Ecommerce Application

Instructions

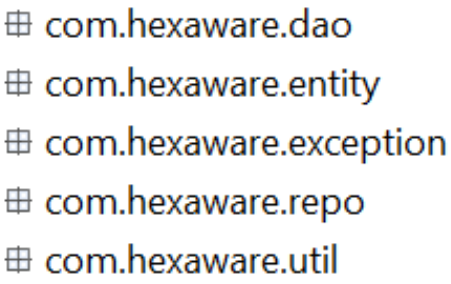
- Project submissions should **be a single zip file containing source code and an additional document with output screenshot**
- Each section builds upon the previous one, and by the end, you will have a comprehensive **Virtual Art Gallery** implemented with a strong focus on **SQL schema design**, **control 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, and retrieve customer information and order details,
2. **Product Management:**
    - Users can view a list of available products, add and delete products.
3. **Cart Management:**
    - Users can add and remove products to their shopping cart.
4. **Order Management:**
    - Users can place orders, which include product details, quantities, and shipping information.
    - The order total is calculated based on the cart contents.

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.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)**.

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

SQL Tables:

1. **customers** table:
   - customer_id (Primary Key)
   - name
   - email
   - password
2. **products** table:
   - product_id (Primary Key)
   - name
   - price
   - description
   - stockQuantity
3. **cart** table:
   - cart_id (Primary Key)
   - customer_id (Foreign Key)
   - product_id (Foreign Key)
   - quantity
4. **orders** table:

- order_id (Primary Key)
- customer_id (Foreign Key)
- order_date
- total_price
- shipping_address

5. **order_items** table (to store order details):
    - order_item_id (Primary Key)
    - order_id (Foreign Key)
    - product_id (Foreign Key)
    - quantity

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

6. **Customer Management:**
    - Create a **User** class with attributes for user details (e.g., **CustomerId**, **name**, **email**, **password**).
    - Implement user authentication and registration functionality.
    - Users can log in with their credentials or register for a new account.
7. **Product Management:**
    - Create a **Product** class with attributes (e.g., **productId**, **name**, **price**, **description**).
    - Implement methods for browsing and searching products.
8. **Cart Management:**
    - Create a **Cart** class to represent the customer's shopping cart.
    - Allow customers to add and remove products from the cart.
    - Calculate the total price in the cart.
9. **Order Management:**
    - Create an **Order** class to represent customer orders.
    - Implement order placement functionality.
    - Store order details, including products, quantity, total price, and shipping information.
10. **Interfaces:**
    - Define an **OrderProcessorRepository** interface with methods for adding/removing products to/from the cart and placing orders. Following methods will interact with database.
        1. **boolean createProduct(Product product)**
        2. **boolean createCustomer(Customer customer)**
        3. **boolean deleteProduct(productId)**
        4. **boolean deleteCustomer(customerId)**
        5. **boolean addToCart(Customer customer, Product product, int quantity):** insert the product in cart.
        6. **boolean removeFromCart(Customer customer, Product product):** delete the product in cart.
        7. **List<Map<Product,quantity>> getAllFromCart(Customer customer):** list the product in cart.

8. **boolean placeOrder(Customer customer, List<Map<Product,quantity>>, string shippingAddress):** should update order table and orderItems table.
9. **List<Map<Product,quantity>> getOrdersByCustomer(customerId)**
10. **List<Map<Product,quantity>> getOrdersByOrderId(orderId)**

11. Implement the above interface in a class called **OrderProcessorRepositoryImpl**.

Connect your application to the SQL database **:**

12. 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.
13. 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,
    - **CustomerNotFoundException**: throw this exception when user enters an invalid customer id which doesn't exist in db
    - **ProductNotFoundException**: throw this exception when user enters an invalid product id which doesn't exist in db
    - **OrderNotFoundException**: throw this exception when user enters an invalid order id which doesn't exist in db
14. Create class named **EcomApp** with main method in com.hexware.app Trigger all the methods in service implementation class by user choose operation from the following menu.
    1. Register Customer.
    2. Create Product.
    3. Delete Product.
    4. Add to cart.
    5. View cart.
    6. Place order.
    7. View Customer Order
15. Create JUnit test cases for **Ecommerce 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 Product created successfully or not.
    - Write test case to test product is added to cart successfully or not.
    - Write test case to test product is ordered successfully or not.
    - write test case to test exception is thrown correctly or not when customer id or product id not found in database.