Assignment-Courier Management System
Instructions

- Project submissions should be done through the partcipants' Github repository and the link should be shared with trainers and Hexavarsity.
- Each section builds upon the previous one, and by the end, you will have a **Courier Management System** implemented with a strong focus on SQL, control flow statements, loops, arrays, collections, exception handling, database interaction and Unit Testing.
- Follow object-oriented principles throughout the project. Use classes and objects to model real-world entities, encapsulate data and behavior, and ensure code reusability.
- Throw user defined exceptions from corresponding methods and handled.
- The following Directory structure is to be followed in the application.
    - **entity**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface to showcase functionalities.
        - Create the implementation class for the above interface with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
        - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object(Use method defined in DBPropertyUtil class to get the connection String).
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.

Task1 Database Design
Design a SQL schema for a Courier Management System with tables for Customers, Couriers, Orders, and Parcels. Define the relationships between these tables using appropriate foreign keys.
Requirements:
• Define the Database Schema • Create SQL tables for entities such as **User, Courier, Employee, Location,Payment**

• Define relationships between these tables **(one-to-many, many-to-many, etc.).**

• **Populate Sample Data** • Insert sample data into the tables to simulate real-world scenarios.

User Table:
User
(UserID INT PRIMARY KEY,
Name VARCHAR(255),
Email VARCHAR(255) UNIQUE,

Password VARCHAR(255),
ContactNumber VARCHAR(20),
Address TEXT
);

Courier

(CourierID INT PRIMARY KEY,
SenderName VARCHAR(255),
SenderAddress TEXT,
ReceiverName VARCHAR(255),
ReceiverAddress TEXT,
Weight DECIMAL(5, 2),
Status VARCHAR(50),
TrackingNumber VARCHAR(20) UNIQUE,
DeliveryDate DATE);

CourierServices

(ServiceID INT PRIMARY KEY,
ServiceName VARCHAR(100),
Cost DECIMAL(8, 2));

Employee Table:

(EmployeeID INT PRIMARY KEY,
Name VARCHAR(255),
Email VARCHAR(255) UNIQUE,
ContactNumber VARCHAR(20),
Role VARCHAR(50),
Salary DECIMAL(10, 2));

Location Table:

(LocationID INT PRIMARY KEY,
LocationName VARCHAR(100),
Address TEXT);

Payment Table:

(PaymentID INT PRIMARY KEY,
CourierID INT,
LocationId INT,
Amount DECIMAL(10, 2),
PaymentDate DATE,
FOREIGN KEY (CourierID) REFERENCES Couriers(CourierID),
FOREIGN KEY (LocationID) REFERENCES Location(LocationID));

Task 2: Select,Where

Solve the following queries in the Schema that you have created above

1. List all customers:
2. List all orders for a specific customer:
3. List all couriers:
4. List all packages for a specific order:
5. List all deliveries for a specific courier:
6. List all undelivered packages:
7. List all packages that are scheduled for delivery today:
8. List all packages with a specific status:
9. Calculate the total number of packages for each courier.
10. Find the average delivery time for each courier
11. List all packages with a specific weight range:
12. Retrieve employees whose names contain 'John'
13. Retrieve all courier records with payments greater than $50.

Task 3: GroupBy, Aggregate Functions, Having, Order By, where

14. Find the total number of couriers handled by each employee.
15. Calculate the total revenue generated by each location
16. Find the total number of couriers delivered to each location.
17. Find the courier with the highest average delivery time:
18. Find Locations with Total Payments Less Than a Certain Amount
19. Calculate Total Payments per Location
20. Retrieve couriers who have received payments totaling more than $1000 in a specific location (LocationID = X):
21. Retrieve couriers who have received payments totaling more than $1000 after a certain date (PaymentDate > 'YYYY-MM-DD'):
22. Retrieve locations where the total amount received is more than $5000 before a certain date (PaymentDate > 'YYYY-MM-DD')

Task 4: Inner Join,Full Outer Join, Cross Join, Left Outer Join,Right Outer Join

23. Retrieve Payments with Courier Information
24. Retrieve Payments with Location Information
25. Retrieve Payments with Courier and Location Information
26. List all payments with courier details
27. Total payments received for each courier
28. List payments made on a specific date

29. Get Courier Information for Each Payment

30. Get Payment Details with Location

31. Calculating Total Payments for Each Courier

32. List Payments Within a Date Range

33. Retrieve a list of all users and their corresponding courier records, including cases where there are no matches on either side

34. Retrieve a list of all couriers and their corresponding services, including cases where there are no matches on either side

35. Retrieve a list of all employees and their corresponding payments, including cases where there are no matches on either side

36. List all users and all courier services, showing all possible combinations.

37. List all employees and all locations, showing all possible combinations:

38. Retrieve a list of couriers and their corresponding sender information (if available)

39. Retrieve a list of couriers and their corresponding receiver information (if available):

40. Retrieve a list of couriers along with the courier service details (if available):

41. Retrieve a list of employees and the number of couriers assigned to each employee:

42. Retrieve a list of locations and the total payment amount received at each location:

43. Retrieve all couriers sent by the same sender (based on SenderName).

44. List all employees who share the same role.

45. Retrieve all payments made for couriers sent from the same location.

46. Retrieve all couriers sent from the same location (based on SenderAddress).

47. List employees and the number of couriers they have delivered:

48. Find couriers that were paid an amount greater than the cost of their respective courier services

Scope: Inner Queries, Non Equi Joins, Equi joins,Exist,Any,All

49. Find couriers that have a weight greater than the average weight of all couriers

50. Find the names of all employees who have a salary greater than the average salary:

51. Find the total cost of all courier services where the cost is less than the maximum cost

52. Find all couriers that have been paid for

53. Find the locations where the maximum payment amount was made

54. Find all couriers whose weight is greater than the weight of all couriers sent by a specific sender (e.g., 'SenderName'):

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.

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

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

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.

Task 2: Loops and Iteration

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

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

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.

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

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.

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., ###-###-####).

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.

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.

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.

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.

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.

Following tasks are incremental stages to build an application and should be done in a single project

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, employeeDetails- collection of Employee Objects, locationDetails - collection of Location Objects.

6. Payment Class:

Variables PaymentID long, CourierID long, Amount double, PaymentDate Date

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.
*/
```

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

Task 8: Collections
Scope: ArrayList/Hashmap
Task: Improve the Courier Management System by using Java collections:
1. Create a new model named **CourierCompanyCollection** in **entity package** replacing the **Array of Objects** with List to accommodate dynamic updates in the **CourierCompany** class

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

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.

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**).