TechShop, an electronic gadgets shop

Instructions:

- Submitting assignments should be a single file or through git hub link shared with trainer and hexavarsity.
- Each assignment builds upon the previous one, and by the end, you will have a comprehensive application implemented in Java/C#/Python with a strong focus on SQL schema design, control flow statements, loops, arrays, collections, and database interaction.
- Follow object-oriented principles throughout the Java programming assignments. Use classes and objects to model real-world entities, encapsulate data and behavior, and ensure code reusability.
- Throw user defined exception from method and handle in the main method.
- The following Directory structure is to be followed in the application.
    - **entity/model**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface/abstract class to showcase functionalities.
        - Create the implementation class for the above interface/abstract class 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.
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.

You are working as a database administrator for a fictional company named "TechShop," which sells electronic gadgets. TechShop maintains data related to their products, customers, and orders. Your task is to design and implement a database for TechShop based on the following requirements:

Database Tables:

1. **Customers:**
    - CustomerID (Primary Key)
    - FirstName
    - LastName
    - Email
    - Phone
    - Address

2. **Products:**
    - ProductID (Primary Key)
    - ProductName
    - Description
    - Price
3. **Orders:**
    - OrderID (Primary Key)
    - CustomerID (Foreign Key referencing Customers)
    - OrderDate
    - TotalAmount
4. **OrderDetails:**
    - OrderDetailID (Primary Key)
    - OrderID (Foreign Key referencing Orders)
    - ProductID (Foreign Key referencing Products)
    - Quantity
5. **Inventory**
    - InventoryID (Primary Key)
    - ProductID (Foreign Key referencing Products)
    - QuantityInStock
    - LastStockUpdate

Task:1. Database Design:

1. Create the database named "TechShop"
2. Define the schema for the Customers, Products, Orders, OrderDetails and Inventory tables based on the provided schema.
3. Create an ERD (Entity Relationship Diagram) for the database.
4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.
5. Insert at least 10 sample records into each of the following tables.
    a. Customers
    b. Products
    c. Orders
    d. OrderDetails

e. Inventory

Tasks 2: Select, Where, Between, AND, LIKE:

1. Write an SQL query to retrieve the names and emails of all customers.
2. Write an SQL query to list all orders with their order dates and corresponding customer names.
3. Write an SQL query to insert a new customer record into the "Customers" table. Include customer information such as name, email, and address.
4. Write an SQL query to update the prices of all electronic gadgets in the "Products" table by increasing them by 10%.
5. Write an SQL query to delete a specific order and its associated order details from the "Orders" and "OrderDetails" tables. Allow users to input the order ID as a parameter.
6. Write an SQL query to insert a new order into the "Orders" table. Include the customer ID, order date, and any other necessary information.
7. Write an SQL query to update the contact information (e.g., email and address) of a specific customer in the "Customers" table. Allow users to input the customer ID and new contact information.
8. Write an SQL query to recalculate and update the total cost of each order in the "Orders" table based on the prices and quantities in the "OrderDetails" table.
9. Write an SQL query to delete all orders and their associated order details for a specific customer from the "Orders" and "OrderDetails" tables. Allow users to input the customer ID as a parameter.
10. Write an SQL query to insert a new electronic gadget product into the "Products" table, including product name, category, price, and any other relevant details.
11. Write an SQL query to update the status of a specific order in the "Orders" table (e.g., from "Pending" to "Shipped"). Allow users to input the order ID and the new status.
12. Write an SQL query to calculate and update the number of orders placed by each customer in the "Customers" table based on the data in the "Orders" table.

Task 3. Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write an SQL query to retrieve a list of all orders along with customer information (e.g., customer name) for each order.
2. Write an SQL query to find the total revenue generated by each electronic gadget product. Include the product name and the total revenue.

3. Write an SQL query to list all customers who have made at least one purchase. Include their names and contact information.
4. Write an SQL query to find the most popular electronic gadget, which is the one with the highest total quantity ordered. Include the product name and the total quantity ordered.
5. Write an SQL query to retrieve a list of electronic gadgets along with their corresponding categories.
6. Write an SQL query to calculate the average order value for each customer. Include the customer's name and their average order value.
7. Write an SQL query to find the order with the highest total revenue. Include the order ID, customer information, and the total revenue.
8. Write an SQL query to list electronic gadgets and the number of times each product has been ordered.
9. Write an SQL query to find customers who have purchased a specific electronic gadget product. Allow users to input the product name as a parameter.
10. Write an SQL query to calculate the total revenue generated by all orders placed within a specific time period. Allow users to input the start and end dates as parameters.

Task 4. Subquery and its type:

1. Write an SQL query to find out which customers have not placed any orders.
2. Write an SQL query to find the total number of products available for sale.
3. Write an SQL query to calculate the total revenue generated by TechShop.
4. Write an SQL query to calculate the average quantity ordered for products in a specific category. Allow users to input the category name as a parameter.
5. Write an SQL query to calculate the total revenue generated by a specific customer. Allow users to input the customer ID as a parameter.
6. Write an SQL query to find the customers who have placed the most orders. List their names and the number of orders they've placed.
7. Write an SQL query to find the most popular product category, which is the one with the highest total quantity ordered across all orders.
8. Write an SQL query to find the customer who has spent the most money (highest total revenue) on electronic gadgets. List their name and total spending.
9. Write an SQL query to calculate the average order value (total revenue divided by the number of orders) for all customers.

10. Write an SQL query to find the total number of orders placed by each customer and list their names along with the order count.

Implement OOPs

Task 1: Classes and Their Attributes:

You are working as a software developer for TechShop, a company that sells electronic gadgets. Your task is to design and implement an application using Object-Oriented Programming (OOP) principles to manage customer information, product details, and orders. Below are the classes you need to create:

Customers Class:

Attributes:

- CustomerID (int)
- FirstName (string)
- LastName (string)
- Email (string)
- Phone (string)
- Address (string)

Methods:

- CalculateTotalOrders(): Calculates the total number of orders placed by this customer.
- GetCustomerDetails(): Retrieves and displays detailed information about the customer.
- UpdateCustomerInfo(): Allows the customer to update their information (e.g., email, phone, or address).

Products Class:

Attributes:

- ProductID (int)
- ProductName (string)
- Description (string)
- Price (decimal)

Methods:

- GetProductDetails(): Retrieves and displays detailed information about the product.
- UpdateProductInfo(): Allows updates to product details (e.g., price, description).
- IsProductInStock(): Checks if the product is currently in stock.

Orders Class:

Attributes:

- OrderID (int)
- Customer (Customer) - Use composition to reference the Customer who placed the order.
- OrderDate (DateTime)
- TotalAmount (decimal)

Methods:

- CalculateTotalAmount() - Calculate the total amount of the order.
- GetOrderDetails(): Retrieves and displays the details of the order (e.g., product list and quantities).
- UpdateOrderStatus(): Allows updating the status of the order (e.g., processing, shipped).
- CancelOrder(): Cancels the order and adjusts stock levels for products.

OrderDetails Class:

Attributes:

- OrderDetailID (int)
- Order (Order) - Use composition to reference the Order to which this detail belongs.
- Product (Product) - Use composition to reference the Product included in the order detail.
- Quantity (int)

Methods:

- CalculateSubtotal() - Calculate the subtotal for this order detail.
- GetOrderDetailInfo(): Retrieves and displays information about this order detail.
- UpdateQuantity(): Allows updating the quantity of the product in this order detail.
- AddDiscount(): Applies a discount to this order detail.

Inventory class:

Attributes:

- InventoryID(int)
- Product (Composition): The product associated with the inventory item.
- QuantityInStock: The quantity of the product currently in stock.
- LastStockUpdate

Methods:

- GetProduct(): A method to retrieve the product associated with this inventory item.
- GetQuantityInStock(): A method to get the current quantity of the product in stock.
- AddToInventory(int quantity): A method to add a specified quantity of the product to the inventory.
- RemoveFromInventory(int quantity): A method to remove a specified quantity of the product from the inventory.
- UpdateStockQuantity(int newQuantity): A method to update the stock quantity to a new value.
- IsProductAvailable(int quantityToCheck): A method to check if a specified quantity of the product is available in the inventory.
- GetInventoryValue(): A method to calculate the total value of the products in the inventory based on their prices and quantities.
- ListLowStockProducts(int threshold): A method to list products with quantities below a specified threshold, indicating low stock.
- ListOutOfStockProducts(): A method to list products that are out of stock.

- ListAllProducts(): A method to list all products in the inventory, along with their quantities.

Task 2: Class Creation:

- Create the classes (Customers, Products, Orders, OrderDetails and Inventory) with the specified attributes.
- Implement the constructor for each class to initialize its attributes.
- Implement methods as specified.

Task 3: Encapsulation:

- Implement encapsulation by making the attributes private and providing public properties (getters and setters) for each attribute.
- Add data validation logic to setter methods (e.g., ensure that prices are non-negative, quantities are positive integers).

Task 4: Composition:

Ensure that the Order and OrderDetail classes correctly use composition to reference Customer and Product objects.

- Orders Class with Composition:
  - In the Orders class, we want to establish a composition relationship with the Customers class, indicating that each order is associated with a specific customer.
  - In the Orders class, we've added a private attribute customer of type Customers, establishing a composition relationship. The Customer property provides access to the Customers object associated with the order.
- OrderDetails Class with Composition:
  - Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered.
  - In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the product in the order detail.
- Customers and Products Classes:
  - The Customers and Products classes themselves may not have direct composition relationships with other classes in this scenario. However, they serve as the basis for composition relationships in the Orders and OrderDetails classes, respectively.
- Inventory Class:
  - The Inventory class represents the inventory of products available for sale. It can have composition relationships with the Products class to indicate which products are in the inventory.

Task 5: Exceptions handling

- **Data Validation**:

- Challenge: Validate user inputs and data from external sources (e.g., user registration, order placement).
- Scenario: When a user enters an invalid email address during registration.
- Exception Handling: Throw a custom InvalidDataException with a clear error message.

- **Inventory Management**:
  - Challenge: Handling inventory-related issues, such as selling more products than are in stock.
  - Scenario: When processing an order with a quantity that exceeds the available stock.
  - Exception Handling: Throw an InsufficientStockException and update the order status accordingly.
- **Order Processing:**
  - Challenge: Ensuring the order details are consistent and complete before processing.
  - Scenario: When an order detail lacks a product reference.
  - Exception Handling: Throw an IncompleteOrderException with a message explaining the issue.
- **Payment Processing:**
  - Challenge: Handling payment failures or declined transactions.
  - Scenario: When processing a payment for an order and the payment is declined.
  - Exception Handling: Handle payment-specific exceptions (e.g., PaymentFailedException) and initiate retry or cancellation processes.
- **File I/O (e.g., Logging):**
  - Challenge: Logging errors and events to files or databases.
  - Scenario: When an error occurs during data persistence (e.g., writing a log entry).
  - Exception Handling: Handle file I/O exceptions (e.g., IOException) and log them appropriately.
- **Database Access:**
  - Challenge: Managing database connections and queries.
  - Scenario: When executing a SQL query and the database is offline.
  - Exception Handling: Handle database-specific exceptions (e.g., SqlException) and implement connection retries or failover mechanisms.
- **Concurrency Control:**
  - Challenge: Preventing data corruption in multi-user scenarios.
  - Scenario: When two users simultaneously attempt to update the same order.
  - Exception Handling: Implement optimistic concurrency control and handle ConcurrencyException by notifying users to retry.
- **Security and Authentication:**
  - Challenge: Ensuring secure access and handling unauthorized access attempts.
  - Scenario: When a user tries to access sensitive information without proper authentication.
  - Exception Handling: Implement custom AuthenticationException and AuthorizationException to handle security-related issues.

Task 6: Collections

- **Managing Products List:**

- Challenge: Maintaining a list of products available for sale (List<Products>).
- Scenario: Adding, updating, and removing products from the list.
- Solution: Implement methods to add, update, and remove products. Handle exceptions for duplicate products, invalid updates, or removal of products with existing orders.

- **Managing Orders List:**
  - Challenge: Maintaining a list of customer orders (List<Orders>).
  - Scenario: Adding new orders, updating order statuses, and removing canceled orders.
  - Solution: Implement methods to add new orders, update order statuses, and remove canceled orders. Ensure that updates are synchronized with inventory and payment records.
- **Sorting Orders by Date:**
  - Challenge: Sorting orders by order date in ascending or descending order.
  - Scenario: Retrieving and displaying orders based on specific date ranges.
  - Solution: Use the List<Orders> collection and provide custom sorting methods for order date. Consider implementing SortedList if you need frequent sorting operations.
- **Inventory Management with SortedList:**
  - Challenge: Managing product inventory with a SortedList based on product IDs.
  - Scenario: Tracking the quantity in stock for each product and quickly retrieving inventory information.
  - Solution: Implement a SortedList<int, Inventory> where keys are product IDs. Ensure that inventory updates are synchronized with product additions and removals.
- **Handling Inventory Updates:**
  - Challenge: Ensuring that inventory is updated correctly when processing orders.
  - Scenario: Decrementing product quantities in stock when orders are placed.
  - Solution: Implement a method to update inventory quantities when orders are processed. Handle exceptions for insufficient stock.
- **Product Search and Retrieval:**
  - Challenge: Implementing a search functionality to find products based on various criteria (e.g., name, category).
  - Scenario: Allowing customers to search for products.
  - Solution: Implement custom search methods using LINQ queries on the List<Products> collection. Handle exceptions for invalid search criteria.
- **Duplicate Product Handling:**
  - Challenge: Preventing duplicate products from being added to the list.
  - Scenario: When a product with the same name or SKU is added.
  - Solution: Implement logic to check for duplicates before adding a product to the list. Raise exceptions or return error messages for duplicates.
- **Payment Records List:**
  - Challenge: Managing a list of payment records for orders (List<PaymentClass>).
  - Scenario: Recording and updating payment information for each order.
  - Solution: Implement methods to record payments, update payment statuses, and handle payment errors. Ensure that payment records are consistent with order records.
- **OrderDetails and Products Relationship:**
  - Challenge: Managing the relationship between OrderDetails and Products.

- Scenario: Ensuring that order details accurately reflect the products available in the inventory.
- Solution: Implement methods to validate product availability in the inventory before adding order details. Handle exceptions for unavailable products.

Task 7: Database Connectivity

- Implement a DatabaseConnector class responsible for establishing a connection to the "TechShopDB" database. This class should include methods for opening, closing, and managing database connections.
- Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.

1: Customer Registration

Description: When a new customer registers on the TechShop website, their information (e.g., name, email, phone) needs to be stored in the database.

Task: Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.

2: Product Catalog Management

Description: TechShop regularly updates its product catalog with new items and changes in product details (e.g., price, description). These changes need to be reflected in the database.

Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency.

3: Placing Customer Orders

Description: Customers browse the product catalog and place orders for products they want to purchase. The orders need to be stored in the database.

Task: Implement an order processing system. Use database connectivity to record customer orders, update product quantities in inventory, and calculate order totals.

4: Tracking Order Status

Description: Customers and employees need to track the status of their orders. The order status information is stored in the database.

Task: Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

5: Inventory Management

Description: TechShop needs to manage product inventory, including adding new products, updating stock levels, and removing discontinued items.

Task: Create an inventory management system with database connectivity. Implement features for adding new products, updating quantities, and handling discontinued products.

6: Sales Reporting

Description: TechShop management requires sales reports for business analysis. The sales data is stored in the database.

Task: Design and implement a reporting system that retrieves sales data from the database and generates reports based on specified criteria.

7: Customer Account Updates

Description: Customers may need to update their account information, such as changing their email address or phone number.

Task: Implement a user profile management feature with database connectivity to allow customers to update their account details. Ensure data validation and integrity.

8: Payment Processing

Description: When customers make payments for their orders, the payment details (e.g., payment method, amount) must be recorded in the database.

Task: Develop a payment processing system that interacts with the database to record payment transactions, validate payment information, and handle errors.

9: Product Search and Recommendations

Description: Customers should be able to search for products based on various criteria (e.g., name, category) and receive product recommendations.

Task: Implement a product search and recommendation engine that uses database connectivity to retrieve relevant product information.