Courier Management System

- 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.

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

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

Tasks to be implemented in Utility class

==========

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.