Car Rental System

Instructions

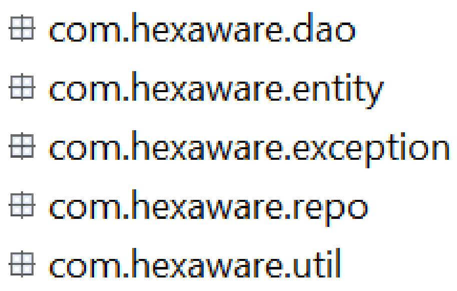
- Project submissions should **be a single zip file containing source code and an additional document with output screenshot**
- Each section builds upon the previous one, and by the end, you will have a comprehensive **Virtual Art Gallery** implemented with a strong focus on **SQL schema design**, **control flow statements, loops, arrays, collections, exception handling ,database interaction** using **JDBC** and **Unit Testing** using **JUnit**
- Remember to provide clear instructions, sample code, and any necessary resources or databases for each assignment.
- 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 handle in the main method**.**

Key Functionalities:

1. **Customer Management**
    - Add new customers, Update customer information, Retrieve customer details.
2. **Car Management:**
    - Add new cars to the system, Update car availability, Retrieve car information.
3. **Lease Management**
    - Create daily or monthly leases for customers.
    - Calculate the total cost of a lease based on the type (Daily or Monthly) and the number of days or months.
4. **Payment Handling:**
    - Record payments for leases.
    - Retrieve payment history for a customer.
    - Calculate the total revenue from payments.

Instructions

- Project submissions should **be a single zip file containing source code and an additional document with output screenshot.**
- Each section builds upon the previous one, and by the end, you will have a comprehensive **Ecommerce Application** implemented with a strong focus on **SQL schema design**, **control flow statements, loops, arrays, collections, exception handling**, database **interaction** using **JDBC** and **Unit Testing** using **Junit.**
- Remember to provide clear instructions, sample code, and any necessary resources or databases for each assignment.
- 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 handle in the main method**.**
- The following **package structure** to be followed in the application.



com.hexaware.dao
com.hexaware.entity
com.hexaware.exception
com.hexaware.repo
com.hexaware.util

- **com.hexaware.entity**
  - Create entity classes in this package. All entity class should not have any business logic.
- **com.hexaware.dao**
  - Create Service Provider interface to showcase functionalities
  - Create the implementation class for the above interface with db interaction.
- **com.hexaware.exception**
  - Create user defined exceptions in this package and handle exceptions whenever needed.
- **com.hexaware.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)**.Top of Form

Create following tables in MySQL Schema with appropriate java class and write the Junit test case for the Car Rental application.

SQL Schema:

1. **Vehicle Table:**
   - vehicleID (Primary Key)
   - make
   - model
   - year
   - dailyRate
   - status (available, notAvailable)
   - passengerCapacity (for cars)
   - engineCapacity (for motorcycles)
2. **Customer Table:**
   - customerID (Primary Key)
   - firstName
   - lastName
   - email
   - phoneNumber
3. **Lease Table:**
   - leaseID (Primary Key)

- vehicleID (Foreign Key referencing Vehicle Table)
- customerID (Foreign Key referencing Customer Table)
- startDate
- endDate
- type (to distinguish between DailyLease and MonthlyLease)

4. **Payment Table:**
    - paymentID (Primary Key)
    - leaseID (Foreign Key referencing Lease Table)
    - paymentDate
    - amount

Create the following java classes, interface and refer to the SQL table for the attributes in java class.

5. **Vehicle (Base Class):**
    - Properties: vehicleID, make, model, year, dailyRate, available
    - Methods: getters and setters
        1. **Car (Subclass of Vehicle):**
            1. Additional Properties: passengerCapacity
            2. Methods: getters and setters
        2. **Motorcycle (Subclass of Vehicle):**
            1. Additional Properties: engineCapacity
            2. Methods: getters and setters
6. **Customer:**
    - Properties: customerID, firstName, lastName, email, phoneNumber
    - Methods: getters and setters
7. **Lease (Abstract Class):**
    - Properties: leaseID, vehicle, customer, startDate, endDate
    - Methods: calculateLeaseCost(), generateLeaseAgreement()
8. **DailyLease (Class):** (Extends Lease abstract class)
    - Methods: calculateLeaseCost(), generateLeaseAgreement()
9. **MonthlyLease (Class):** (Extends Lease abstract class)
    - Methods: calculateLeaseCost(), generateLeaseAgreement()
10. **Payment:**
    - Properties: paymentID, rental, paymentDate, amount
    - Methods: recordPayment()
11. Create Interface for **ICarLeaseRepository** and add following methods which interact with database.
    - **Car Management**
        1. void addCar(Car car);
        2. void removeCar(int carID);
        3. List<Car> listAvailableCars();
        4. List<Car> listRentedCars();
        5. Car findCarById(int carID);
    - **Customer Management**
        1. void addCustomer(Customer customer);

2. void removeCustomer(int customerID);
3. List<Customer> listCustomers();
4. Customer findCustomerById(int customerID);

- **Lease Management**
  1. Lease createLease(int customerID, int carID, Date startDate, Date endDate);
  2. void returnCar(int leaseID);
  3. List<Lease> listActiveLeases();
  4. List<Lease> listLeaseHistory();
- **Payment Handling**
  1. void recordPayment(Lease lease, double amount);
  2. List<Payment> listPayments(Lease lease);

12. Implement the above interface in a class called **ICarLeaseRepositoryImpl**.
13. Connect your application to the SQL database and write code to establish a connection to your SQL database.
    - Create a utility class **DBConnection** in a package **com.hexaware.util** 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.
    - Create a utility class **PropertyUtil** which contains a static method named **getPropertyString()** which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.
14. Create the exceptions in package **com.hexaware.myexceptions** and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,
    - **CarNotFoundException**: throw this exception when user enters an invalid car id which doesn't exist in db.
    - **LeaseNotFoundException**: throw this exception when user enters an invalid lease id which doesn't exist in db.
    - **CustomerrNotFoundException**: throw this exception when user enters an invalid customer id which doesn't exist in db.
15. Create JUnit test cases for **Car Lease System** are essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of JUnit test cases for various components of the system:
    - Write test case to test car created successfully or not.
    - Write test case to test lease is created successfully or not.
    - Write test case to test lease is retrieved successfully or not.
    - write test case to test exception is thrown correctly or not when customer id or car id or lease id not found in database.