Name: ");
    String companyName = scanner.nextLine();

    CourierCompany newCompany = new CourierCompany(companyName);

    while (true) {
        System.out.print("Enter Employee ID to add (or -1 to stop): ");
        int empID = scanner.nextInt();
        scanner.nextLine();
        if (empID == -1) break;

        Employee selectedEmployee = findEmployeeById(empID, employeeList);
        if (selectedEmployee != null) {
            newCompany.addEmployee(selectedEmployee);
            System.out.println("Added Employee: " + selectedEmployee.getEmployeeName());
        } else {
            System.out.println("Invalid Employee ID! Try again.");
        }
    }
```

```
    while (true) {
        System.out.print("Enter Location ID to add (or -1 to stop): ");
        int locID = scanner.nextInt();
        scanner.nextLine();

        if (locID == -1) break;

        Location selectedLocation = findLocationById(locID, locationList);
        if (selectedLocation != null) {
            newCompany.addLocation(selectedLocation);
            System.out.println("Added Location: " + selectedLocation.getLocationName());
        } else {
            System.out.println("Invalid Location ID! Try again.");
        }
    }

    companyList.add(newCompany);
    System.out.println("Courier Company '" + companyName + "' created successfully!");
}
```

---

7. void assignEmployeeLocationCompanyToCourier(Scanner scanner, Map<Integer, List<Courier>> courierMap, List<CourierCompany> companyList)

- **Purpose**: Assign an employee, location, and company to a specific courier.
- **Input**:
  - scanner: For admin to choose courier and assignment details.
  - courierMap: To locate and update specific courier details.
  - companyList: List of available companies to assign.
- **Output**: No return value. Updates the courier details in the map.

```
public void assignEmployeeLocationCompanyToCourier(Scanner scanner, Map<Integer, List<Courier>> courierMap,
                                                   List<CourierCompany> companyList) {

    System.out.print("Enter Courier ID to assign Employee & Location: ");
    int courierId = scanner.nextInt();
    scanner.nextLine();

    Courier selectedCourier = findCourierById(courierId, courierMap);
    if (selectedCourier == null) {
        System.out.println("Invalid Courier ID! Try again.");
        return;
    }

    System.out.println("Available Courier Companies:");
    for (int i = 0; i < companyList.size(); i++) {
        System.out.println((i + 1) + ". " + companyList.get(i).getCompanyName());
    }

    System.out.print("Select a Courier Company (Enter Number): ");
    int companyChoice = scanner.nextInt();
    scanner.nextLine();
    if (companyChoice < 1 || companyChoice > companyList.size()) {
        System.out.println("Invalid choice! Try again.");
        return;
```

```
    CourierCompany selectedCompany = companyList.get(companyChoice - 1);
    String assignedCompany = selectedCompany.getCompanyName();

    System.out.println("\nEmployees in " + selectedCompany.getCompanyName() + ":");
    for (Employee emp : selectedCompany.getEmployeeDetails()) {
        System.out.println("ID: " + emp.getEmployeeID() + " | Name: " + emp.getEmployeeName());
    }

    System.out.print("Enter Employee ID to assign: ");
    int empId = scanner.nextInt();
    scanner.nextLine();

    int assignedEmployee = findEmployee(empId, selectedCompany.getEmployeeDetails());
    if (assignedEmployee == -1) {
        System.out.println("Invalid Employee ID! Try again.");
        return;
    }

    System.out.println("\nLocations in " + selectedCompany.getCompanyName() + ":");
    for (Location loc : selectedCompany.getLocationDetails()) {
        System.out.println("ID: " + loc.getLocationID() + " | Name: " + loc.getLocationName());
    }

    System.out.print("Enter Location ID to assign: ");
    int locId = scanner.nextInt();
    scanner.nextLine();
```

```
    int assignedLocation = findLocation(locId, selectedCompany.getLocationDetails());
    if (assignedLocation == -1) {
        System.out.println("Invalid Location ID! Try again.");
        return;
    }

    selectedCourier.setEmployeeId(assignedEmployee);
    selectedCourier.setLocationId(assignedLocation);
    selectedCourier.setCourierCompanyId(assignedCompany);

    selectedCompany.addCourier(selectedCourier);
    System.out.println("Successfully assigned Employee and Location to Courier ID: " + courierId);
}
```

---

8. void viewCourierCompanyReport(List<CourierCompany> courierCompanyList)

- **Purpose**: To generate and display a report for all courier companies.
- **Input**: courierCompanyList: List of all courier companies.
- **Output**: No return value. Prints summary (no. of couriers, employees, etc.)

```
public void viewCourierCompanyReport(List<CourierCompany> courierCompanyList) {
    if (courierCompanyList.isEmpty()) {
        System.out.println("No courier companies available.");
        return;
    }
    System.out.println("\nCourier Company Report : ");
    for (CourierCompany company : courierCompanyList) {
        System.out.println("\nCompany: " + company.getCompanyName());
        System.out.println("Total Couriers: " + company.getCourierDetails().size());
        System.out.println("Total Employees: " + company.getEmployeeDetails().size());
        System.out.println("Total Locations: " + company.getLocationDetails().size());
```

```
        if (company.getCourierDetails().isEmpty()) {
            System.out.println("No couriers assigned yet.");
        } else {
            System.out.println("\nCourier Details:");
            for (Courier courier : company.getCourierDetails()) {
                System.out.println(courier);
            }
        }
    }
}
```

ICourierUserService

1. **void createUser(Scanner scanner, ArrayList<User> userList)**
   - **Purpose**: To register a new user in the system.
   - **Input**:
     - scanner: For inputting user data (username, password, etc.)
     - userList: List where new user is added.
   - **Output**: No return value. New user is appended to the user list.

```
public class CourierUserServiceImp implements ICourierUserService {

    @Override
    public void createUser(Scanner scanner, ArrayList<User> userList) {
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();

        System.out.print("Enter Email: ");
        String email = scanner.nextLine();

        System.out.print("Enter Password: ");
        String password = scanner.nextLine();

        System.out.print("Enter Address: ");
        String address = scanner.nextLine();

        System.out.print("Enter Phone Number: ");
        String phoneNumber = scanner.nextLine();

        int userId = userList.size() + 1;

        User newUser = new User(userId, name, email, password, phoneNumber, address);
        userList.add(newUser);

        System.out.println("User created successfully! User ID: " + newUser.getUserID());
    }
```

---

2. **User loginUser(Scanner scanner, ArrayList<User> userList)**

- **Purpose**: To authenticate a user by verifying credentials.
- **Input**:
  - scanner: For reading username and password.
  - userList: List of registered users to verify against.
- **Output**: Returns the logged-in User object if credentials match; otherwise, returns null.

```
@Override
public User loginUser(Scanner scanner, ArrayList<User> userList) {
    int attempts = 3;
    while (attempts > 0) {
        System.out.print("Enter Email: ");
        String email = scanner.nextLine();
        System.out.print("Enter Password: ");
        String password = scanner.nextLine();

        for (User user : userList) {
            if (user.getEmail().equals(email) && user.getPassword().equals(password)) {
                System.out.println("Login successful! Welcome, " + user.getUserName());
                return user;
            }
        }
        attempts--;
        if (attempts > 0) {
            System.out.println("Invalid email or password. Attempts left: " + attempts);
        } else {
            System.out.println("Too many failed attempts. Returning to menu.");
        }
    }
    return null;
}
```

---

3. **String placeCourier(User user, Map<Integer, List<Courier>> courierMap, ArrayList<Payment> paymentList, Scanner scanner)**

- **Purpose**: For a user to place a new courier order.
- **Input**:
  - user: The logged-in user.
  - courierMap: Map storing couriers placed by users.
  - paymentList: List to store the payment made for the courier.
  - scanner: For inputting courier and payment details.
- **Output**: Returns a unique **tracking number** for the placed courier.

```java
public String placeCourier(User user, Map<Integer, List<Courier>> courierMap, ArrayList<Payment> paymentList, Scanner scanner) {
    System.out.print("Enter Receiver's Name: ");
    String receiverName = scanner.nextLine();
    System.out.print("Enter Receiver's Address: ");
    String receiverAddress = scanner.nextLine();
    System.out.print("Enter Weight (in kg): ");
    double weight = scanner.nextDouble();
    scanner.nextLine();
    double costPerKg = 50;
    double cost = weight * costPerKg;
    System.out.println("Calculated Cost: ₹" + cost);

    Courier newCourier = new Courier(user.getUserName(), user.getAddress(), receiverName, receiverAddress,
            weight, user.getUserID());
    courierMap.putIfAbsent(user.getUserID(), new ArrayList<>());
    courierMap.get(user.getUserID()).add(newCourier);
    System.out.print("Do you want to proceed with the payment of ₹" + cost + " (yes/no)? ");
    String paymentChoice = scanner.nextLine();

    if ("yes".equalsIgnoreCase(paymentChoice)) {
        long paymentID = System.currentTimeMillis();
        Payment payment = new Payment(paymentID, newCourier.getCourierID(), cost, new Date());
        paymentList.add(payment);
        System.out.println("Payment Successful! Your Tracking Number: " + newCourier.getTrackingNumber());
        return newCourier.getTrackingNumber();
    } else {
        System.out.println("Courier placed without payment.");
        return null;
    }
```

---

4. **String getOrderStatus(String trackingNumber, Map<Integer, List<Courier>> courierMap)**

- **Purpose**: To retrieve the status of a courier using its tracking number.
- **Input**:
    - trackingNumber: The unique identifier for the courier.
    - courierMap: Data source to search the courier.
- **Output**: Returns the current **status** of the courier or "Not Found".

```java
public String getOrderStatus(String trackingNumber, Map<Integer, List<Courier>> courierMap) {
    for (List<Courier> courierList : courierMap.values()) {
        for (Courier courier : courierList) {
            if (courier.getTrackingNumber().equals(trackingNumber)) {
                return "Order Status: " + courier.getStatus();
            }
        }
    }
    return "Tracking Number not found!";
}
```

---

5. **String cancelOrder(String trackingNumber, Map<Integer, List<Courier>> courierMap)**

- **Purpose**: To cancel a courier order before it is shipped/delivered.
- **Input**:

- trackingNumber: The ID of the courier to cancel.
- courierMap: Map holding all couriers.

- **Output**: Returns a confirmation message: **"Cancelled Successfully"** or **"Tracking number not found"**.

```
public String cancelOrder(String trackingNumber, Map<Integer, List<Courier>> courierMap) {

    for (Map.Entry<Integer, List<Courier>> entry : courierMap.entrySet()) {
        List<Courier> courierList = entry.getValue();

        Iterator<Courier> iterator = courierList.iterator();
        while (iterator.hasNext()) {
            Courier courier = iterator.next();
            if (courier.getTrackingNumber().equals(trackingNumber)) {

                if (courier.getStatus().equalsIgnoreCase( anotherString: "Delivered")) {
                    return "Order cannot be canceled as it has already been delivered!";
                }

                iterator.remove();
                return "Order with Tracking Number " + trackingNumber + " has been successfully canceled.";
            }
        }
    }
    return "Tracking Number not found!";
}
```

Conclusion :

The Admin and User interfaces in this courier management system are thoughtfully designed to handle core operations efficiently. Admin methods provide comprehensive control over courier tracking, employee and company management, and system-wide reporting, ensuring smooth backend operations. Meanwhile, user-focused methods enable seamless courier booking, order tracking, and account handling, offering a user-friendly experience. Together, these interfaces form a cohesive and functional system that supports end-to-end courier service management.
